

# EN3021 Group Project - Progress as of OCT17

210415N Navarathne D.M.G.B.  
210594J Senevirathne I.U.B.  
210609M Silva M.K.Y.U.N.

October 17, 2024

**Current status:** 45% Complete

## Progress this week

We started the following parts of the project for the week OCT10 - OCT17:

- Almost completed the integration of the modules into the processor
- Following modules were completed
  - ALU control
  - Immediate generator

## ALU Control

The ALU control was made to get the input from the control signal ALUOp, and func3 and func7 from the instruction to control the ALU. Further this passes on a branch instruction type to the ALU to get a branch flag according to the type.

Table 1: Instruction Set and corresponding ALU operation

Inst.	Type	ALUOp	F3	F7	OP
ADD	R	10	000	0	ADD
SUB	R	10	000	1	SUB
SLL	R	10	001	0	SLL
SLT	R	10	010	0	SLT
SLTU	R	10	011	0	SLTU
XOR	R	10	100	0	XOR
SRL	R	10	101	0	SRL
SRA	R	10	101	1	SRA
OR	R	10	110	0	OR
AND	R	10	111	0	AND
ADDI	I	10	000	X	ADD
SLTI	I	10	010	X	SLT
SLTIU	I	10	011	X	SLTU
XORI	I	10	100	X	XOR
ORI	I	10	110	X	OR
ANDI	I	10	111	X	AND

Inst.	Type	ALUOp	F3	F7	OP
SLLI	I	10	001	X	SLL
SRLI	I	10	101	X	SRL
SRAI	I	10	101	X	SRA
JALR	I	10	000	X	ADD
LUI	U	XX	XXX	X	-
AUIPC	U	XX	XXX	X	-
JAL	UJ	XX	XXX	X	-
SB	S	00	000	X	ADD
SH	S	00	001	X	ADD
SW	S	00	010	X	ADD
LB	I	00	000	X	ADD
LH	I	00	001	X	ADD
LW	I	00	010	X	ADD
LBU	I	00	100	X	ADD
LHU	I	00	101	X	ADD
BEQ	SB	01	000	X	SUB
BNE	SB	01	001	X	SUB
BLT	SB	01	100	X	SLT
BGE	SB	01	101	X	SLT
BLTU	SB	01	110	X	SLTU
BGEU	SB	01	111	X	SLTU
FENCE	I	00	000	X	-
ECALL	I	00	000	X	-
EBREAK	I	00	000	X	-

```

1 //
2 // +-----+
3 // | Instruction set |
4 // +-----+
5 // | Inst. | Type | ALUOp | F3 | F7 | OP |
6 // +-----+
7 // | ADD   | R   | 10    | 000 | 0 | ADD |
8 // | SUB   | R   | 10    | 000 | 1 | SUB |
9 // | SLL   | R   | 10    | 001 | 0 | SLL |
10 // | SLT   | R   | 10    | 010 | 0 | SLT |
11 // | SLTU  | R   | 10    | 011 | 0 | SLTU |
12 // | XOR   | R   | 10    | 100 | 0 | XOR |
13 // | SRL   | R   | 10    | 101 | 0 | SRL |
14 // | SRA   | R   | 10    | 101 | 1 | SRA |
15 // | OR    | R   | 10    | 110 | 0 | OR  |
16 // | AND   | R   | 10    | 111 | 0 | AND |
17 // +-----+
18 // | ADDI  | I   | 10    | 000 | X | ADD |
19 // | SLTI  | I   | 10    | 010 | X | SLT |
20 // | SLTIU | I   | 10    | 011 | X | SLTU |
21 // | XORI  | I   | 10    | 100 | X | XOR |
22 // | ORI   | I   | 10    | 110 | X | OR  |
23 // | ANDI  | I   | 10    | 111 | X | AND |
24 // | SLLI  | I   | 10    | 001 | X | SLL |
25 // | SRLI  | I   | 10    | 101 | X | SRL |
26 // | SRAI  | I   | 10    | 101 | X | SRA |
27 // | JALR  | I   | 10    | 000 | X | ADD |
28 // +-----+
29 // | LUI   | U   | XX    | XXX | X | -   |
30 // | AUIPC | U   | XX    | XXX | X | -   |

```

```

31 // | JAL | UJ | XX | XXX | X | - |
32 // +-----+-----+-----+-----+-----+-----+
33 // | SB | S | 00 | 000 | X | ADD |
34 // | SH | S | 00 | 001 | X | ADD |
35 // | SW | S | 00 | 010 | X | ADD |
36 // +-----+-----+-----+-----+-----+-----+
37 // | LB | I | 00 | 000 | X | ADD |
38 // | LH | I | 00 | 001 | X | ADD |
39 // | LW | I | 00 | 010 | X | ADD |
40 // | LBU | I | 00 | 100 | X | ADD |
41 // | LHU | I | 00 | 101 | X | ADD |
42 // +-----+-----+-----+-----+-----+-----+
43 // | BEQ | SB | 01 | 000 | X | SUB |
44 // | BNE | SB | 01 | 001 | X | SUB |
45 // | BLT | SB | 01 | 100 | X | SLT |
46 // | BGE | SB | 01 | 101 | X | SLT |
47 // | BLTU | SB | 01 | 110 | X | SLTU |
48 // | BGEU | SB | 01 | 111 | X | SLTU |
49 // +-----+-----+-----+-----+-----+-----+
50 // | FENCE | I | 00 | 000 | X | - |
51 // | ECALL | I | 00 | 000 | X | - |
52 // | EBREAK | I | 00 | 000 | X | - |
53 // +-----+-----+-----+-----+-----+-----+
54
55
56 module alu_ctrl(
57     input [1:0] ALUOp,
58     input [2:0] FUNC3,
59     input FUNC7,
60     output reg [3:0] ALUCTRL,
61     output reg [2:0] BRANCHCONDITION
62 );
63
64
65 parameter ADD = 4'b0000, SUB = 4'b0001, SLL = 4'b0010,
66           SRL = 4'b0011, SRA = 4'b0100, AND = 4'b0101,
67           OR = 4'b0110, XOR = 4'b0111, SLT = 4'b1000,
68           SLTU = 4'b1001;
69
70
71 always@(*) begin
72     if (ALUOp == 2'b10) begin // ----- R type / I type
73         case (FUNC3)
74             3'b000:
75                 begin
76                     if ((ALUOp == 2'b00)&(FUNC7 == 1'b1))
77                         ALUCTRL <= SUB;
78                     else
79                         ALUCTRL <= ADD;
80                 end
81
82             3'b001:
83                 ALUCTRL <= SLL;
84
85             3'b010:
86                 ALUCTRL <= SLT;
87
88             3'b011:
89                 ALUCTRL <= SLTU;
90
91             3'b100:
92                 ALUCTRL <= XOR;
93
94             3'b101:
95                 begin
96                     if (FUNC7 == 1'b0)
97                         ALUCTRL <= SRL;
98                     else

```

```

99         ALUCTRL <= SRA;
100     end
101
102     3'b110:
103         ALUCTRL <= OR;
104
105     3'b111:
106         ALUCTRL <= AND;
107
108     default:
109         ALUCTRL <= 4'bx;
110
111     endcase
112 end else if (ALUOp == 2'b00) begin
113     ALUCTRL <= ADD;
114
115 end else if (ALUOp == 2'b01) begin
116     case(FUNC3[2:1])
117     2'b00:
118         ALUCTRL <= SUB;
119
120     2'b10:
121         ALUCTRL <= SLT;
122
123     2'b11:
124         ALUCTRL <= SLTU;
125
126     default:
127         ALUCTRL <= 4'bx;
128
129     endcase
130 end else begin
131     ALUCTRL <= 4'bx;
132
133 end
134
135 if (ALUOp == 2'b01) begin
136     BRANCHCONDITION <= FUNC3;
137 end else begin
138     BRANCHCONDITION <= 3'bx;
139 end
140
141
142 end
143 endmodule

```

## Adding Branch flags to the ALU

The ALU was modified to indicate the satisfaction of a branch condition, from the result that was calculated from the existing ALU Operations.

```

1 module alu(
2     input  [31:0] A, B,
3     input  [3:0]  CTRL,
4     input  [2:0]  BRANCHCONDITION,
5     output reg [31:0] OUT,
6     output reg    BRANCHFLAG
7 );
8
9 reg  [31:0] A_ALU, B_ALU;
10 wire [31:0] SUM, BAND, BXOR;
11 reg  CIN;
12
13 // CLA
14 cla32 cla32_0(.A(A_ALU), .B(B_ALU^{4{CIN}}), .CIN(CIN), .OF(), .SUM(SUM), .BAND(BAND)
, .BXOR(BXOR));

```

```

15
16 parameter ADD = 4'b0000, SUB = 4'b0001, SLL = 4'b0010,
17         SRL = 4'b0011, SRA = 4'b0100, AND = 4'b0101,
18         OR  = 4'b0110, XOR = 4'b0111, SLT = 4'b1000,
19         SLTU = 4'b1001;
20
21 parameter BEQ = 3'b000, BNE = 3'b001, BLT = 3'b100,
22         BGE = 3'b101, BLTU= 3'b110, BGEU= 3'b111;
23
24
25 always@(*) begin
26     case (CTRL)
27     ADD:
28         begin
29             CIN <= 1'b0;
30             A_ALU <= A;
31             B_ALU <= B;
32             OUT <= SUM;
33         end
34
35     SUB:
36         begin
37             CIN <= 1'b1;
38             A_ALU <= A;
39             B_ALU <= B;
40             OUT <= SUM;
41         end
42
43     AND:
44         begin
45             CIN <= 1'bx;
46             A_ALU <= A;
47             B_ALU <= B;
48             OUT <= BAND;
49         end
50
51     OR:
52         begin
53             CIN <= 1'bx;
54             A_ALU <= A;
55             B_ALU <= B;
56             OUT <= A|B;
57         end
58
59     XOR:
60         begin
61             CIN <= 1'bx;
62             A_ALU <= A;
63             B_ALU <= B;
64             OUT <= BXOR;
65         end
66
67     SLL:
68         begin
69             CIN <= 1'bx;
70             A_ALU <= 32'bx;
71             B_ALU <= 32'bx;
72             OUT <= A << B;
73         end
74
75     SRL:
76         begin
77             CIN <= 1'bx;
78             A_ALU <= 32'bx;
79             B_ALU <= 32'bx;
80             OUT <= A >> B;
81         end
82

```

```

83     SRA:
84         begin
85             CIN <= 1'bx;
86             A_ALU <= 32'bx;
87             B_ALU <= 32'bx;
88             OUT <= A >>> B;
89         end
90
91     SLT:
92         begin
93             CIN <= 1'b1;
94             A_ALU <= A;
95             B_ALU <= B;
96             OUT <= {31'b0, SUM[31]};
97         end
98
99     SLTU:
100        begin
101            CIN <= 1'bx;
102            A_ALU <= 32'bx;
103            B_ALU <= 32'bx;
104            OUT <= (A < B) ? 32'b1 : 32'b0;
105        end
106
107    default:
108        begin
109            A_ALU <= 32'bx;
110            B_ALU <= 32'bx;
111            CIN <= 1'bx;
112            OUT <= 32'bx;
113        end
114    endcase
115
116
117    case(BRANCHCONDITION)
118        BEQ:
119            BRANCHFLAG <= |SUM;
120
121        BNE:
122            BRANCHFLAG <= ~|SUM;
123
124        BLT:
125            BRANCHFLAG <= OUT[0];
126
127        BGE:
128            BRANCHFLAG <= ~OUT[0];
129
130        BLTU:
131            BRANCHFLAG <= OUT[0];
132
133        BGEU:
134            BRANCHFLAG <= ~OUT[0];
135
136        default:
137            BRANCHFLAG <= 1'bx;
138    endcase
139 end
140
141 endmodule

```

## Immediate Generator

The immediate generator formats the immediate into the proper form by using the instruction.

Table 2: Instruction Set and Immediate format

Inst.	Type	ALUOp	F3	F7	Immediate Location	Format
ADDI	I	10	000	X	[31:20]	[11:0]
SLTI	I	10	010	X	[31:20]	[11:0]
SLTIU	I	10	011	X	[31:20]	[11:0]
XORI	I	10	100	X	[31:20]	[11:0]
ORI	I	10	110	X	[31:20]	[11:0]
ANDI	I	10	111	X	[31:20]	[11:0]
SLLI	I	10	001	X	[24:20] (no ext)	[4:0]
SRLI	I	10	101	X	[24:20] (no ext)	[4:0]
SRAI	I	10	101	X	[24:20] (no ext)	[4:0]
JALR	I	10	000	X	[31:20]	[11:0]
LUI	U	XX	XXX	X	[31:12] (up)	[31:12]
AUIPC	U	XX	XXX	X	[31:12] (up)	[31:12]
JAL	UJ	XX	XXX	X	[31:12] (up)	[20][10:1][11][19:12]
SB	S	00 (?)	000	X	[31:25][11:7]	[11:5][4:0]
SH	S	00 (?)	001	X	[31:25][11:7]	[11:5][4:0]
SW	S	00 (?)	010	X	[31:25][11:7]	[11:5][4:0]
LB	I	00 (?)	000	X	[31:20]	[11:0]
LH	I	00 (?)	001	X	[31:20]	[11:0]
LW	I	00 (?)	010	X	[31:20]	[11:0]
LBU	I	00 (?)	100	X	[31:20]	[11:0]
LHU	I	00 (?)	101	X	[31:20]	[11:0]
BEQ	SB	01	000	X	[31:25][11:7]	[12][10:5][4:1][11]
BNE	SB	01	001	X	[31:25][11:7]	[12][10:5][4:1][11]
BLT	SB	01	100	X	[31:25][11:7]	[12][10:5][4:1][11]
BGE	SB	01	101	X	[31:25][11:7]	[12][10:5][4:1][11]
BLTU	SB	01	110	X	[31:25][11:7]	[12][10:5][4:1][11]
BGEU	SB	01	111	X	[31:25][11:7]	[12][10:5][4:1][11]

```

1 //
2 //
+-----+
3 // | Instruction set (Which use imm. gen.)
4 // |
+-----+-----+-----+-----+-----+-----+
5 // | Inst. | Type | ALUOp | F3 | F7 | Immediate location | Format
6 // |
+-----+-----+-----+-----+-----+-----+
7 // | ADDI | I | 10 | 000 | X | [31:20] | [11:0]
8 // | SLTI | I | 10 | 010 | X | [31:20] | [11:0]

```

9	//		SLTIU		I		10		011		X		[31:20]		[11:0]	
10	//		XORI		I		10		100		X		[31:20]		[11:0]	
11	//		ORI		I		10		110		X		[31:20]		[11:0]	
12	//		ANDI		I		10		111		X		[31:20]		[11:0]	
13	//		SLLI		I		10		001		X		[24:20] (no ext)		[4:0]	
14	//		SRLI		I		10		101		X		[24:20] (no ext)		[4:0]	
15	//		SRAI		I		10		101		X		[24:20] (no ext)		[4:0]	
16	//		JALR		I		10		000		X		[31:20]		[11:0]	
17	//	+-----+-----+-----+-----+-----+-----+-----+-----+														
18	//		LUI		U		XX		XXX		X		[31:12] (up)		[31:12]	
19	//		AUIPC		U		XX		XXX		X		[31:12] (up)		[31:12]	
20	//		JAL		UJ		XX		XXX		X		[31:12] (up)			
21	//	+-----+-----+-----+-----+-----+-----+-----+-----+														
22	//		SB		S		00 (?)		000		X		[31:25] [11:7]		[11:5] [4:0]	
23	//		SH		S		00 (?)		001		X		[31:25] [11:7]		[11:5] [4:0]	
24	//		SW		S		00 (?)		010		X		[31:25] [11:7]		[11:5] [4:0]	
25	//	+-----+-----+-----+-----+-----+-----+-----+-----+														
26	//		LB		I		00 (?)		000		X		[31:20]		[11:0]	
27	//		LH		I		00 (?)		001		X		[31:20]		[11:0]	
28	//		LW		I		00 (?)		010		X		[31:20]		[11:0]	
29	//		LBU		I		00 (?)		100		X		[31:20]		[11:0]	
30	//		LHU		I		00 (?)		101		X		[31:20]		[11:0]	
31	//	+-----+-----+-----+-----+-----+-----+-----+-----+														
32	//		BEQ		SB		01		000		X		[31:25] [11:7]		[12] [10:5] [4:1] [11]	
33	//		BNE		SB		01		001		X		[31:25] [11:7]		[12] [10:5] [4:1] [11]	
34	//		BLT		SB		01		100		X		[31:25] [11:7]		[12] [10:5] [4:1] [11]	
35	//		BGE		SB		01		101		X		[31:25] [11:7]		[12] [10:5] [4:1] [11]	
36	//		BLTU		SB		01		110		X		[31:25] [11:7]		[12] [10:5] [4:1] [11]	
37	//		BGEU		SB		01		111		X		[31:25] [11:7]		[12] [10:5] [4:1] [11]	
38	//	+-----+-----+-----+-----+-----+-----+-----+-----+														
39																
40	module immediate_gen(															
41	input [31:0] INSTRUCTION,															



```

42     output reg [31:0] IMMEDIATE_OUT
43 );
44
45 always@(*) begin
46     case(INSTRUCTION[6:0])
47         7'b0110111:          // LUI
48             begin
49                 IMMEDIATE_OUT[31:12] <= INSTRUCTION[31:21];
50                 IMMEDIATE_OUT[11:0]  <= 12'b0;
51             end
52
53         7'b0010111:          // AUIPC
54             begin
55                 IMMEDIATE_OUT[31:12] <= INSTRUCTION[31:21];
56                 IMMEDIATE_OUT[11:0]  <= 12'b0;
57             end
58
59         7'b1101111:          // JAL
60             begin
61                 IMMEDIATE_OUT[31:20] <= {12{INSTRUCTION[31]}};
62                 IMMEDIATE_OUT[19:12] <= INSTRUCTION[19:12];
63                 IMMEDIATE_OUT[11]    <= INSTRUCTION[20];
64                 IMMEDIATE_OUT[10:1]  <= INSTRUCTION[30:21];
65                 IMMEDIATE_OUT[0]     <= 1'b0;
66             end
67
68         7'b1100111:          // JALR
69             begin
70                 IMMEDIATE_OUT[31:12] <= {20{INSTRUCTION[31]}};
71                 IMMEDIATE_OUT[11:0]  <= INSTRUCTION[31:20];
72             end
73
74         7'b0010011:          // SLLI / SRLI / SRAI
75             begin
76                 if ((INSTRUCTION[14:12] == 001) | (INSTRUCTION[14:12] == 001)) begin
77                     // SLLI / SRLI / SRAI
78                     IMMEDIATE_OUT[31:5] <= 27'b0;
79                     IMMEDIATE_OUT[4:0]  <= INSTRUCTION[24:20];
80
81                 end else begin
82                     // Other I type
83                     IMMEDIATE_OUT[31:12] <= {12{INSTRUCTION[31]}};
84                     IMMEDIATE_OUT[11:0]  <= INSTRUCTION[31:20];
85                 end
86             end
87
88         7'b0100011:          // Store
89             begin
90                 IMMEDIATE_OUT[31:12] <= {20{INSTRUCTION[31]}};
91                 IMMEDIATE_OUT[11:5]  <= INSTRUCTION[31:25];
92                 IMMEDIATE_OUT[4:0]   <= INSTRUCTION[11:7];
93             end
94
95         7'b0000011:          // Load
96             begin
97                 IMMEDIATE_OUT[31:12] <= {20{INSTRUCTION[31]}};
98                 IMMEDIATE_OUT[11:0]  <= INSTRUCTION[31:20];
99             end
100
101         7'b1100011:          // Branch
102             begin
103                 IMMEDIATE_OUT[31:13] <= {19{INSTRUCTION[31]}};
104                 IMMEDIATE_OUT[12]    <= INSTRUCTION[31];
105                 IMMEDIATE_OUT[11]    <= INSTRUCTION[7];
106                 IMMEDIATE_OUT[10:5]  <= INSTRUCTION[30:25];
107                 IMMEDIATE_OUT[4:1]   <= INSTRUCTION[11:8];
108                 IMMEDIATE_OUT[0]     <= 1'b0;
109             end

```

```

110
111     default:
112         IMMEDIATE_OUT[31:0] <= 32'bx;
113
114     endcase
115 end
116 endmodule

```

## Integration into the processor

The completed modules were integrated into the processor. As the control unit is still under completion, this wasn't added. but the rest were added.

```

1 module processor_main(
2     input CLK
3 );
4
5 // Program counter
6 wire [31:0] INS_ADDR, INS;
7 wire [31:0] INS_ADDR_IN;
8
9 ProgramCounter pc(
10     .instruct_address(INS_ADDR),
11     .clk(CLK),
12     .instruct_address_in(INS_ADDR_IN)
13 );
14
15
16
17
18 // ---- Instruction memory ----
19
20 InstructMem imem(
21     .Pro_count(INS_ADDR),
22     .inst_out(INS)
23 );
24
25
26
27
28 // ---- Control signals ----
29
30 wire CTRL_BRANCH, CTRL_MEMREAD, CTRL_MEMTOREG, CTRL_MEMWRITE, CTRL_ALUSRC, CTRL_REGWRITE;
31 wire [1:0] CTRL_ALUOP;
32
33 ///////////////////////////////////////////////////
34 // Contol unit goes here //
35 ///////////////////////////////////////////////////
36
37
38
39 // ---- Registry file ----
40
41 wire [31:0] RS1_DATA, RS2_DATA, RD_DATA;
42
43 Register_File regfile(
44     .Read_reg01(INS[19:15]), // rs1 addr
45     .Read_reg02(INS[24:20]), // rs2 addr
46     .Write_reg(INS[11:7]), // rd addr
47     .Write_data(RD_DATA), // data written to register
48     .Read_data01(RS1_DATA), // rs1 data
49     .Read_data02(RS2_DATA), // rs2 data
50     .write_signal(CTRL_REGWRITE), // control regwrite
51     .clk(CLK) // clk
52 );
53

```

```

54
55
56
57 // ---- Sign extension ----
58
59 wire [31:0] IMM_EXT; // Sign extended / instruction correctly formatted immediate
60
61 immediate_gen immgen(
62     .INSTRUCTION(INS),
63     .IMMEDIATE_OUT(IMM_EXT)
64 );
65
66
67
68
69 // ---- ALU control ----
70
71 wire [3:0] ALU_OPCMD;
72 wire [2:0] ALU_BRANCHCMD;
73
74 alu_ctrl aluctrl(
75     .ALUOp(CTRL_ALUOP),
76     .FUNC3(INS[14:12]),
77     .FUNC7(INS[30]),
78     .ALUCTRL(ALU_OPCMD),
79     .BRANCHCONDITION(ALU_BRANCHCMD)
80 );
81
82
83
84
85 // ---- ALU ----
86
87 wire [31:0] B_RS2_IMM, ALU_OUT;
88 wire ALU_BRANCHFLAG;
89 assign B_RS2_IMM = (CTRL_ALUSRC == 1'b0) ? RS2_DATA : IMM_EXT;
90
91 alu alu(
92     .A(RS1_DATA),
93     .B(B_RS2_IMM),
94     .CTRL(ALU_OPCMD),
95     .BRANCHCONDITION(ALU_BRANCHCMD),
96     .OUT(ALU_OUT),
97     .BRANCHFLAG(ALU_BRANCHFLAG)
98 );
99
100
101
102
103 // ---- Data memory ----
104
105 wire [31:0] DMEM_OUT;
106
107 DataMem dmem(
108     .clk(CLK),
109     .write_en(CTRL_MEMWRITE),
110     .read_en(CTRL_MEMREAD),
111     .address(ALU_OUT), // Address bus width is 32 bits
112     .data_in(RS2_DATA), // Data bus width is 32 bits
113     .data_out(DMEM_OUT)
114 );
115
116 assign RD_DATA = (CTRL_MEMTOREG == 1'b1) ? DMEM_OUT : ALU_OUT;
117
118
119
120
121 // ---- Address calculation ----

```

```

122
123 wire [31:0] PC_ORDINARY, PC_BRANCH;
124
125 cla32 add_0(.A(INS_ADDR), .B(32'd4), .CIN(1'b0), .OF(), .SUM(PC_ORDINARY), .BAND(),
    .BXOR());
126
127 wire [31:0] IMM_EXT_SHIFTED;
128 assign IMM_EXT_SHIFTED = IMM_EXT << 1'b1;
129
130 cla32 add_1(.A(INS_ADDR), .B(IMM_EXT_SHIFTED), .CIN(1'b0), .OF(), .SUM(PC_BRANCH), .
    BAND(), .BXOR());
131
132 assign INS_ADDR_IN = ((CTRL_BRANCH & ALU_BRANCHFLAG) == 1'b0) ? PC_ORDINARY : PC_BRANCH;
133
134 endmodule

```

## Members' Contribution

- Nawarathna D.M.G.B - Refining control logic, program counter increment logic
- Senavirathne I.U.B. - Control unit
- Silva M.K.Y.U.N - ALU control, Immediate generation