NISHITH SHAH (SJSU ID:011478948) **DATE:1ST DEC'16**

5 – STAGE PIPELINED NIOS II SOFT PROCESSOR FOR THE EXECUTION OF ASSEMBLY CODE OF DOT PRODUCT OF 2 VECTORS WITH DATA FORWARDING:

MY APPROACH:

AVECTOR = 16'h0000
N = 19
DOT_PRODUCT = 18

BVECTOR = 16'h000A
LOOP = 5
STOP => pc = 5'h0E
(continuous looping at STOP)

INSTRUCTION	*Rd*	*Rs*	*Rt*	*IMMEDIATE DATA , ADDRESS OR OPX*	*OPCODES*
MOVIA R2, **AVECTOR**	00010	00010	-	0000000000000000	000101
MOVIA R3, **BVECTOR**	00011	00011	-	0000000000001001	000101
MOVIA R4, **N**	00100	00100	-	0000000000010011	000101
LDW R4,0(R4)	00100	00100	-	0000000000000000	010001
ADD R5,RO,RO	00101	00000	00000	00000011111	111010
LOOP: LDW R6,0(R2)	00110	00010	-	0000000000000000	010001
LDW R7, 0(R3)	00111	00011	-	0000000000000000	010001
MUL R8, R6, R7	01000	00110	00111	00000011011	111010
ADD R5, R5, R8	00101	00101	01000	00000011111	111010
ADDI R2, R2, 1	00010	00010	-	0000000000000001	000100
ADDI R3, R3, 1	00011	00011	-	0000000000000001	000100
SUBI R4, R4, 1	00100	00100	-	0000000000000001	000111
BGT R4, R0, **LOOP**	00100	00000	-	0000000000000101	010000
STW R5, **DOT_PRODUCT**(R0)	00101	00000	-	0000000000010010	001111
BR **STOP**	-	-	-	00000000000000000000000000	000110

ALU OPERATION	*BIT SEQUENCE*
ADDITION	00 = 0
MOVE	01 = 1
SUBTRACTION	10 = 2
MULTIPLICATION	11 = 3

ALU SOURCE 1	*ALU SOURCE 2*	*BIT SEQUENCE*
REGISTER	REGISTER	00 = 0
-----------	IMMEDIATE	01 = 1
-----------	ADDRESS	10 = 2

NISHITH SHAH (SJSU ID:011478948)   DATE:1ST DEC'16

VERILOG CODE:

DESIGN FILE:

```
module nios_ii(loc,alu_output,clk);

	//IN-OUT DECLARATION
	output		loc, alu_output;
	input		clk;

	//IN-OUT TYPE DECLARATION
	reg[31:0]	loc, alu_output;
	wire		clk;

	//MEMORY DECLARATION (INSTRUCTION MEMORY, REGISTER BANK, DATA MEMORY)
	reg[31:0]	insmem_48[0:14],
			regmem_48[0:31],
			datamem_48[0:31];
	reg[1:0]	exit;

	//INTERMEDIATE REGISTERS
	reg[4:0]	pc_48;
	reg[31:0]	Inst_F_48, Inst_D_48;

	reg[5:0]	opcode_D_48;
	reg[10:0]	opx_D_48;
	reg[15:0]	imm_D_48;
	reg[31:0]	Imm_SignExt_D_48;
	reg[4:0]	Rd_D_48;
	reg[31:0]	Des_addr_D_48;
	reg[4:0]	Rs_D_48;
	reg[31:0]	Alu_ip1_D_48;
	reg[4:0]	Rt_D_48;
	reg[31:0]	Alu_ip2_D_48;
	reg[2:0]	Alu_opx_D_48;
	reg[1:0]	Alu_src2_D_48;
	reg		Reg_wr_D_48, Mem_wr_D_48;
	reg		fwd_req_D_48;
	reg[1:0]	Br_D_48;

	reg[5:0]	opcode_E_48;
	reg[10:0]	opx_E_48;
	reg[31:0]	Des_addr_E_48;
	reg[1:0]	Br_E_48;
	reg[2:0]	Alu_opx_E_48;
	reg[1:0]	Alu_src2_E_48, Br_En_E_48;
	reg		Reg_wr_E_48, Mem_wr_E_48;
	reg[4:0]	Rd_E_48,Rs_E_48, Rt_E_48;
	reg[31:0]	Alu_ip1_E_48;
	reg[31:0]	Alu_ip2_E_48;
	reg[31:0]	Alu_src1_48,
			Alu_src2_48;
	reg[15:0]	imm_E_48;
	reg[31:0]	Imm_SignExt_E_48, Alu_output_E_48;
	reg		fwd_req_E_48;
```

```
        reg[5:0]        opcode_M_48;
        reg             Reg_wr_M_48, Mem_wr_M_48;
        reg[31:0]       Des_addr_M_48;
        reg[4:0]        Rd_M_48, Rs_M_48, Rt_M_48;
        reg[1:0]        Br_En_M_48;
        reg[31:0]       Alu_output_M_48, Alu_output_M1_48;
        reg             fwd_req_M_48;

        reg[31:0]       Alu_output_W_48;
        reg[5:0]        opcode_W_48;
        reg[4:0]        Rd_W_48, Rs_W_48;
        reg             Reg_wr_W_48, Mem_wr_W_48, fwd_req_W_48;

initial
        begin
        //INITIALIZE ALL VARIABLES
        pc_48=0; Inst_F_48=0; exit=0;

        Inst_D_48=0;opcode_D_48=0;opx_D_48=0;imm_D_48=0;Imm_SignExt_D_48=0;Des_addr_D_48=0;

        Rd_D_48=0;Alu_ip1_D_48=0;Rs_D_48=0;Alu_ip2_D_48=0;Rt_D_48=0;Alu_opx_D_48=0;Alu_src2_D_48=0;
        Alu_src1_48=0;Alu_src2_48=0;Reg_wr_D_48=0;Mem_wr_D_48=0;Br_D_48=0;fwd_req_D_48=0;

        opcode_E_48=0;Des_addr_E_48=0;Br_E_48=0;Alu_opx_E_48=0;Alu_src2_E_48=0;Br_En_E_48=0;
        Reg_wr_E_48=0;Mem_wr_E_48=0;Rd_E_48=0;Rs_E_48=0;Rt_E_48=0;Alu_ip1_E_48=0;Alu_ip2_E_48=0;
        imm_E_48=0;Imm_SignExt_E_48=0;Alu_output_E_48=0;fwd_req_E_48=0;opx_E_48=0;

        opcode_M_48=0;Reg_wr_M_48=0;Mem_wr_M_48=0;Rd_M_48=0;Rs_M_48=0;Rt_M_48=0;
        Des_addr_M_48=0;Br_En_M_48=0;fwd_req_M_48=0;Alu_output_M_48=0;Alu_output_M1_48=0;

        Alu_output_W_48=0;opcode_W_48=0;Rd_W_48=0;Rs_W_48=0;Reg_wr_W_48=0;Mem_wr_W_48=0;
        fwd_req_W_48=0;

//PROGRAM COUNTER (INITIALLY ZERO TO POINT AT FIRST LINE OF CODE)
        pc_48 = 5'h00;

//INSTRUCTION MEMORY (WILL CONTAIN THE WHOLE PROGRAM IN ORDER OF EXECUTION)
        insmem_48[0] = 32'h10800005;  //MOVIA R2,AVECTOR
        insmem_48[1] = 32'h18C00245;  //MOVIA R3,BVECTOR
        insmem_48[2] = 32'h210004C5;  //MOVIA R4,N
        insmem_48[3] = 32'h21000011;  //LDW R4,0(R4)
        insmem_48[4] = 32'h280007FA;  //ADD R5,R0,R0
        insmem_48[5] = 32'h30800011;  //LDW R6,0(R2)
        insmem_48[6] = 32'h38C00011;  //LDW R7,0(R3)
        insmem_48[7] = 32'h418E06FA;  //MUL R8,R6,R7
        insmem_48[8] = 32'h295007FA;  //ADD R5,R5,R8
        insmem_48[9] = 32'h10800044;  //ADDI R2,R2,1
        insmem_48[10] = 32'h18C00044; //ADDI R3,R3,1
        insmem_48[11] = 32'h21000047; //SUBI R4,R4,1
        insmem_48[12] = 32'h20000150; //BGT R4,R0,LOOP
        insmem_48[13] = 32'h2800048F; //STW R5,DOT_PRODUCT(R0)
        insmem_48[14] = 32'h00000006; //BR STOP

//REGISTER MEMORY (WILL CONTAIN THE INTERMEDIATE DATA'S THAT NEED TO BE TEMPORARILY STORED)
        regmem_48[0] = 32'h00000000;
        regmem_48[1] = 32'h00000000;
        regmem_48[2] = 32'h00000000;
        regmem_48[3] = 32'h00000000;
        regmem_48[4] = 32'h00000000;
        regmem_48[5] = 32'h00000000;
        regmem_48[6] = 32'h00000000;
        regmem_48[7] = 32'h00000000;
        regmem_48[8] = 32'h00000000;
        regmem_48[9] = 32'h00000000;
        regmem_48[10] = 32'h00000000;
        regmem_48[11] = 32'h00000000;
        regmem_48[12] = 32'h00000000;
        regmem_48[13] = 32'h00000000;
        regmem_48[14] = 32'h00000000;
        regmem_48[15] = 32'h00000000;

        //DATA MEMORY
        datamem_48[0] = 32'h00000000;       //0     A vector
        datamem_48[1] = 32'h00000001;       //1
        datamem_48[2] = 32'h00000001;       //1
        datamem_48[3] = 32'h00000004;       //4
        datamem_48[4] = 32'h00000007;       //7
        datamem_48[5] = 32'h00000008;       //8
        datamem_48[6] = 32'h00000009;       //9
        datamem_48[7] = 32'h00000004;       //4
        datamem_48[8] = 32'h00000008;       //8

        datamem_48[9] = 32'h00000000;       //0     B vector
        datamem_48[10] = 32'h00000002;      //2
        datamem_48[11] = 32'h00000002;      //2
        datamem_48[12] = 32'h00000009;      //9
        datamem_48[13] = 32'h00000005;      //5
        datamem_48[14] = 32'h00000007;      //7
        datamem_48[15] = 32'h00000008;      //8
        datamem_48[16] = 32'h00000009;      //9
        datamem_48[17] = 32'h00000006;      //6

        //ANSWER STORAGE
        datamem_48[18] = 32'h00000000;

        //LOCATION OF N
        datamem_48[19] = 32'h00000009;
        end
//------------------------------------------------------------FETCH STAGE------------------------------------------------------------//
always@(pc_48)
        begin
                Inst_F_48 = insmem_48[pc_48];
        end

always@(posedge clk)
        begin
                Inst_D_48 <= Inst_F_48;

                if(Br_En_E_48 == 2'b00 && pc_48 == 5'h0c)       //CONDITIONAL BRANCH "BGT R4,R0,LOOP"
                begin
                        pc_48 <= pc_48;
                        exit = 0;
```

```
		end
		else if(Br_En_E_48 == 2'b11 && pc_48 == 5'h0c) //CONDITIONAL BRANCH "BGT R4,R0,LOOP"
		begin
			pc_48 <= 5;
			exit = 0;
		end
		else if(Br_En_E_48 == 2'b10 && pc_48 == 5'h0e) //UNCONDITIONAL BRANCH "BR STOP"
		begin
			pc_48 <= pc_48 - 1;
			exit = 0;
		end
		else if(opcode_D_48 == 6'h011 && pc_48 == 5'h06 && exit < 2)
		begin
			pc_48 <= pc_48;
			exit <= exit + 1;
		end
		else
		begin
			pc_48 <= pc_48 + 1;			//NO BRANCHING
			exit = 0;
		end
	end
//--------------------------------------------------------DECODE STAGE----------------------------------------------------------------//
always@(Inst_D_48)
	begin
		opcode_D_48 = Inst_D_48[5:0];
		Rd_D_48 = Inst_D_48[31:27];
		Rs_D_48 = Inst_D_48[26:22];
		Alu_ip1_D_48 = regmem_48[Rs_D_48];		//VALUE OF SOURCE REGISTER
		Des_addr_D_48 = regmem_48[Rd_D_48];		//VALUE OF DESTINATION REGISTER

		if(pc_48 > 1)
			begin
				if(Rs_D_48 == Rd_E_48 || Rt_D_48 == Rd_E_48)
					fwd_req_D_48 = 1;
				else
					fwd_req_D_48 = 0;
			end

		case(opcode_D_48)

		//I - TYPE INSTRUCTION FILTERING
		6'h04:					//ADDI
		begin
			imm_D_48 = Inst_D_48[21:6];
			Alu_opx_D_48 = 3'b000;		//ADD
			Reg_wr_D_48 = 1'b1;		//ANSWER IS NEEDED TO BE STORED IN A REGISTER "Rd_48"
			Mem_wr_D_48 = 1'b0;
			Br_D_48 = 2'b00;		//NOT A BRANCHING INSTRUCTION
			Alu_src2_D_48 = 2'b01;		//ALU SOURCE 2 IS AN IMMEDIATE VALUE

			//SIGN EXTENSION OPERATION
			if(imm_D_48[15] == 1'b1)
				Imm_SignExt_D_48 = {16'hffff,imm_D_48};
			else
				Imm_SignExt_D_48 = {16'h0000,imm_D_48};
```

```
        end

        6'h05:                          //MOVIA
        begin
                imm_D_48 = Inst_D_48[21:6];
                Alu_opx_D_48 = 3'b100;          //MOV
                Reg_wr_D_48 = 1'b1;             //ANSWER NEEDS TO BE WRITTEN IN A REGISTER "Rd_48"
                Mem_wr_D_48 = 1'b0;
                Br_D_48 = 2'b00;                //NOT A BRANCHING INSTRUCTION
                Alu_src2_D_48 = 2'b01;          //ALU SOURCE 2 IS AN IMMEDIATE VALUE

                //SIGN EXTENSION OPERATION
                if(imm_D_48[15] == 1'b1)
                        Imm_SignExt_D_48 = {16'hffff,imm_D_48};
                else
                        Imm_SignExt_D_48 = {16'h0000,imm_D_48};
                end

        6'h07:                          //SUBI
                begin
                        imm_D_48 = Inst_D_48[21:6];
                        Alu_opx_D_48 = 3'b010;//SUBTRACT
                        Reg_wr_D_48 = 1'b1;     //ANSWER NEEDS TO BE WRITTEN IN A REGISTER "Rd_48"
                        Mem_wr_D_48 = 1'b0;
                        Br_D_48 = 2'b00;        //NOT A BRANCHING INSTRUCTION
                        Alu_src2_D_48 = 2'b01; //ALU SOURCE 2 IS AN IMMEDIATE DATA

                        //SIGN EXTENSION OPERATION
                        if(imm_D_48[15] == 1'b1)
                                Imm_SignExt_D_48 = {16'hffff,imm_D_48};
                        else
                                Imm_SignExt_D_48 = {16'h0000,imm_D_48};
                end

        6'h0F:                          //STW
                begin
                        imm_D_48 = Inst_D_48[21:6];
                        Alu_opx_D_48 = 3'b000;//ADD
                        Reg_wr_D_48 = 1'b0;
                        Mem_wr_D_48 = 1'b1;  //ANSWER NEEDS TO BE WRITTEN IN A MEMORY LOCATION
                        Br_D_48 = 2'b00;        //NOT A BRANCHING INSTRUCTION
                        Alu_src2_D_48 = 2'b01; //ALU SOURCE 2 IS AN IMMEDIATE DATA

                        //SIGN EXTENSION OPERATION
                        if(imm_D_48[15] == 1'b1)
                                Imm_SignExt_D_48 = {16'hffff,imm_D_48};
                        else
                                Imm_SignExt_D_48 = {16'h0000,imm_D_48};
                end

        6'h11:                          //LDW
                begin
                        imm_D_48 = Inst_D_48[21:6];
                        Alu_opx_D_48 = 3'b000;//ADD
                        Reg_wr_D_48 = 1'b1;
                        Mem_wr_D_48 = 1'b0;
```

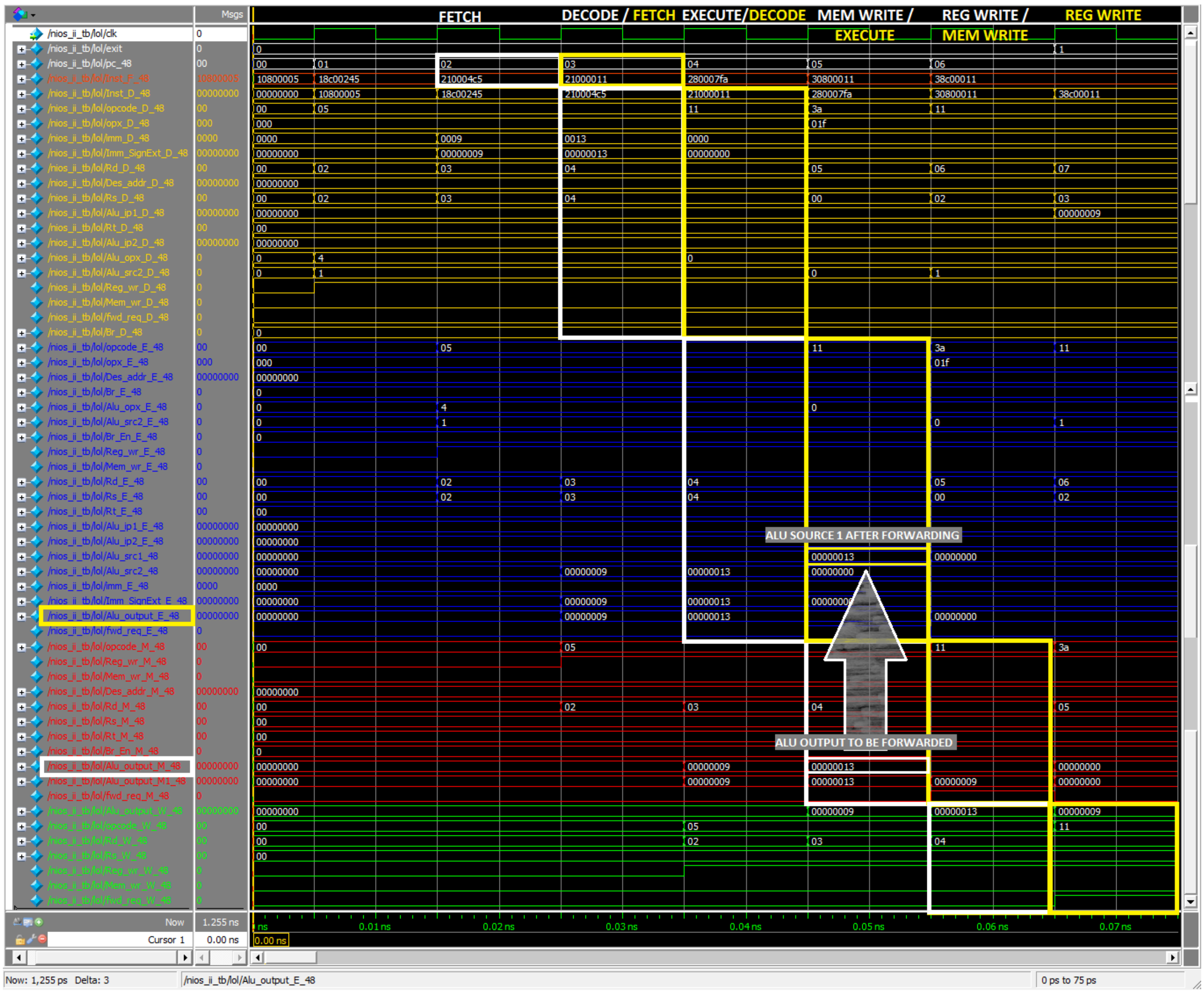
```
                    Br_D_48 = 2'b00;           //NOT A BRANCHING INSTRUCTION
                    Alu_src2_D_48 = 2'b01; //ALU SOURCE 2 IS AN IMMEDIATE DATA

                    //SIGN EXTENSION OPERATION
                    if(imm_D_48[15] == 1'b1)
                        Imm_SignExt_D_48 = {16'hffff,imm_D_48};

                    else
                        Imm_SignExt_D_48 = {16'h0000,imm_D_48};
                end

        //R - TYPE INSTRUCTION FILTERING
            6'h3A:                                              //ALL R-TYPE INSTRUCTIONS
            begin
                opx_D_48 = Inst_D_48[16:6];
                Rt_D_48 = Inst_D_48[21:17];
                Alu_ip2_D_48 = regmem_48[Rt_D_48];
                Br_D_48 = 2'b00;                    //NOT A BRANCHING INSTRUCTION
                Alu_src2_D_48 = 2'b00;              //ALU SOURCE IS A REGISTER "Rt_48"
                Reg_wr_D_48 = 1'b1;                 //ANSWER NEEDS TO BE WRITTEN IN A REGISTER
                Mem_wr_D_48 = 1'b0;
                if(opx_D_48 == 11'h01b)             //MUL
                    begin
                        Alu_opx_D_48 = 3'b011;          //MULTIPLY
                        //fwd_req_D_48 = 1'b1;
                    end
                else if(opx_D_48 == 11'h01f && pc_48 == 6'h09)//ADD
                    begin
                        Alu_opx_D_48 = 3'b000;          //ADDITION
                        fwd_req_D_48 = 1'b1;
                    end
                else                                    //ADD
                    begin
                        Alu_opx_D_48 = 3'b000;          //ADDITION
                    end
            end

        //J - TYPE INSTRUCTION FILTERING
            6'h10:                                      //BGT
                begin
                    imm_D_48 = Inst_D_48[31:6];
                    Imm_SignExt_D_48 = {16'h0000,imm_D_48};
                    Alu_opx_D_48 = 3'b001;              //COMPARE
                    Reg_wr_D_48 = 1'b0;
                    Mem_wr_D_48 = 1'b0;
                    Br_D_48 = 2'b11;                    //ENABLE BRANCH SELECT BIT
                    Alu_src2_D_48 = 2'b00;              //ALU SOURCE 2 IS A REGISTER
                end
            6'h06:                                      //BR
                begin
                    Alu_opx_D_48 = 3'b001;              //COMPARE
                    Imm_SignExt_D_48 = 32'h00000000;
                    Reg_wr_D_48 = 1'b0;
                    Mem_wr_D_48 = 1'b0;
                    Br_D_48 = 2'b10;                    //ENABLE BRANCH SELECT BIT
                    Alu_src2_D_48 = 2'b01;              //ALU SOURCE 2 IS A IMMEDIATE DATA
                end
            endcase
        end
always@(posedge clk)
        begin
        //DECODE STAGE
            fwd_req_E_48 <= fwd_req_D_48;
            Des_addr_E_48 <= Des_addr_D_48;
            opcode_E_48 <= opcode_D_48;
            opx_E_48 <= opx_D_48;
            Rd_E_48 <= Rd_D_48;
            Rs_E_48 <= Rs_D_48;
            Rt_E_48 <= Rt_D_48;
            Alu_ip1_E_48 <= Alu_ip1_D_48;              //SOURCE REGISTER VALUE
            Alu_ip2_E_48 <= Alu_ip2_D_48;              //TARGET REGISTER VALUE
            Imm_SignExt_E_48 <= Imm_SignExt_D_48;
            Alu_opx_E_48 <= Alu_opx_D_48;
            Reg_wr_E_48 <= Reg_wr_D_48;
            Mem_wr_E_48 <= Mem_wr_D_48;
            Br_E_48 <= Br_D_48;                        //BRANCH SELECT BIT
            Alu_src2_E_48 <= Alu_src2_D_48; //ALU source 2 selection bits (Rt or Immediate data)
        end
//---------------------------------------------------------EXECUTE STAGE------------------------------------------------------//
always@(Rs_E_48,Rd_E_48,Rt_E_48,Alu_ip1_E_48,Alu_ip2_E_48,Imm_SignExt_E_48,Reg_wr_E_48,
        Mem_wr_E_48,Br_E_48,Alu_src2_E_48,Alu_opx_E_48,fwd_req_E_48,Des_addr_E_48,opcode_E_48,
        opx_E_48)
        begin
            if(fwd_req_E_48 == 0)
            begin
                Alu_src1_48 = Alu_ip1_E_48;            //ALU SOURCE 1 IS ALWAYS THE SOURCE REGISTER
                if(Alu_src2_E_48 == 2'b00)             //ALU SOURCE 2 IS Rt_48 FOR R-TYPE INSTRUCTION
                    Alu_src2_48 = Alu_ip2_E_48;
                else if (Alu_src2_E_48 == 2'b01) //ALU SOURCE 2 IS IMM DATA FOR I-TYPE INSTRUCTION
                Alu_src2_48 = Imm_SignExt_E_48;
            end
        else                                           //FORWARDING
        begin
            if(opx_E_48 == 11'h01b)                    //MULTIPLY
            begin
                Alu_src1_48 = Alu_ip1_E_48;
                Alu_src2_48 = Alu_ip2_E_48;
            end
            else if(opx_E_48 == 11'h01f)               //ADD
            begin
                Alu_src2_48 = Alu_output_M1_48;
                Alu_src1_48 = Alu_ip1_E_48;     //ALU SOURCE 2 IS Rt_48 FOR R-TYPE INSTRUCTION
            end
            else
            begin
                Alu_src1_48 = Alu_output_M1_48;
                if(Alu_src2_E_48 == 2'b00)       //ALU SOURCE 2 IS Rt_48 FOR R-TYPE INSTRUCTION
                    Alu_src2_48 = Alu_ip2_E_48;
                else if (Alu_src2_E_48 == 2'b01) //ALU SOURCE 2 IS IMM DATA FOR I-TYPE INSTRUCTION
                    Alu_src2_48 = Imm_SignExt_E_48;
                end
            end
        case(Alu_opx_E_48)
                3'b000:                                         //ADD
                begin
                        //CALCULATE THE SUM OR CALCULATE THE MEMORY ADDRESS
                        Alu_output_E_48 = Alu_src1_48 + Alu_src2_48;
                        Br_En_E_48=2'b00;
                        //DISABLE BRANCH
                end

                3'b001:                                         //COMPARE
                begin
                        Alu_src1_48 = Des_addr_E_48;
                        Alu_src2_48 = Alu_ip1_E_48;
                        Alu_output_E_48 = Alu_ip1_E_48;
                        if(Alu_src1_48 > Alu_src2_48)           //BRANCH ENABLE IF R4 GREATER THAN R0
                                Br_En_E_48 = 2'b11;
                        else
                                Br_En_E_48 = 2'b00;             //DISABLE BRANCH
                end

                3'b010:                                         //SUBTRACT
                begin
                        Alu_output_E_48 = Alu_src1_48-Alu_src2_48;
                        Br_En_E_48 = 2'b00;                     //DISABLE BRANCH
                end

                3'b011:                                         //MULTIPLY
                begin
                        Alu_output_E_48 = Alu_src1_48*Alu_src2_48;
                        Br_En_E_48 = 2'b00;                     //DISABLE BRANCH
                end

                3'b100:                                         //MOVE
                begin
                        Alu_output_E_48 = Alu_src2_48;
                        Br_En_E_48 = 2'b00;                     //DISABLE BRANCH
                end
        endcase
end

always@(posedge clk)
        begin
                Des_addr_M_48<=Des_addr_E_48;
                Rd_M_48<=Rd_E_48;
                opcode_M_48<=opcode_E_48;
                Alu_output_M_48<=Alu_output_E_48;
                Reg_wr_M_48<=Reg_wr_E_48;
                Mem_wr_M_48<=Mem_wr_E_48;
                fwd_req_M_48 <= fwd_req_E_48;
        end
```

```
//----------------------------------------------MEMORY READ STAGE--------------------------------------------------------------//
always@(Alu_output_M_48,Reg_wr_M_48,Mem_wr_M_48,Rd_M_48,opcode_M_48)
        begin
                if(Mem_wr_M_48==1'b1)
                begin
                        if(Reg_wr_M_48==1'b0&&opcode_M_48==6'h0F) //STORE DATA IN MEMORY
                                datamem_48[Alu_output_M_48] = Des_addr_M_48;
                end
                else if(Mem_wr_M_48==1'b0 && opcode_M_48==6'h11) //LOAD DATA IN REGISTER
                        Alu_output_M1_48 = datamem_48[Alu_output_M_48];
                else
                        Alu_output_M1_48 = Alu_output_M_48;
        end

always@(posedge clk)
        begin
        //MEMORY READ STAGE
                Rd_W_48 <= Rd_M_48;
                Alu_output_W_48 <= Alu_output_M1_48;
                Reg_wr_W_48 <= Reg_wr_M_48;
                Mem_wr_W_48 <= Mem_wr_M_48;
                opcode_W_48 <= opcode_M_48;
                fwd_req_W_48 <= fwd_req_M_48;
        end

//----------------------------------------------------WRITE BACK STAGE--------------------------------------------------------//
always@(Alu_output_W_48,Rd_W_48,Mem_wr_W_48,Reg_wr_W_48)
        begin
                if(Reg_wr_W_48 == 1'b1)
                        regmem_48[Rd_W_48] = Alu_output_W_48;
                end
endmodule
```

NISHITH SHAH (SJSU ID:011478948) DATE:1ST DEC'16

OUTPUT:

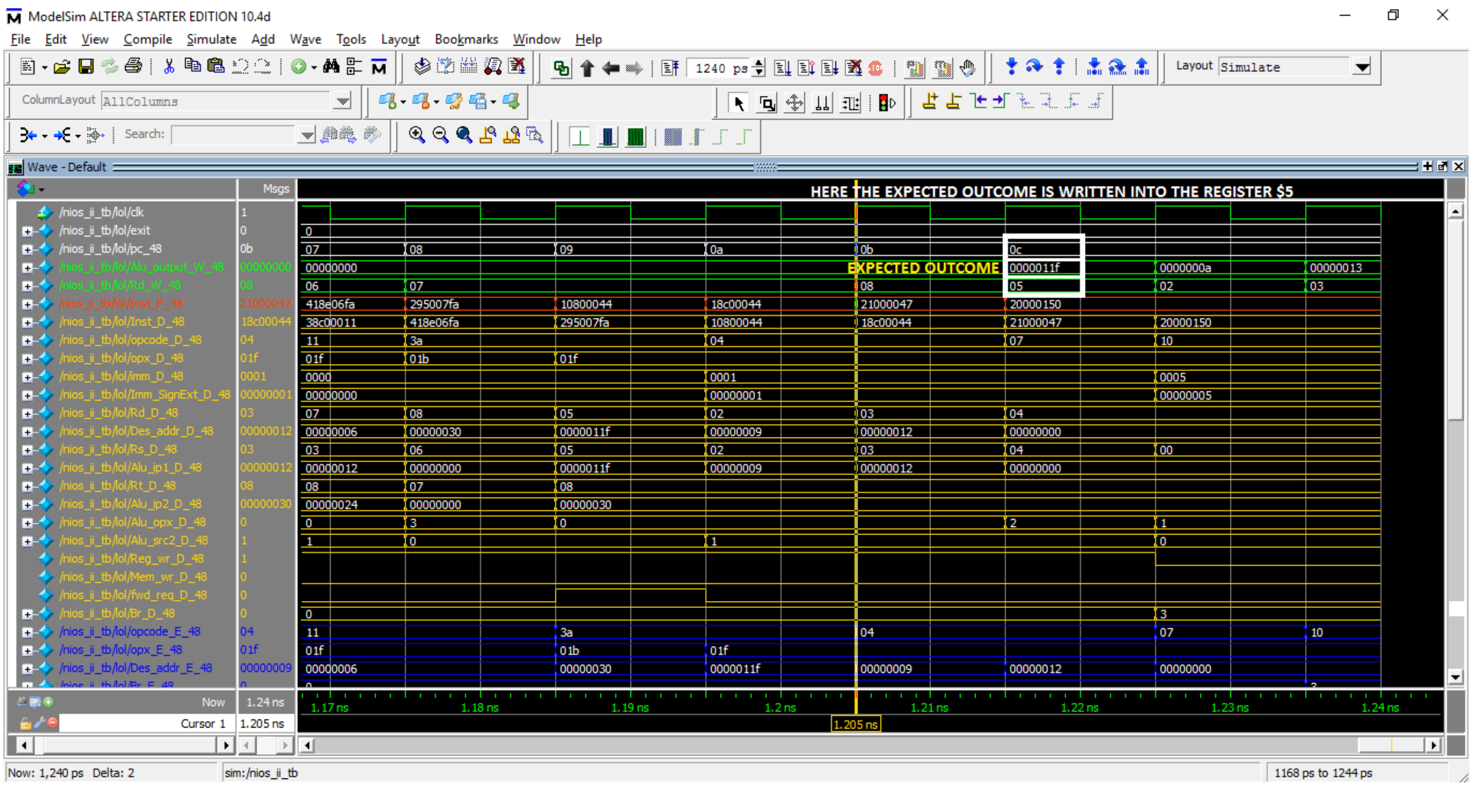
DATA FORWARDING BETWEEN MOVIA R4,N & LDW R4,0(R4) :



Here, ALU output of the 3rd instruction is the location from which data is to be read by the next instruction. Instead of waiting for the values to be written in Rd, the ALU output of 3rd instruction was forwarded to the ALU input of the 4th instruction.

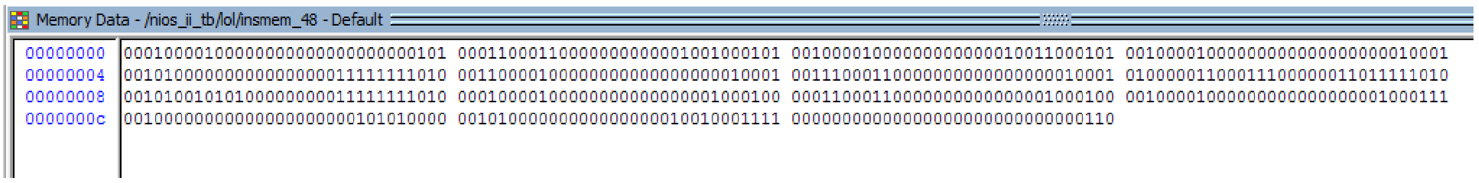
NISHITH SHAH (SJSU ID:011478948) **DATE:1ST DEC'16**

FINAL OUTPUT:


ModelSim ALTERA STARTER EDITION 10.4d
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout AllColumns
Search:
Wave - Default
HERE THE EXPECTED OUTCOME IS WRITTEN INTO THE REGISTER $5
EXPECTED OUTCOME

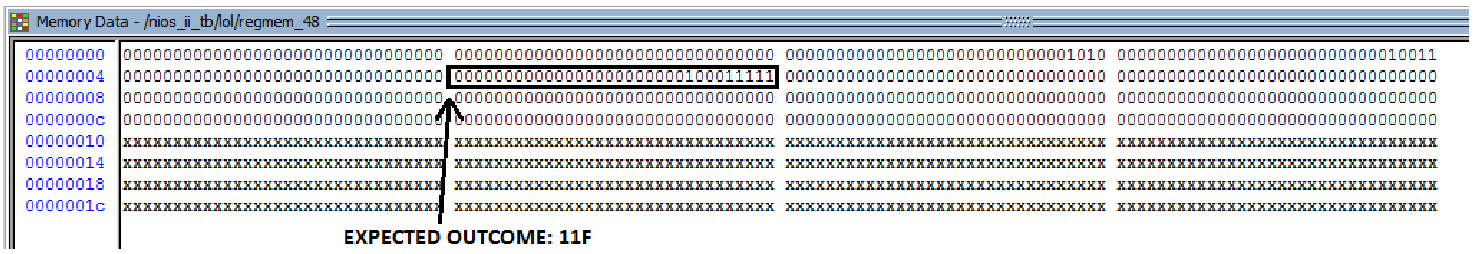
```
/nios_ii_tb/lol/clk              1
/nios_ii_tb/lol/exit             0
/nios_ii_tb/lol/pc_48            0b
/nios_ii_tb/lol/Alu_output_W_48  00000000
/nios_ii_tb/lol/Rd_W_48          08
/nios_ii_tb/lol/Inst_F_48        21000047
/nios_ii_tb/lol/Inst_D_48        18c00044
/nios_ii_tb/lol/opcode_D_48      04
/nios_ii_tb/lol/opx_D_48         01f
/nios_ii_tb/lol/imm_D_48         0001
/nios_ii_tb/lol/Imm_SignExt_D_48 00000001
/nios_ii_tb/lol/Rd_D_48          03
/nios_ii_tb/lol/Des_addr_D_48    00000012
/nios_ii_tb/lol/Rs_D_48          03
/nios_ii_tb/lol/Alu_ip1_D_48     00000012
/nios_ii_tb/lol/Rt_D_48          08
/nios_ii_tb/lol/Alu_ip2_D_48     00000030
/nios_ii_tb/lol/Alu_opx_D_48     0
/nios_ii_tb/lol/Alu_src2_D_48    1
/nios_ii_tb/lol/Reg_wr_D_48      1
/nios_ii_tb/lol/Mem_wr_D_48      0
/nios_ii_tb/lol/fwd_req_D_48     0
/nios_ii_tb/lol/Br_D_48          0
/nios_ii_tb/lol/opcode_E_48      04
/nios_ii_tb/lol/opx_E_48         01f
/nios_ii_tb/lol/Des_addr_E_48    00000009

Now       1.24 ns
Cursor 1  1.205 ns

Now: 1,240 ps  Delta: 2    sim:/nios_ii_tb    1168 ps to 1244 ps
```

INSTRUCTION MEMORY:



```
Memory Data - /nios_ii_tb/lol/insmem_48 - Default
00000000 00010000100000000000000000000101 00011000110000000000001001000101 00100001000000000000010011000101 00100001000000000000000000010001
00000004 00101000000000000000001111111010 00110000100000000000000000010001 00111000110000000000000000010001 01000001100011100000011011111010
00000008 00101001010100000000001111111010 00010000100000000000000001000100 00011000110000000000000001000100 00100001000000000000000001000111
0000000c 00100000000000000000000101010000 00101000000000000000010010001111 00000000000000000000000000000110
```

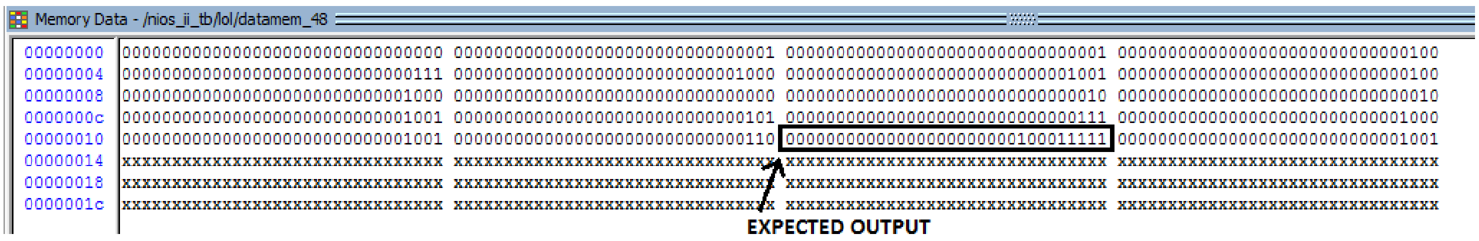
REGISTER MEMORY:



```
Memory Data - /nios_ii_tb/lol/regmem_48
00000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000001010 00000000000000000000000000010011
00000004 00000000000000000000000000000000 00000000000000000000000100011111 00000000000000000000000000000000 00000000000000000000000000000000
00000008 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
0000000c 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
00000010 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000014 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000018 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

EXPECTED OUTCOME: 11F

DATA MEMORY:



```
Memory Data - /nios_ii_tb/lol/datamem_48
00000000 00000000000000000000000000000000 00000000000000000000000000000001 00000000000000000000000000000001 00000000000000000000000000000100
00000004 00000000000000000000000000000111 00000000000000000000000000001000 00000000000000000000000000001001 00000000000000000000000000000100
00000008 00000000000000000000000000001000 00000000000000000000000000000000 00000000000000000000000000000010 00000000000000000000000000000010
0000000c 00000000000000000000000000001001 00000000000000000000000000000101 00000000000000000000000000000111 00000000000000000000000000001000
00000010 00000000000000000000000000001001 00000000000000000000000000000110 00000000000000000000000100011111 00000000000000000000000000001001
00000014 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000018 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

EXPECTED OUTPUT

RESULT:

Final answer ($11F_{16}$ or 287_{10}) is stored at datamem_48[32'H12 or 18] in data memory.

<u>CONCLUSION:</u>

Without forwarding of data the same program took 250 clock cycle with the clock duration of 10 ps/clock.
After using data forwarding technique this program took only 122 clock cycle with the clock duration of 10 ps/clock to complete the same task.