Mini Project - 2

Subject: EE 275
(Advance Computer Architecture)

Name: Sagar M. Shah

SJSU ID: 011426818

Department: Electrical engineering

Professor: Dr. Chang Choo

University: San Jose State University

MY APPROACH:

AVECTOR = 16‘h0000
N = 9
DOT_PRODUCT = 23

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	-	0000000000001010	000101
MOVIA R4, **N**	00100	00100	-	0000000000001001	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

Instruction Type:

- **R-type instruction:**

Rd	Rs	Rt	Immediate value	Opcode

- **I type instruction:**

Rd	Rs	Immediate value	Opcode

- **J-type instruction:**

Immediate value	Opcode

Where:

Rd = Destination register
Rs = Source register
Rt = Target register
Immediate = immediate value
Opcode = digital significance of instruction

Name of variables and its meaning:

- **Register in fetch stage:**

instF_18 = Instruction.

- **Registers in Decode stage:**

alu_en_D_18 = decides if ALU operation is to be done or not.
reg_rw_D_18 = write operations are there over register or not.
mem_rw_D_18 = write operations are there over memory.
alu_fn_D_18 = Decides which operation ALU should do.
alu_ip_sel_18 = selects 2nd input of ALU.
brn_en_D_18 = Decides to branch or not.
op_D_18 = opcode.
Rd_D_18 = destination register.
Rs_D_18 = source register.
Rt_D_18 = target register.
opx_D_18 = opcode extension.
imm_D_18 = immediate value.
Badd_D_18 = branching address.
instD_18 = instruction.
alu_ip1_D_18 = 1st input of ALU.
alu_ip2_0_D_18 = 1st option of 2nd input (immediate).
alu_ip2_1_D_18 = 2nd option of 2nd input (data of target register).
alu_ip2_D_18 = final decided 2nd input.
imm_SE_D_18 = sign extended immediate value.
Badd_SE_D_18 = sign extended branching address.
Change_D = Signal for data forwarding need.

- **Register in Execution stage:**

alu_en_E_18 = decides if ALU operation is to be done or not.
reg_rw_E_18 = write operations are there over register or not.
mem_rw_E_18 = write operations are there over memory.
alu_fn_E_18 = Decides which operation ALU should do.
brn_en_E_18 = Decides to branch or not.
op_E_18 = opcode.
Rd_E_18 = destination register.
alu_out_E_18 = output of ALU operation.
alu_ip1_E_18 = 1st input of ALU.
alu_ip2_E_18 = final decided 2nd input.
imm_SE_E_18 = sign extended immediate value.
Badd_SE_E_18 = sign extended branching address.
Change_E = Signal for data forwarding need.
alu_ip1F_E_18 = 1st input of ALU after data forwarding.
alu_ip2F_E_18 = 2nd input of ALU after data forwarding.

- **Register in Memory stage:**

reg_rw_M_18 = write operations are there over register or not.
mem_rw_M_18 = write operations are there over memory.
Rd_M_18 = destination register.
alu_out_M_18 = output of ALU operation.
op_M_18 = opcode.

- **Register in Write back stage:**

reg_rw_WB_18 = write operations are there over register or not.
mem_rw_WB_18 = write operations are there over memory.
Rd_WB_18 = destination register.
alu_out_WB_18 = output of ALU operation.
op_WB_18 = opcode.
pc_18 = Program counter.

- **Register used for data forwarding:**

Pre_Rd_18 = Destination reg. of previous instruction.
Pre_Rs_18 = Source reg. of previous instruction.
Pre_Rt_18 = target reg. of previous instruction.

- **Memory used:**

imem = Instruction memory
Word size = 32 bit.
Depth = 15

rmem = Register memory
Word size = 32 bit.
Depth = 32

dmem = data memory
Word size = 32 bit.
Depth = 32

• Program to be run over NIOS II architecture:

```
.include "nios_macros.s"
.global _start
_start:
        movia r2, AVECTOR             /* Register r2 is a pointer to vector A */
        movia r3, BVECTOR             /* Register r3 is a pointer to vector B */
        movia r4, N
        ldw r4, 0(r4)                 /* Register r4 is used as the counter for loop iterations */
        add r5, r0, r0                /* Register r5 is used to accumulate the product */
LOOP:   ldw r6, 0(r2)                 /* Load the next element of vector A */
        ldw r7, 0(r3)                 /* Load the next element of vector B */
        mul r8, r6, r7                /* Compute the product of next pair of elements */
        add r5, r5, r8                /* Add to the sum */
        addi r2, r2, 1                /* Increment the pointer to vector A */
        addi r3, r3, 1                /* Increment the pointer to vector B */
        subi r4, r4, 1                /* Decrement the counter */
        bgt r4, r0, LOOP              /* Loop again if not ?nished */
        stw r5, DOT_PRODUCT(r0)       /* Store the result in memory */
STOP: br STOP

N:
.word 9                       /* Specify the number of elements */
AVECTOR:
.word 0, 1, 1, 4, 2, 6, 8, 1, 8       /* Specify the elements of vector A SJSU ID*/
BVECTOR:
.word 2, 3, 3, 6, 4, 8, 0, 3, 0 /* Specify the elements of vector B Plus 2 to each Digit of SJSU ID*/
```

• Expected answer:

(0x2) +(1x3) +(1x3) +(4x6) +(2x4) +(6x8) +(8x0) +(1x3) +(8x0) = 89

• Verilog Implementation of NIOS II architecture:

```
module NIOS_II( alu_out_WB_18, Rd_WB_18, clk_18);

	output alu_out_WB_18, Rd_WB_18;
	input clk_18;

	wire clk_18;
	reg[31:0] imem[0:14],rmem[0:31],dmem[0:31];
	reg[4:0] Pre_Rd_18,Pre_Rs_18,Pre_Rt_18;
	reg[1:0] counter=0;

	// register for fetch stage
	reg[31:0] instF_18;

	// registers for Decode stage
	reg alu_en_D_18, reg_rw_D_18,mem_rw_D_18,change_D;
	reg[1:0] alu_fn_D_18,alu_ip_sel_18,brn_en_D_18;
	reg[5:0] op_D_18,pc_18;
	reg[4:0] Rd_D_18,Rs_D_18,Rt_D_18;
	reg[10:0] opx_D_18;
	reg[15:0] imm_D_18;
	reg[25:0] Badd_D_18;
	reg[31:0]
instD_18,alu_ip1_D_18,alu_ip2_0_D_18,alu_ip2_1_D_18,alu_ip2_D_18,imm_SE_D_18,Badd_SE_D_
18;

	// registers for Execute stage
	reg alu_en_E_18,reg_rw_E_18,mem_rw_E_18,change_E;
	reg[1:0] alu_fn_E_18,brn_en_E_18;
	reg[5:0] op_E_18;
	reg[4:0] Rd_E_18;
	reg[31:0] alu_out_E_18,alu_ip1_E_18,alu_ip2_E_18,imm_SE_E_18,Badd_SE_E_18;
	reg[31:0] alu_ip1F_E_18,alu_ip2F_E_18;

	// registers for memory stage
	reg reg_rw_M_18,mem_rw_M_18;
	reg[4:0] Rd_M_18;
	reg[5:0] op_M_18;
	reg[31:0] alu_out_M_18;

	// registers for write back stage
	reg reg_rw_WB_18,mem_rw_WB_18;
	reg[4:0] Rd_WB_18;
	reg[5:0] op_WB_18;
	reg[31:0] alu_out_WB_18;
	initial
	begin
		instF_18=0;Pre_Rd_18=0;Pre_Rs_18=0;Pre_Rt_18=0;
		alu_en_D_18=0;reg_rw_D_18=0;mem_rw_D_18=0;alu_fn_D_18=0;alu_ip_sel_18=0;
		brn_en_D_18=0;op_D_18=0;Rd_D_18=0;Rs_D_18=0;
```

```
                    Rt_D_18=0;opx_D_18=0;imm_D_18=0;Badd_D_18=0;instD_18=0;alu_ip1_D_18=0;
                    alu_ip2_0_D_18=0;alu_ip2_1_D_18=0;alu_ip2_D_18=0;
                    imm_SE_D_18=0;Badd_SE_D_18=0;change_D=0;

          alu_en_E_18=0;reg_rw_E_18=0;mem_rw_E_18=0;alu_fn_E_18=0;brn_en_E_18=0;op_E_18=
0;
                    Rd_E_18=0;reg_rw_E_18=0;mem_rw_E_18=0;
alu_out_E_18=0;alu_ip1_E_18=0;alu_ip2_E_18=0;imm_SE_E_18=0;Badd_SE_E_18=0;
                    change_E=0;alu_ip2F_E_18=0;alu_ip1F_E_18=0;

          reg_rw_M_18=0;mem_rw_M_18=0;reg_rw_M_18=0;mem_rw_M_18=0;Rd_M_18=0;
                    alu_out_M_18=0;op_M_18=0;

          reg_rw_WB_18=0;mem_rw_WB_18=0;Rd_WB_18=0;alu_out_WB_18=0;op_WB_18=0;
                    pc_18=0;

                    // instruction memory //
                    imem[0]=32'h10000004;
                    imem[1]=32'h18000284;
                    imem[2]=32'h200005c4;
                    imem[3]=32'h21000017;
                    imem[4]=32'h28000c7a;
                    imem[5]=32'h30800017;
                    imem[6]=32'h38c00017;
                    imem[7]=32'h418e09fa;
                    imem[8]=32'h29500c7a;
                    imem[9]=32'h10800044;
                    imem[10]=32'h18c00044;
                    imem[11]=32'h21000045;
                    imem[12]=32'h203ffdd6;
                    imem[13]=32'h28000855;
                    imem[14]=32'hffffffc6;

                    // Register memory //
                    rmem[0]=32'h00000000;
                    rmem[1]=32'h00000000;
                    rmem[2]=32'h00000000;
                    rmem[3]=32'h00000000;
                    rmem[4]=32'h00000000;
                    rmem[5]=32'h00000000;
                    rmem[6]=32'h00000000;
                    rmem[7]=32'h00000000;
                    rmem[8]=32'h00000000;
                    rmem[9]=32'h00000000;
                    rmem[10]=32'h00000000;
                    rmem[11]=32'h00000000;
                    rmem[12]=32'h00000000;
                    rmem[13]=32'h00000000;
                    rmem[14]=32'h00000000;
                    rmem[15]=32'h00000000;
                    rmem[16]=32'h00000000;
```

```
rmem[17]=32'h00000000;
rmem[18]=32'h00000000;
rmem[19]=32'h00000000;
rmem[20]=32'h00000000;
rmem[21]=32'h00000000;
rmem[22]=32'h00000000;
rmem[23]=32'h00000000;
rmem[24]=32'h00000000;
rmem[25]=32'h00000000;
rmem[26]=32'h00000000;
rmem[27]=32'h00000000;
rmem[28]=32'h00000000;
rmem[29]=32'h00000000;
rmem[30]=32'h00000000;
rmem[31]=32'h00000000;

// data memory //
dmem[0]=32'h00000000; // 0 A vector starts here
dmem[1]=32'h00000001; // 1
dmem[2]=32'h00000001; // 1
dmem[3]=32'h00000004; // 4
dmem[4]=32'h00000002; // 2
dmem[5]=32'h00000006; // 6
dmem[6]=32'h00000008; // 8
dmem[7]=32'h00000001; // 1
dmem[8]=32'h00000008; // 8
dmem[9]=32'h00000000;
dmem[10]=32'h00000002; // 2 B vector starts here
dmem[11]=32'h00000003; // 3
dmem[12]=32'h00000003; // 3
dmem[13]=32'h00000006; // 6
dmem[14]=32'h00000004; // 4
dmem[15]=32'h00000008; // 8
dmem[16]=32'h00000000; // 0
dmem[17]=32'h00000003; // 3
dmem[18]=32'h00000000; // 0
dmem[19]=32'h00000000;
dmem[20]=32'h00000000;
dmem[21]=32'h00000000; // final answer (Dot product)
dmem[22]=32'h00000000;
dmem[23]=32'h00000009; // location of N (no. of loops)
dmem[24]=32'h00000000;
dmem[25]=32'h00000000;
dmem[26]=32'h00000000;
dmem[27]=32'h00000000;
dmem[28]=32'h00000000;
dmem[29]=32'h00000000;
dmem[30]=32'h00000000;
dmem[31]=32'h00000000;
end
```

```
always @(posedge clk_18)
begin
if(brn_en_E_18==2'b11 && pc_18 == 6'h0c)
begin
        pc_18 <= (pc_18) - imm_SE_E_18[3:0];
        counter <= 0;
end
else if(brn_en_E_18!=2'b11 && pc_18 == 6'h0c && counter < 2)
begin
        pc_18 <= pc_18;
        counter <= counter +1;
end
else if(brn_en_E_18!=2'b11 && pc_18 == 6'h0c && counter >= 2)
begin
        pc_18 <= pc_18+1;
end
else if(brn_en_E_18==2'b00 && pc_18 == 6'h0e)
        pc_18 <= pc_18;
else
        pc_18 <= pc_18+1;

        // 1st stage to 2nd stage //
        instD_18 <= instF_18;

        // 2nd stage to 3rd stage //
        alu_en_E_18 <= alu_en_D_18;
        reg_rw_E_18 <= reg_rw_D_18;
        mem_rw_E_18 <= mem_rw_D_18;
        alu_fn_E_18 <= alu_fn_D_18;
        Rd_E_18 <= Rd_D_18;
        alu_ip1_E_18 <= alu_ip1_D_18;
        alu_ip2_E_18 <= alu_ip2_D_18;
        imm_SE_E_18 <= imm_SE_D_18;
        Badd_SE_E_18 <= Badd_SE_D_18;
        op_E_18 <= op_D_18;
        change_E <= change_D;

        // 3rd stage to 4th stage //
        reg_rw_M_18 <= reg_rw_E_18;
        mem_rw_M_18 <= mem_rw_E_18;
        Rd_M_18 <= Rd_E_18;
        alu_out_M_18 <= alu_out_E_18;
        op_M_18 <= op_E_18;

        // 4th stage to 5th stage //
        reg_rw_WB_18 <= reg_rw_M_18;
        mem_rw_WB_18 <= mem_rw_M_18;
        Rd_WB_18 <= Rd_M_18;
        alu_out_WB_18<= alu_out_M_18;
        op_WB_18 <= op_M_18;
end
```

```
		always @(pc_18)//---------------------------------------------------------------------------------------
------------// stage 1
		begin
			instF_18 = imem[pc_18];
		end

		always @(instD_18)//------------------------------------------------------------------------------------
------------// stage 2
		begin
			Pre_Rd_18 = Rd_D_18;
			Pre_Rs_18 = Rs_D_18;
			Pre_Rt_18 = Rt_D_18;
			op_D_18 = instD_18[5:0];
			Rd_D_18 = instD_18[31:27];
			Rs_D_18 = instD_18[26:22];
			case(op_D_18)

			//------------- I type inst. -------------//
			6'h04: //MOVIA,ADDI
		begin
			imm_D_18 = instD_18[21:6];
			alu_fn_D_18 = 2'b00; //00 for +
			alu_en_D_18 = 1'b1; //alu enabled
			reg_rw_D_18 = 1'b1; //register write back enable
			mem_rw_D_18 = 1'b0; //memory write disable
			alu_ip_sel_18 = 2'b00; //0 for imm as input
		if(Pre_Rd_18 == Rs_D_18 && pc_18 != 1)
			change_D = 1'b1; //change the input for ALU
		else
			change_D = 1'b0;
		if(reg_rw_WB_18==1'b0)
		begin
			alu_ip1_D_18 = rmem[Rs_D_18];
			alu_ip2_1_D_18 = rmem[Rt_D_18];
		end
		else
		begin
		repeat(1)
		@(negedge clk_18);
			alu_ip1_D_18 = rmem[Rs_D_18];
			alu_ip2_1_D_18 = rmem[Rt_D_18];
		end
		end
			6'h05: //SUBI
		begin
			imm_D_18 = instD_18[21:6];
			alu_fn_D_18 = 2'b01; //01 for -
			alu_en_D_18 = 1'b1; //alu enabled
			reg_rw_D_18 = 1'b1; //register write back enable
			mem_rw_D_18 = 1'b0; //memory write disable
```

```
        alu_ip_sel_18 = 2'b00; //0 for imm as input
if(Pre_Rd_18 == Rs_D_18 && pc_18 != 1)
        change_D = 1'b1; //change the input for ALU
else
        change_D = 1'b0;
if(reg_rw_WB_18==1'b0)
begin
        alu_ip1_D_18 = rmem[Rs_D_18];
        alu_ip2_1_D_18 = rmem[Rt_D_18];
end
else
begin
repeat(1)
@(negedge clk_18);
        alu_ip1_D_18 = rmem[Rs_D_18];
        alu_ip2_1_D_18 = rmem[Rt_D_18];
end
end
        6'h15: //STW
begin
        imm_D_18 = instD_18[21:6];
        alu_fn_D_18 = 2'b00; //00 for +
        alu_en_D_18 = 1'b1; //alu enabled
        reg_rw_D_18 = 1'b0; //register write back disable
        mem_rw_D_18 = 1'b1; //memory write enable
        alu_ip_sel_18 = 2'b00; //0 for imm as input
if(Pre_Rd_18 == Rs_D_18 && pc_18 != 1)
        change_D = 1'b1; //change the input for ALU
else
        change_D = 1'b0;
if(reg_rw_WB_18==1'b0)
begin
        alu_ip1_D_18 = rmem[Rs_D_18];
        alu_ip2_1_D_18 = rmem[Rt_D_18];
end
else
begin
repeat(1)
@(negedge clk_18);
        alu_ip1_D_18 = rmem[Rs_D_18];
        alu_ip2_1_D_18 = rmem[Rt_D_18];
end
end
        6'h17: //LDW
begin
        imm_D_18 = instD_18[21:6];
        alu_fn_D_18 = 2'b00; //00 for +
        alu_en_D_18 = 1'b1; //alu enabled
        reg_rw_D_18 = 1'b1; //register write back enable
        mem_rw_D_18 = 1'b0; //memory write disable
        alu_ip_sel_18 = 2'b00; //0 for imm as input
```

```
if(Pre_Rd_18 == Rs_D_18 && pc_18 != 1)
        change_D = 1'b1; //change the input for ALU
else
        change_D = 1'b0;
if(reg_rw_WB_18==1'b0 && mem_rw_WB_18 == 1'b0)
begin
        alu_ip1_D_18 = dmem[rmem[Rs_D_18]];
        alu_ip2_1_D_18 = rmem[Rt_D_18];
end
else
begin
repeat(1)
@(negedge clk_18);
        alu_ip1_D_18 = dmem[rmem[Rs_D_18]];
        alu_ip2_1_D_18 = rmem[Rt_D_18];
end
end

        //------------- R type inst. -------------//
        6'h3a: //ADD,MUL
begin
        Rt_D_18 = instD_18[21:17];
        opx_D_18 = instD_18[16:6];
if(opx_D_18==11'h31)
        alu_fn_D_18 = 2'b00; //00 for +
else if(opx_D_18==11'h27)
        alu_fn_D_18 = 2'b10; //10 for *
        alu_en_D_18 = 1'b1; //alu enabled
        reg_rw_D_18 = 1'b1; //register write back enable
        mem_rw_D_18 = 1'b0; //memory write disable
        alu_ip_sel_18 = 2'b01; //1 for reg Rt as input
if(Pre_Rd_18 == Rs_D_18 || Pre_Rd_18 == Rt_D_18 && pc_18 != 1)
        change_D = 1'b1; //change the input for ALU
else
        change_D = 1'b0;
if(reg_rw_WB_18==1'b0)
begin
        alu_ip1_D_18 = rmem[Rs_D_18];
        alu_ip2_1_D_18 = rmem[Rt_D_18];
end
else
begin
repeat(1)
@(negedge clk_18);
        alu_ip1_D_18 = rmem[Rs_D_18];
        alu_ip2_1_D_18 = rmem[Rt_D_18];
end
end

        //------------- J type inst. -------------//
        6'h06: //BR
```

```
begin
        imm_D_18 = instD_18[21:6];
        Badd_D_18 = {Rd_D_18,Rs_D_18,imm_D_18};
        Rd_D_18 = 0;
        Rs_D_18 = 0;
        alu_en_D_18 = 1'b0; //alu disabled
        reg_rw_D_18 = 1'b0; //register write disable
        mem_rw_D_18 = 1'b0; //memory write disable
        alu_ip_sel_18 = 2'b10; //2 for badd as input
if(Pre_Rd_18 == Rs_D_18 || Pre_Rd_18 == Rt_D_18 && pc_18 != 1)
        change_D = 1'b1; //change the input for ALU
else
        change_D = 1'b0;
end
        6'h16: //BGT
begin
        imm_D_18 = instD_18[21:6];
        alu_fn_D_18 = 2'b11; //01 for -
        alu_en_D_18 = 1'b1; //alu enabled
        reg_rw_D_18 = 1'b0; //register write disable
        mem_rw_D_18 = 1'b0; //memory write disable
        alu_ip_sel_18 = 2'b01; //1 for reg Rt as input
if(Pre_Rd_18 == Rs_D_18 || Pre_Rd_18 == Rt_D_18 && pc_18 != 1)
        change_D = 1'b1; //change the input for ALU
else
        change_D = 1'b0;
if(reg_rw_WB_18==1'b0)
begin
        alu_ip1_D_18 = rmem[Rd_D_18];
        alu_ip2_1_D_18 = rmem[Rs_D_18];
end
else
begin
repeat(1)
@(negedge clk_18);
        alu_ip1_D_18 = rmem[Rd_D_18];
        alu_ip2_1_D_18 = rmem[Rs_D_18];
end
end
endcase
if(op_D_18 != 6'h3a && op_D_18 != 6'h06)
begin
if(imm_D_18[15]==1'b1)
        imm_SE_D_18 = {16'hffff,imm_D_18};
else
        imm_SE_D_18 = {16'h0000,imm_D_18};
end
if(op_D_18 == 6'h16)
        Badd_SE_D_18 = imm_SE_D_18;
else if (op_D_18 == 6'h06)
begin
```

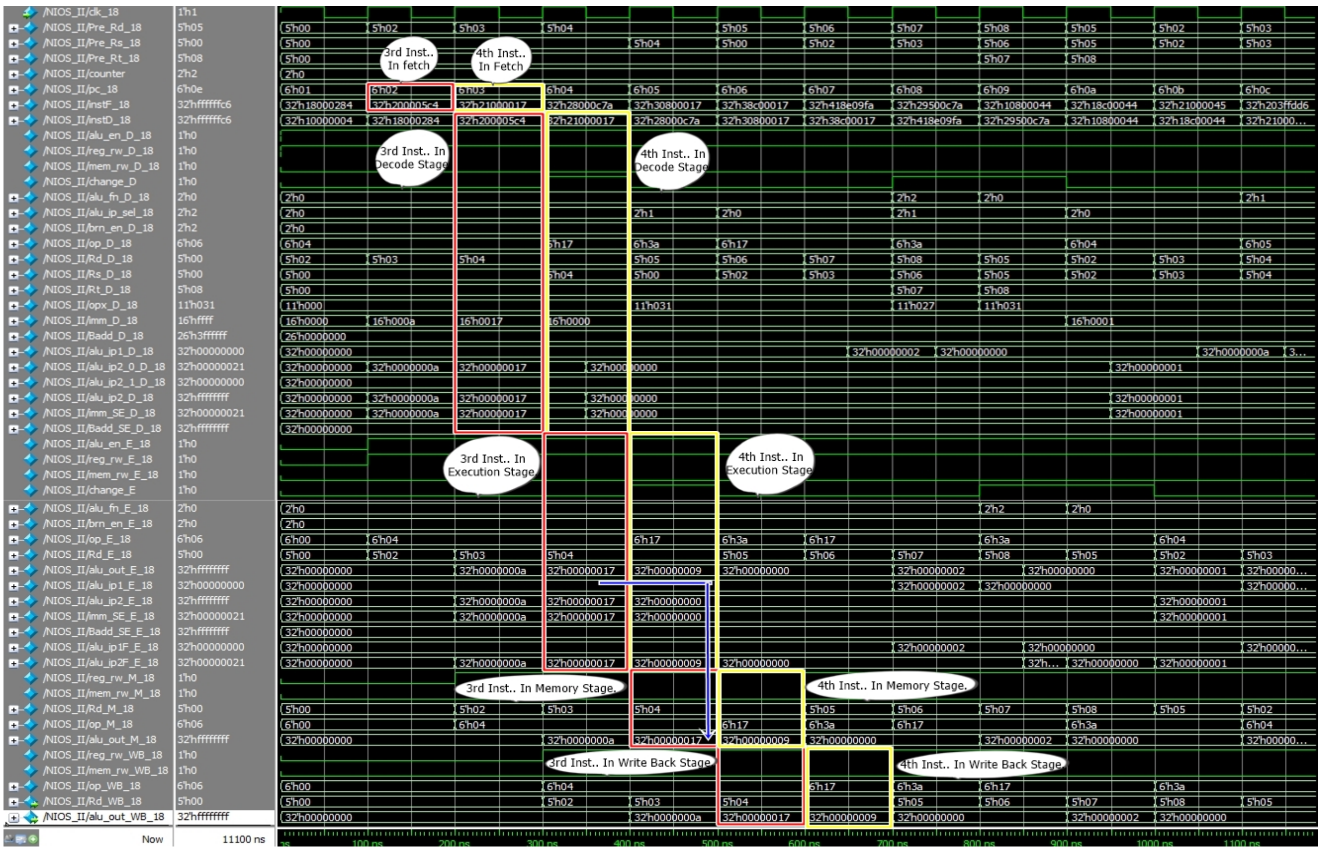
```
		if(Badd_D_18[25]==1'b1)
			Badd_SE_D_18 = {6'h3f,Badd_D_18};
		else
			Badd_SE_D_18 = {6'h00,Badd_D_18};
		end
			alu_ip2_0_D_18 = imm_SE_D_18;
		if(alu_ip_sel_18 == 2'b00)
			alu_ip2_D_18 = alu_ip2_0_D_18;
		else if (alu_ip_sel_18 == 2'b01)
			alu_ip2_D_18 = alu_ip2_1_D_18;
		else
			alu_ip2_D_18 = Badd_SE_D_18;
		end
		always @(alu_en_E_18,change_E,reg_rw_E_18,mem_rw_E_18,alu_fn_E_18,

	op_E_18,Rd_E_18,alu_ip1_E_18,alu_ip2_E_18,imm_SE_E_18,Badd_SE_E_18)//--------//
stage 3
		begin
			brn_en_E_18 = 2'b00;
		if(alu_en_E_18)
		begin
		if (change_E == 1'b1 && op_E_18 == 6'h17)
		begin
			alu_ip2F_E_18 = dmem[alu_out_M_18];
		end
		else if (change_E == 1'b1 && op_E_18 == 6'h3a && alu_fn_E_18 == 2'b10)
		begin
		if(reg_rw_WB_18==1'b0)
		begin
			alu_ip1F_E_18 = rmem[Rd_WB_18];
		end
		else
		begin
		repeat(1)
		@(negedge clk_18);
			alu_ip1F_E_18 = rmem[Rd_WB_18];
		end
			alu_ip2F_E_18 = alu_out_M_18;
		end
		else if (change_E == 1'b1 && op_E_18 == 6'h3a && alu_fn_E_18 == 2'b00)
		begin
			alu_ip1F_E_18 = alu_ip1_E_18;
			alu_ip2F_E_18 = alu_out_M_18;
		end
		else
		begin
			alu_ip1F_E_18 = alu_ip1_E_18;
			alu_ip2F_E_18 = alu_ip2_E_18;
		end
		case(alu_fn_E_18)
			2'b00:
```

```
                        alu_out_E_18 = alu_ip1F_E_18 + alu_ip2F_E_18;
                        2'b01:
                        alu_out_E_18 = alu_ip1F_E_18 - alu_ip2F_E_18;
                        2'b10:
                        alu_out_E_18 = alu_ip1F_E_18 * alu_ip2F_E_18;
                        2'B11:
                begin
                if(alu_ip1_E_18 > alu_ip2_E_18) begin
                        brn_en_E_18 = 2'b11;
                end
                else
                        brn_en_E_18 = 2'b00;
                end
                endcase
                end
                else
                begin
                        alu_out_E_18 = Badd_SE_E_18;
                        brn_en_D_18 = 2'b10;
                end
                end
                always @(reg_rw_M_18,mem_rw_M_18,Rd_M_18,alu_out_M_18,op_M_18)//------
-----------// stage 4
                begin
                if(mem_rw_M_18 == 1'b1 && op_M_18 == 6'h15)
                begin
                if(reg_rw_M_18 == 1'b0)
                        dmem[21] = rmem[Rd_M_18];
                end
                end
                always
@(reg_rw_WB_18,mem_rw_WB_18,Rd_WB_18,alu_out_WB_18,op_WB_18)//---------// stage 5
                begin
                if(reg_rw_WB_18 == 1'b1)
                        rmem[Rd_WB_18] = alu_out_WB_18;
                end
endmodule
```

• Test bench code:

```
`include "Nios_ii.v"
module NIOS_II_TB();
reg clk_18 = 0;
wire[31:0] alu_out_WB_18,Rd_D_18;
NIOS_II DUT(.clk_18(clk_18),.alu_out_WB_18(alu_out_WB_18),.Rd_WB_18(Rd_WB_18));
always
#5 clk_18 = ~clk_18;

initial begin

$dumpfile("nios.vcd");

$dumpvars(0, NIOS_II_TB);

end

endmodule
```

- Data forwarding between **movia r4, N & ldw r4,0(r4)**

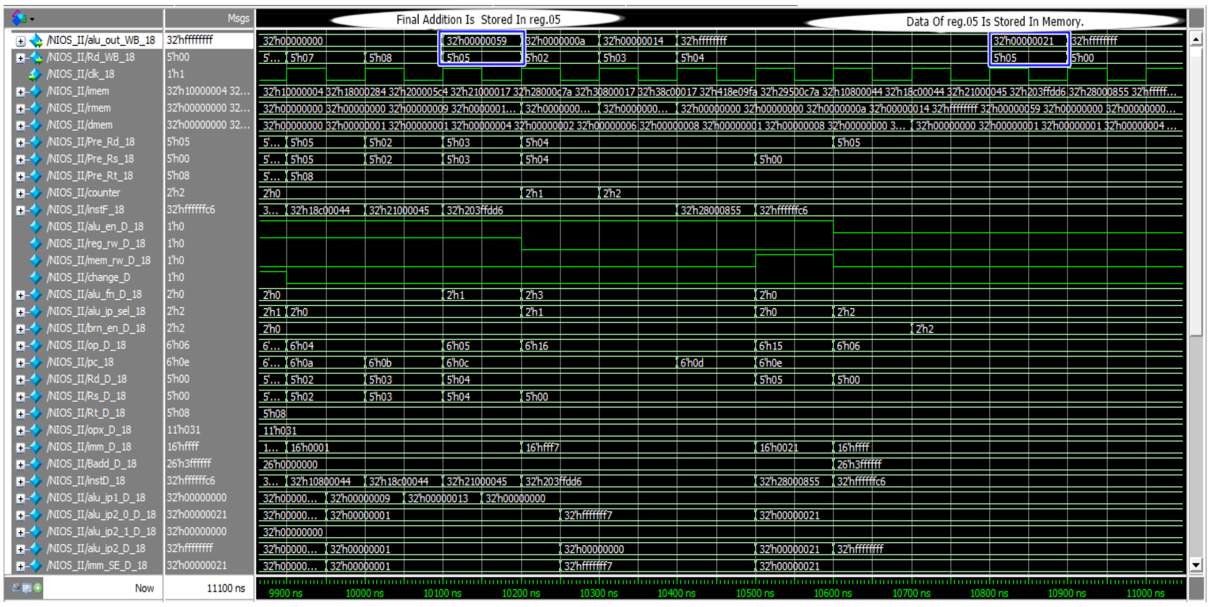

3rd Inst.. In fetch
4th Inst.. In Fetch
3rd Inst.. In Decode Stage
4th Inst.. In Decode Stage
3rd Inst.. In Execution Stage
4th Inst.. In Execution Stage
3rd Inst.. In Memory Stage.
4th Inst.. In Memory Stage.
3rd Inst.. In Write Back Stage
4th Inst.. In Write Back Stage
Now
11100 ns

Here, output of ALU of 3rd instruction is the location from which data is to be read by next instruction. Instead of waiting till the values to be written in Rd, it was forwarded to 4th instruction (shown as -->).

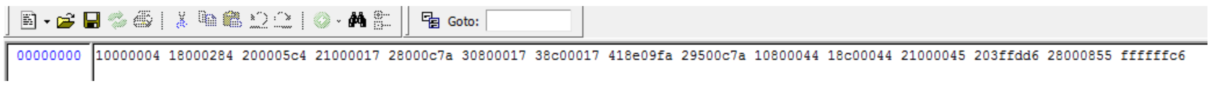
-Here, the output of ALU for instruction 3 is 23(h'00000017). Which is thelocation of data needed by 4th the next instruction.

-That is forwarded to 4th instruction before even writing to register memoryto get the data (rmem[32h'00000017] = 32h'00000009)

- **Final output of code:**

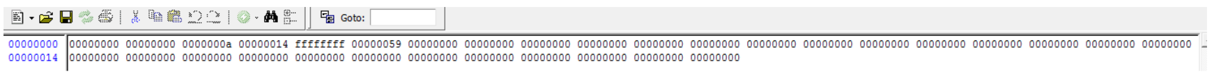

Final Addition Is Stored In reg.05
Data Of reg.05 Is Stored In Memory.
Msgs
/NIOS_II/alu_out_WB_18 32'hffffffff
/NIOS_II/Rd_WB_18 5'h00
/NIOS_II/clk_18 1'h1
/NIOS_II/imem 32'h10000004 32...
/NIOS_II/rmem 32'h00000000 32...
/NIOS_II/dmem 32'h00000000 32...
/NIOS_II/Pre_Rd_18 5'h05
/NIOS_II/Pre_Rs_18 5'h00
/NIOS_II/Pre_Rt_18 5'h08
/NIOS_II/counter 2'h2
/NIOS_II/instF_18 32'hffffffc6
/NIOS_II/alu_en_D_18 1'h0
/NIOS_II/reg_rw_D_18 1'h0
/NIOS_II/mem_rw_D_18 1'h0
/NIOS_II/change_D 1'h0
/NIOS_II/alu_fn_D_18 2'h0
/NIOS_II/alu_ip_sel_18 2'h2
/NIOS_II/brn_en_D_18 2'h2
/NIOS_II/op_D_18 6'h06
/NIOS_II/pc_18 6'h0e
/NIOS_II/Rd_D_18 5'h00
/NIOS_II/Rs_D_18 5'h00
/NIOS_II/Rt_D_18 5'h08
/NIOS_II/opx_D_18 11'h031
/NIOS_II/imm_D_18 16'hffff
/NIOS_II/Badd_D_18 26'h3ffffff
/NIOS_II/instD_18 32'hffffffc6
/NIOS_II/alu_ip1_D_18 32'h00000000
/NIOS_II/alu_ip2_0_D_18 32'h00000021
/NIOS_II/alu_ip2_1_D_18 32'h00000000
/NIOS_II/alu_ip2_D_18 32'hffffffff
/NIOS_II/imm_SE_D_18 32'h00000021
Now 11100 ns
9900 ns
10000 ns
10100 ns
10200 ns
10300 ns
10400 ns
10500 ns
10600 ns
10700 ns
10800 ns
10900 ns
11000 ns

- **Instruction memory:**



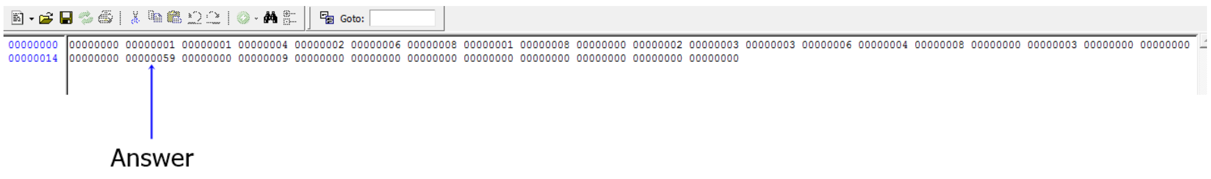
```
00000000  10000004 18000284 200005c4 21000017 28000c7a 30800017 38c00017 418e09fa 29500c7a 10800044 18c00044 21000045 203ffdd6 28000855 ffffffc6
```

- **Register memory at the end:**



```
00000000  00000000 00000000 0000000a 00000014 ffffffff 00000059 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000014  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

- **Data memory at the end:**



```
00000000  00000000 00000001 00000001 00000004 00000002 00000006 00000008 00000001 00000008 00000000 00000002 00000003 00000003 00000006 00000004 00000008 00000000 00000003 00000000 00000000
00000014  00000000 00000059 00000000 00000009 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

• Conclusion:

I implemented a Nios ii soft processor in Verilog to simulate vector dot product of 2 vectors namely
Avector:011426818
Bvector:233648030
And got the output at the 21st location in the data memory as “89=h00000059”

Without forwarding of data the same program took 250 clock cycle
with the clock duration of 10 ps.
After using data forwarding technique this program took only 110 clock cycle to complete the same task.