**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 BVECTOR = 16‘h000A**

**N = 19 LOOP = 5**

**DOT\_PRODUCT = 18 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 | - | 00000000 | 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 |

**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)

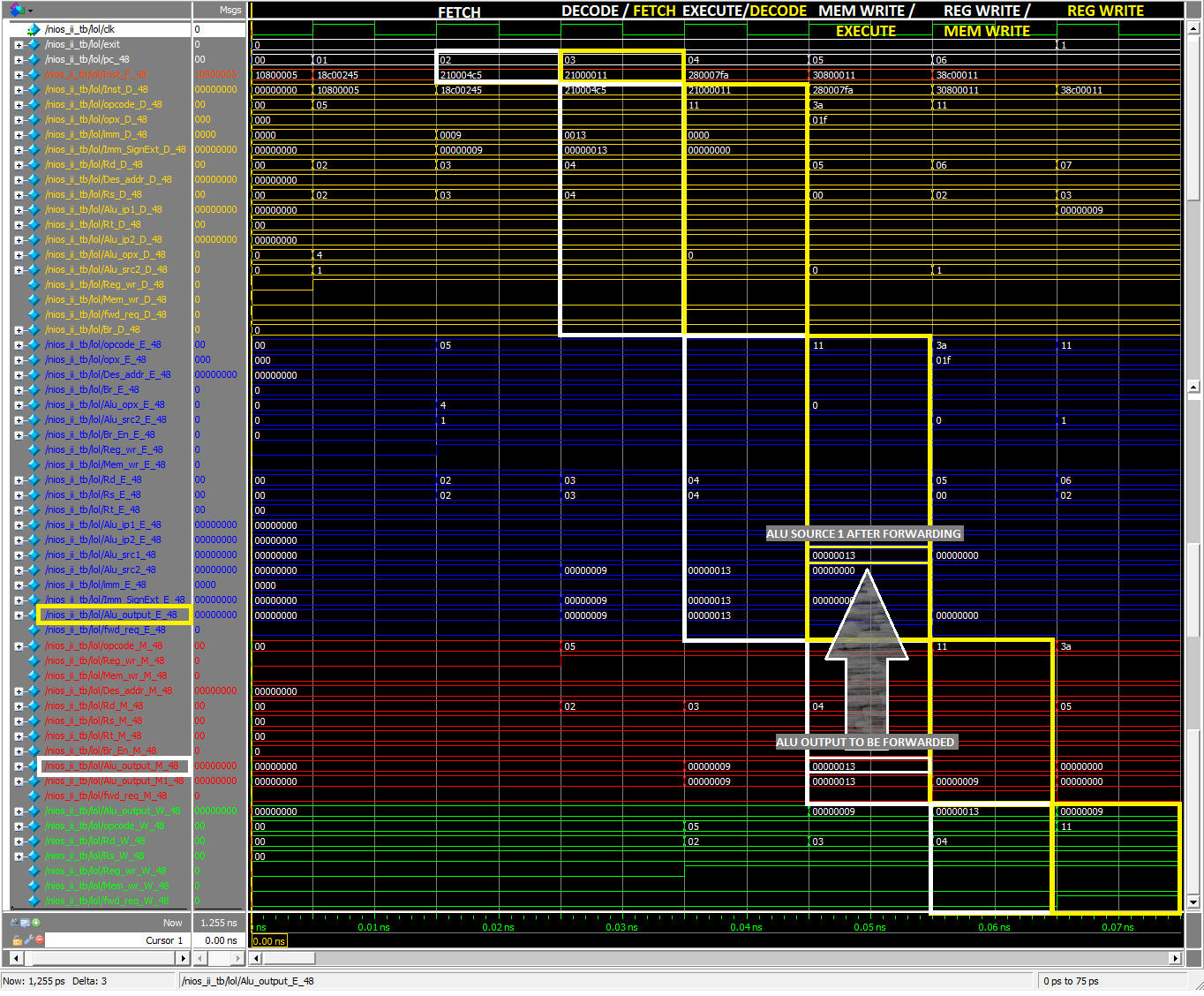
regmem\_48[Rd\_W\_48] = Alu\_output\_W\_48;

end

endmodule

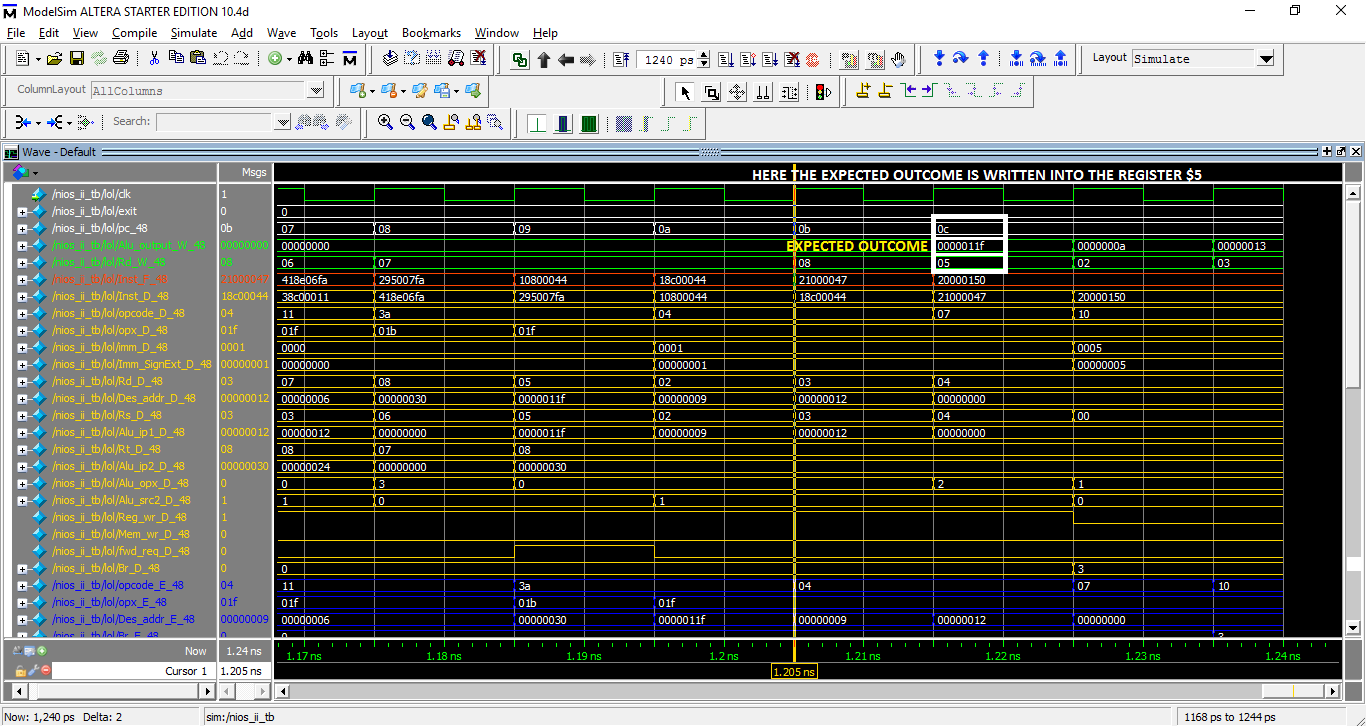
**OUTPUT:**

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

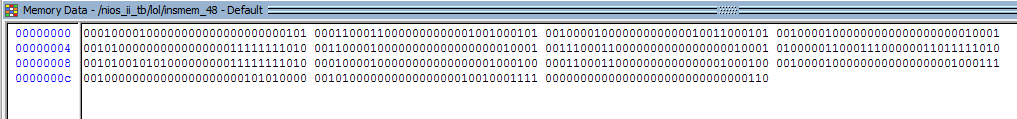
****

**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.**

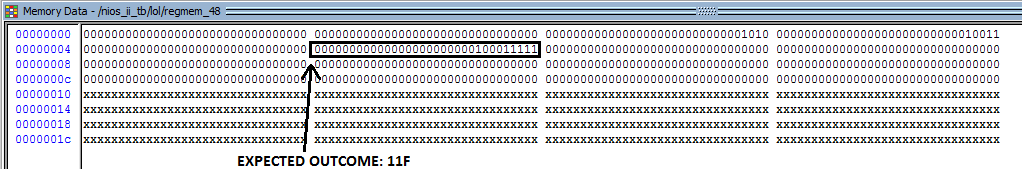
**FINAL OUTPUT:**

****

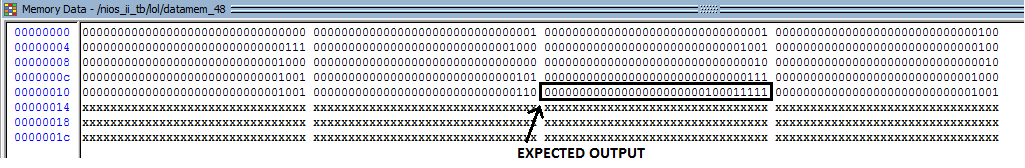
**INSTRUCTION MEMORY:**



**REGISTER MEMORY:**

****

**DATA MEMORY:**

****

**RESULT:**

Final answer (11F16 or 28710) is stored at datamem\_48[32’H12 or 18] in data memory.

**CONCLUSION:**

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.