DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION ENGINEERING

UNIVERSITY OF MORATUWA

In20-S5-EN3021 - Digital System Design

RISC-V Pipelined Processor

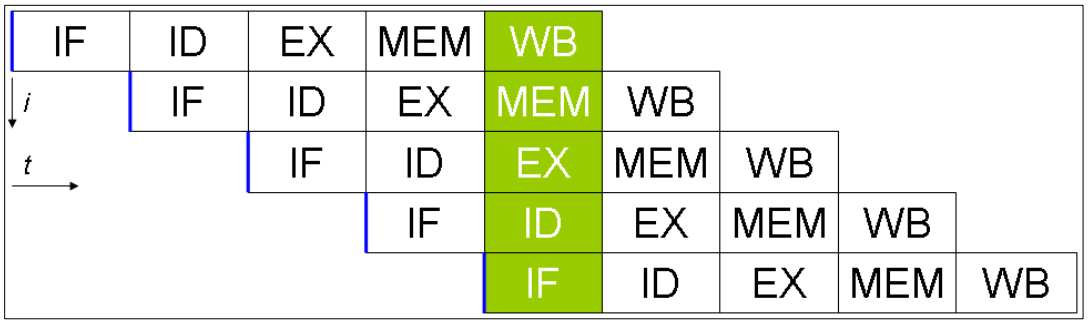
Arukgoda A.M.O.	**200051J**
Himeka S.H.D.	**200222K**
Wijethunga W.L.N.K.	**200733D**

Contents

Abstract

We have implemented the five main stages of risc-v pipelined processor in this, which are memory access, write back, instruction fetch, decode, and execution. Each stage is responsible for a specific part of instruction processing. This design processes one instruction at a time and executes multiple instructions simultaneously.

Five stage pipeline design



The processor operates through five key stages in its instruction execution pipeline.

1. Instruction Fetch (IF):

- Responsible for updating the Program Counter (PC) and fetching instructions.
- PC holds the address of the next instruction.
- A stall request is sent if correct instruction fetching is pending to avoid processing incorrect instructions.
- Outputs are directed to the IF/ID register for the next stage.

2. Instruction Decode (ID):

- Decodes RISC-V instructions, performs branch prediction, and accesses the register file.
- Manages potential stalls in the pipeline by forwarding stall requests if data dependencies exist.
- Outputs are forwarded to the ID/EX register for the Execute stage.

3. Execute (EX):

- Executes Arithmetic Logic Unit (ALU) operations, involving mathematical and logical operations.
- Requires nine inputs, including the result from the previous stage.
- Outputs are directed to the EX/MEM register for the Memory Access stage.

4. Memory Access (MEM):

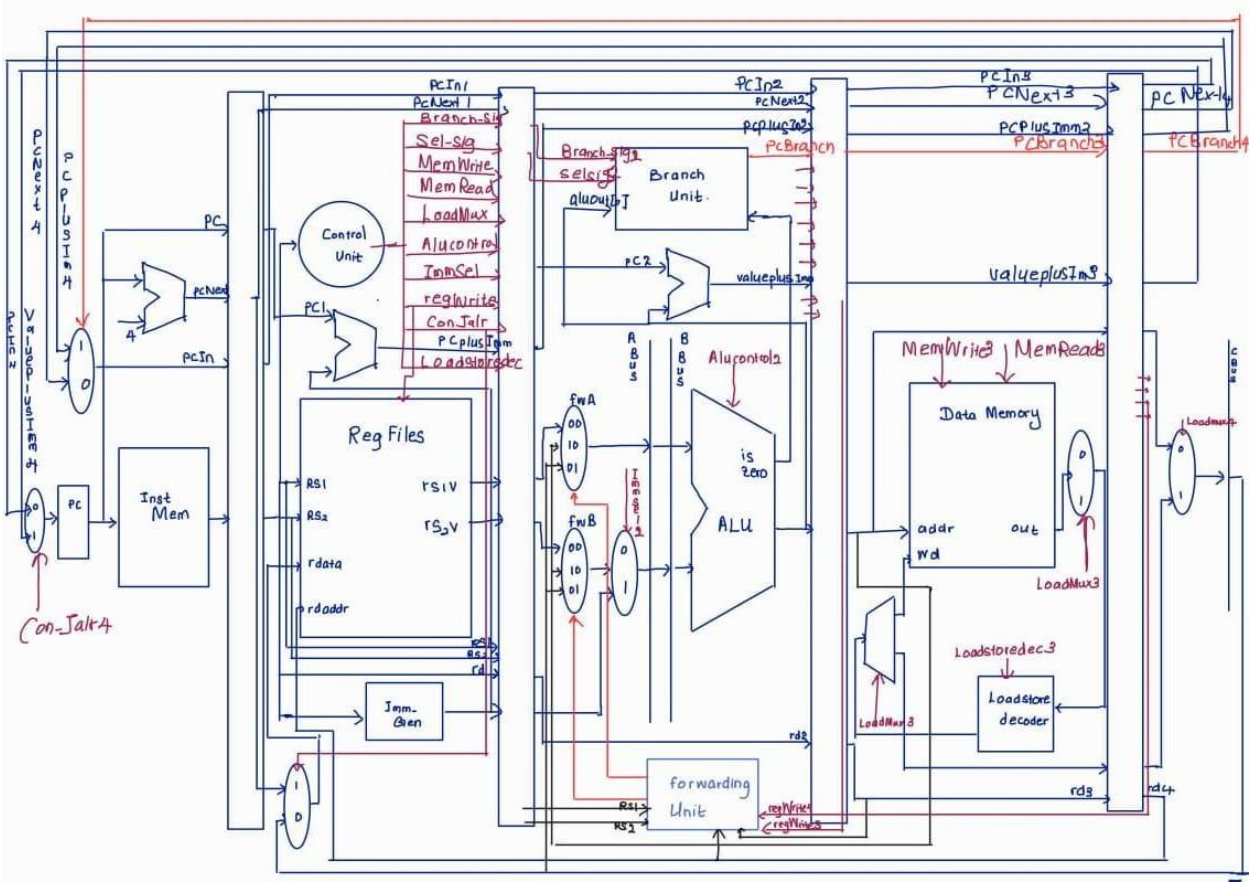
- Accesses data memory through the data cache, performing load and write operations.
- A stall request is forwarded if memory conditions are not met, preventing inaccurate instruction execution.
- Outputs are directed to the MEM/WB register for the Write Back stage.

5. Write Back (WB):

- Final stage where outputs from instruction execution are written back to the register files.

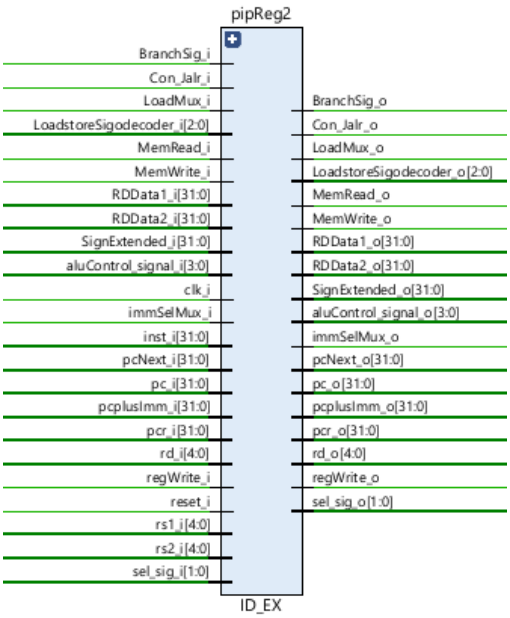
Throughout these stages, the pipeline is carefully managed to ensure accurate and efficient instruction processing, with each stage contributing to the seamless flow of data through the processor.

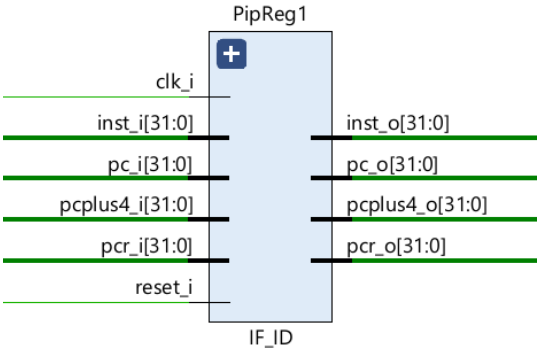
Datapath

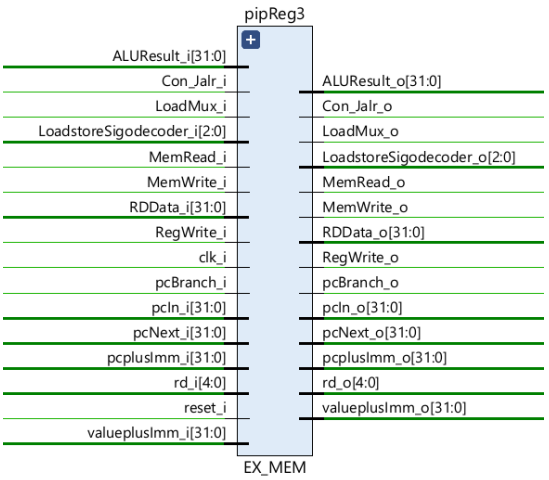


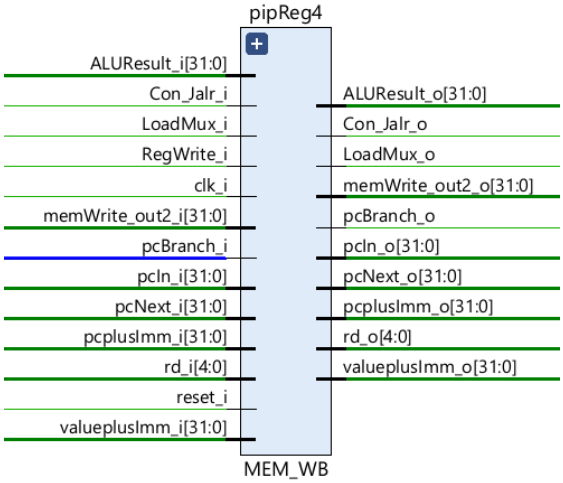
Intermediate Registers

Registers such as IF/ID, ID/EX, EX/MEM, and MEM/WB transfer data between different stages of the pipeline. These intermediary registers organize and synchronize the flow of data between pipeline stages, effectively minimizing data hazards and ensuring a consistent progression of information throughout the processor.









Hazards

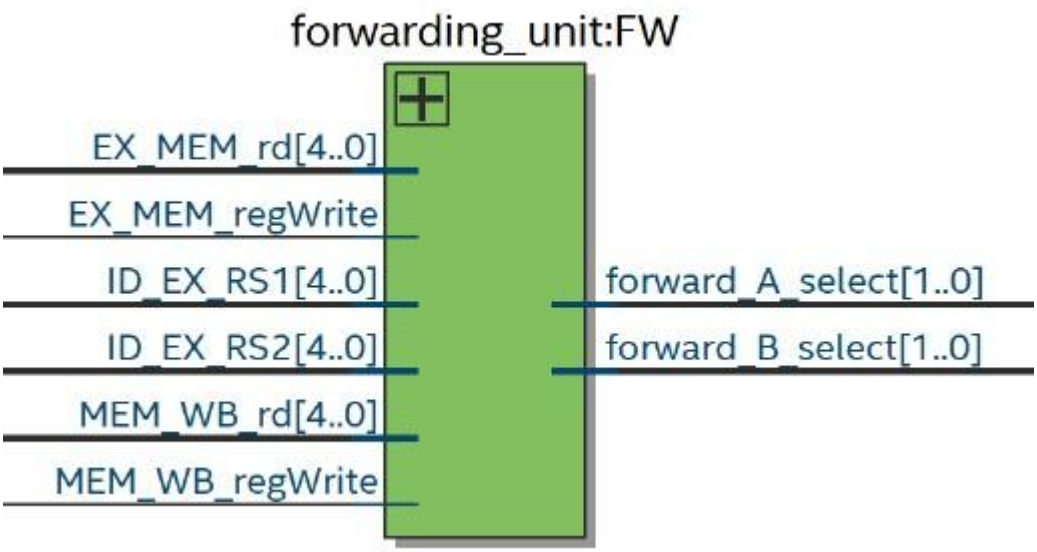
The pipeline architecture faces three types of dependencies, introducing potential hazards: structural, control, and data. Structural hazards arise due to resource conflicts within the pipeline, where the same hardware is accessed concurrently from different places in a clock cycle. This structural hazard poses risks of unpredictable outcomes if both stages access the same memory address concurrently. To mitigate this, the memory is divided into Instruction Memory (IMEM) and Data Memory (DMEM), addressing potential hazards by preventing simultaneous access.

Control dependencies, or branch hazards, stem from branch instructions. When fetching a branch instruction, the processor cannot determine it is a branch until after the DC stage and cannot ascertain

whether the branch is taken until the EX stage. Consequently, the fetch unit must retrieve an instruction every cycle, leading to instances where incorrect instructions are fetched and potentially executed, introducing hazards.

Within a pipelined architecture, three types of data hazards—Read-After-Write (RAW), Write-After-Read (WAR), and Write-After-Write (WAW)—can pose challenges

Data Forwarding



Data Forwarding is implemented to minimize the occurrence of stalls and enhance the efficiency of the processor by addressing data hazards. Although a stall controller can ensure accurate functionality, it significantly impacts performance. The control of data forwarding is managed within the instruction decode stage. The data forwarding mechanism involves forwarding register write data from both the Arithmetic Logic Unit (ALU) and memory at appropriate times. Two specific conditions are considered for data forwarding:

1. Read after Write (ALU):

- Triggered when there is a read operation, the ALU is writing, and the register address matches.

2. Read after Write (Memory):

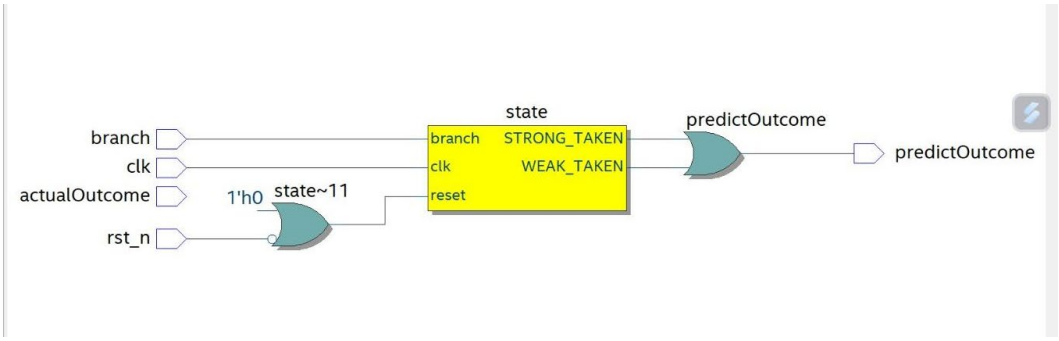
- Activated when there is a read operation, memory is writing , and the register address matches.

Stall Controller

The stall controller manages stall conditions in intermediate registers between stages, preventing the propagation of inaccurate data. Its primary responsibility is generating a 6-bit output value based on incoming stall requests. When specific stages issue stall requests, the stall controller activates the

corresponding bits in the output value. In the event of a stall request at a particular stage, the stall controller ensures that all preceding registers between that stage and the initial one are halted. By synchronizing the flow of information between stages, the stall controller is needed for averting data hazards and guaranteeing that each pipeline stage works with precise and consistent data.

Branch Predictor



The branch_predictor module determines the control signal `addermuxselect` based on the specified branch operation (`funct3`) and the comparison of two input data values (`readData1` and `b`).

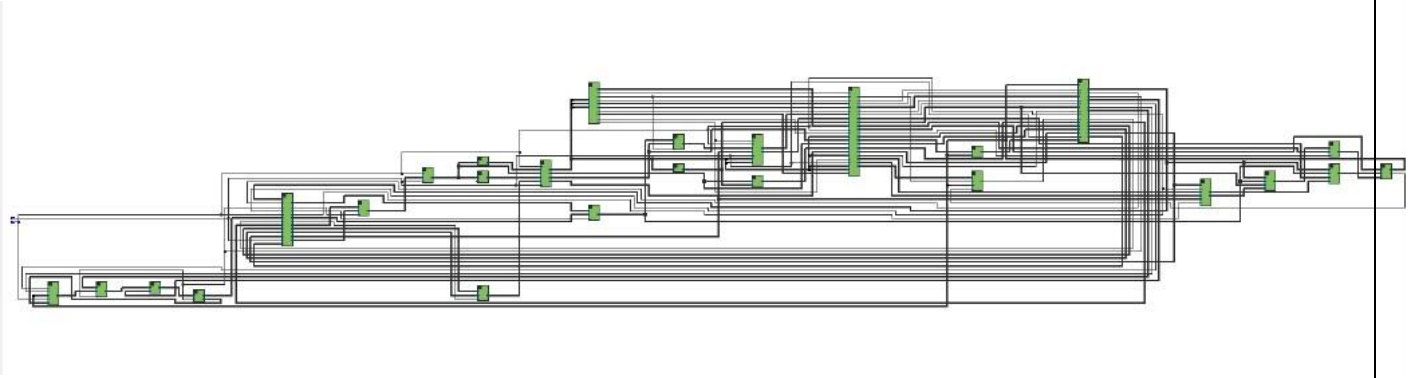
For the BEQ (Branch if Equal) operation with `funct3` equal to 3'b000, the logic dictates that if `readData1` is equal to `b`, then `addermuxselect` is set to 1; otherwise, it is set to 0.

Similarly, for the BLT (Branch if Less Than) operation with `funct3` equal to 3'b100, the module specifies that if `readData1` is less than `b`, then `addermuxselect` is set to 1; otherwise, it is set to 0.

For the BGT (Branch if Greater Than) operation with `funct3` equal to 3'b101, the logic dictates that if `readData1` is greater than `b`, then `addermuxselect` is set to 1; otherwise, it is set to 0.

Implementation

Netlist Viewer

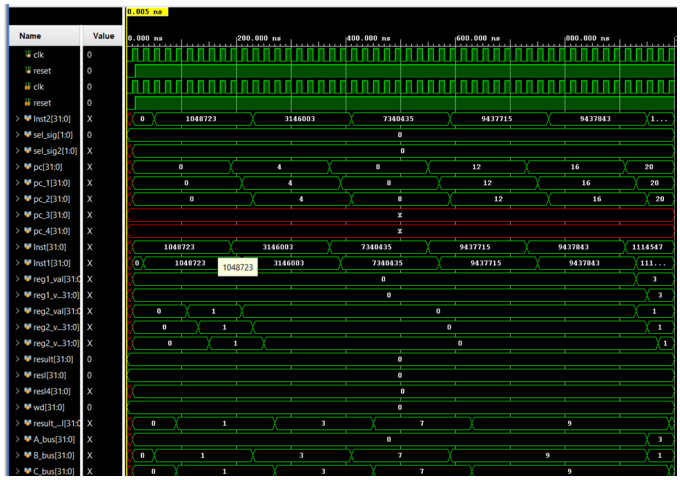


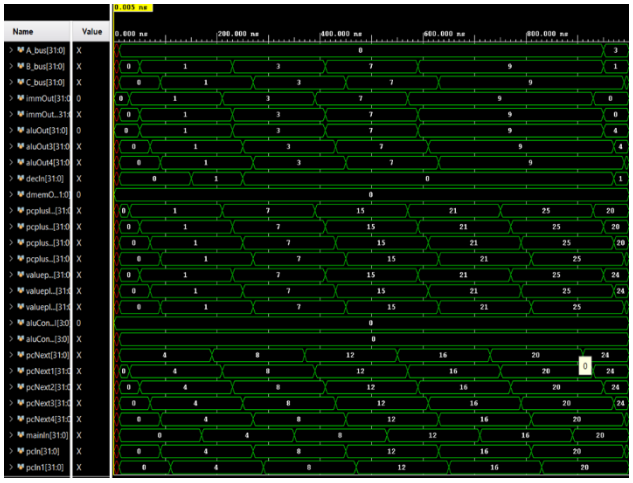
Simulation

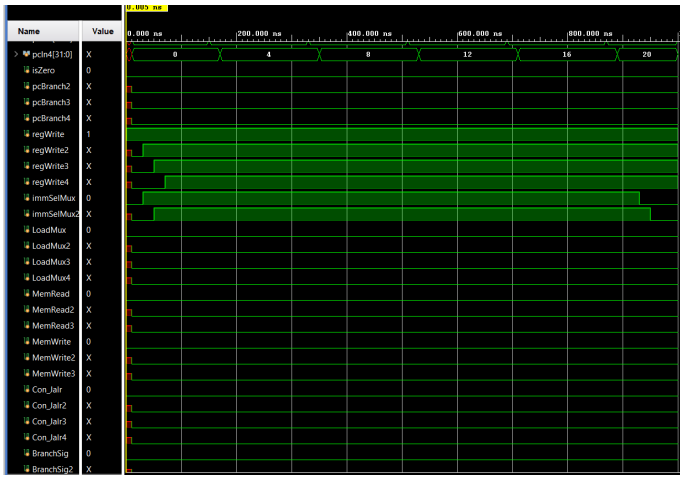
Pipeline Registers

Following instructions were simulated to implement the functionality of pipeline registers.

```
logic [INS_W - 1:0] Inst_memory [70:0];

 assign Inst_memory[0]   = 32'h00007033;//      and  r0,r0,r0           ALURes
assign Inst_memory[0]   = 32'h00100093;//      addi r1,r0, 1           ALUResu
assign Inst_memory[4]   = 32'h00300113; //addi x2, x0, 3
assign Inst_memory[8]   = 32'h00700193; //addi x3, x0, 7
assign Inst_memory[12]   = 32'h00900213;//addi x4, x0, 9
assign Inst_memory[16]   = 32'h00900293;//addi x5, x0, 9
assign Inst_memory[20]   = 32'h001101b3;//add x3, x2, x1  4 in reg3
 assign Inst_memory[24]   =32'h004121a3;  //sw x4, 3(x2) store 9 in mem 6
 assign Inst_memory[28]   = 32'h001101b3;//add x3, x2, x1  4 in reg3
 assign Inst_memory[32]   = 32'h001101b3;//add x3, x2, x1  4 in reg3
 assign Inst_memory[36]   = 32'h001101b3;//add x3, x2, x1  4 in reg3
 assign Inst_memory[40]   = 32'h00312403;//lw x8 3(x2) load 9 to reg 8
```





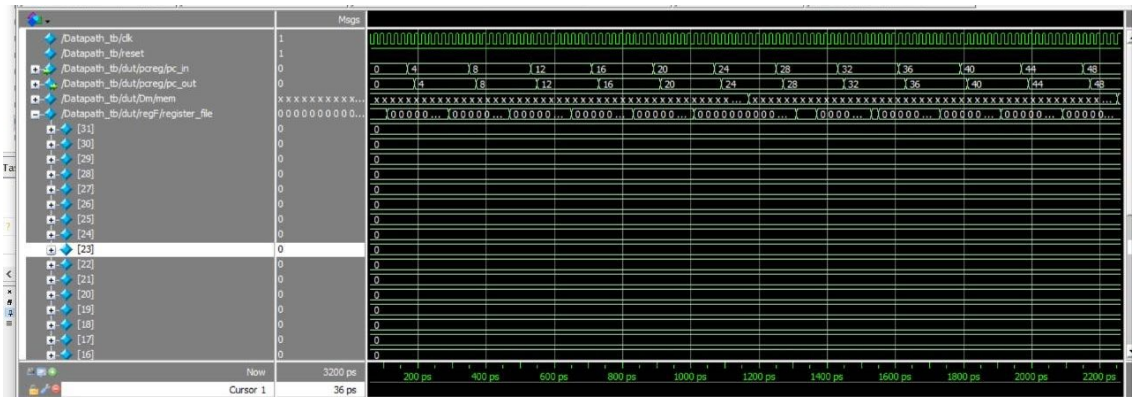


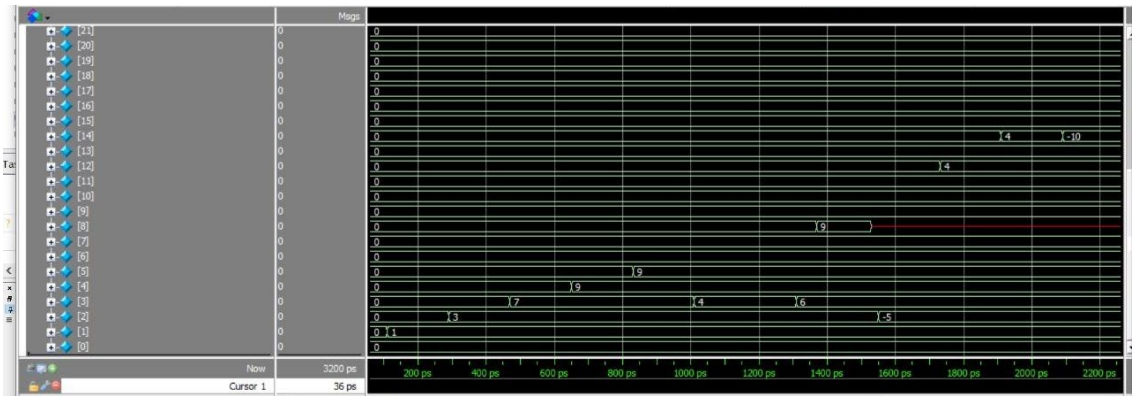
Data Forwarding Unit

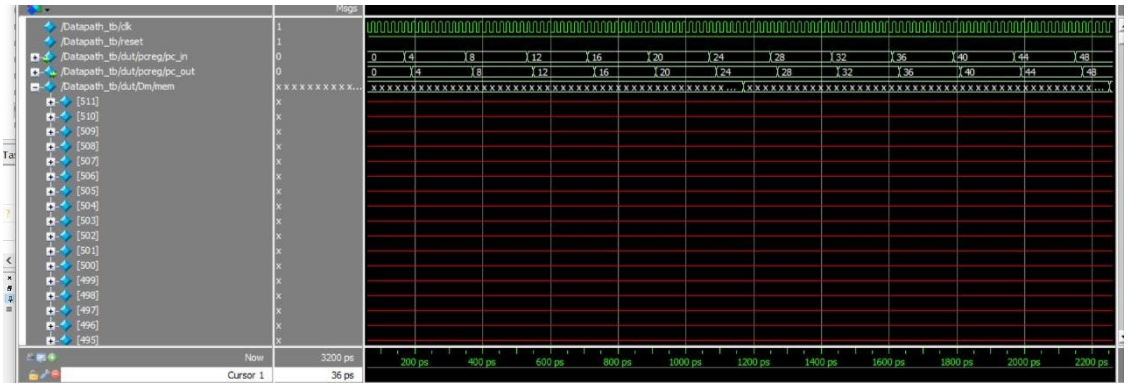
This is the instruction set we used to implement the data forwarding unit.

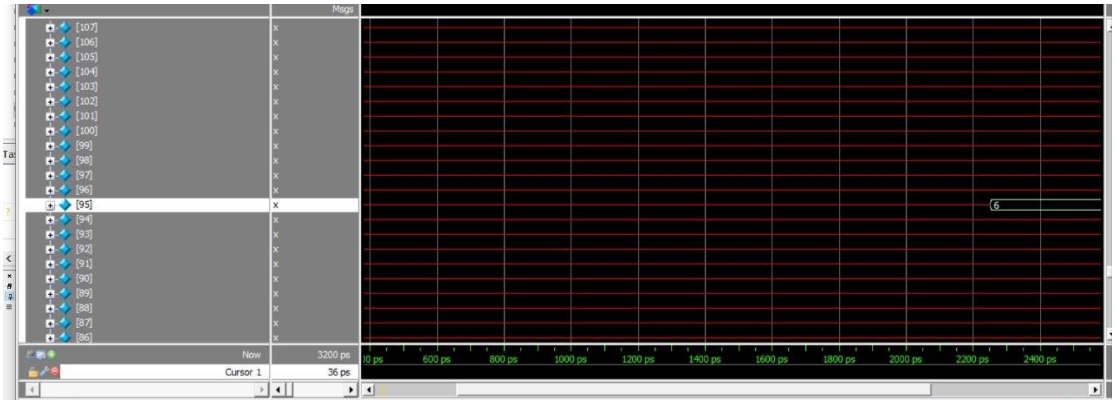
```
    //assign Inst_memory[0]   = 32'h00007033;//      and  r0,r0,r0          ALUResult = h0 =
    assign Inst_memory[0]   = 32'h00100093;//      addi r1,r0, 1          ALUResult = h1 = r1
    assign Inst_memory[4]   = 32'h00300113; //addi x2, x0, 3
    assign Inst_memory[8]   = 32'h00700193; //addi x3, x0, 7
    assign Inst_memory[12]   = 32'h00900213;//addi x4, x0, 9
    assign Inst_memory[16]   = 32'h00900293;//addi x5, x0, 9
    assign Inst_memory[20]   = 32'h001101b3;//add x3, x2, x1  4 in reg3
    assign Inst_memory[24]   =32'h004121a3;  //sw x4, 3(x2) store 9 in mem 6
//  assign Inst_memory[28]   = 32'h001101b3;//add x3, x2, x1  4 in reg3
//  assign Inst_memory[32]   = 32'h001101b3;//add x3, x2, x1  4 in reg3
//  assign Inst_memory[36]   = 32'h001101b3;//add x3, x2, x1  4 in reg3
    assign Inst_memory[28]   = 32'h00312403;//lw x8 3(x2) load 9 to reg 8
    assign Inst_memory[32]   =  32'h40308133;//sub x2, x1 ,x3 //-5 in reg2
    assign Inst_memory[36]   =  32'h00510633;//add x12 ,x2,x5 //4 in reg12   12
    assign Inst_memory[40]   =  32'h00220733;//add x14, x4,x2 // 4 in reg14
    assign Inst_memory[44]   =  32'h00210733;//add x14, x2,x2 // -10in reg14
    assign Inst_memory[48]   =  32'h06312223;//sw x3 ,100(x2) //6 in mem 95


//  assign Inst_memory[0] =32'b0000000_00010_00000_000_00001_0010011; //addi x1, x0, 2
//    assign Inst_memory[4] =32'b0000000_00001_00000_000_00010_0010011; //addi x2, x0, 1
```

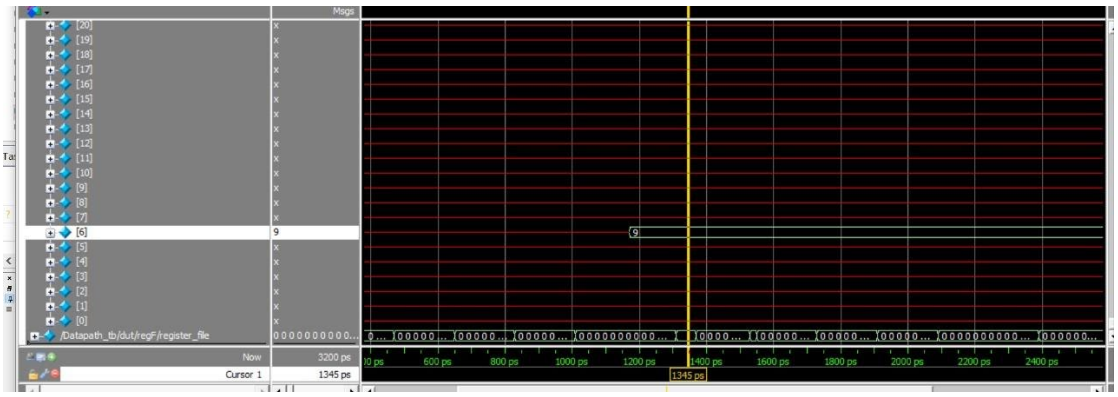
Following simulation results were obtained.






/Datapath_tb/clk
/Datapath_tb/reset
/Datapath_tb/dut/pcreg/pc_in
/Datapath_tb/dut/pcreg/pc_out
/Datapath_tb/dut/Dm/mem
Now
3200 ps
Cursor 1
36 ps


Now
3200 ps
Cursor 1
36 ps
6


/Datapath_tb/dut/regF/register_file
Now
3200 ps
Cursor 1
1345 ps
9

Resource Utilization Report

Analysis & Synthesis Resource Usage Summary	
Resource	**Usage**
Estimated Total logic elements	40,734
Total combinational functions	23934
Logic element usage by number of LUT inputs	
-- 4 input functions	23377
-- 3 input functions	302
-- <=2 input functions	255
Logic elements by mode	
-- normal mode	23867
-- arithmetic mode	67
Total registers	17157
-- Dedicated logic registers	17157
-- I/O registers	0
I/O pins	80
Embedded Multiplier 9-bit elements	0
Maximum fan-out node	clk~input
Maximum fan-out	17157
Total fan-out	146393
Average fan-out	3.55

Compilati	Combinati	Dedicated	Memory B	DSP Eleme	DSP 9x9	DSP 18x18	Pins	Virtual Pins	Full Hierarchy Name	Entity Name	Library Name
a	13302 (13	16797 (0)	0	0	0	0	21	0	|try	try	work
|Clock_(	43 (43)	29 (29)	0	0	0	0	0	0	|try|Clock_divider:cd	Clock_divider	work
|Datapa	13232 (0)	16768 (0)	0	0	0	0	0	0	|try|Datapath:dp	Datapath	work
|ALUn	1144 (523	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu	ALUnitHW	work
|lpm	621 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0	lpm_mult	work
|m	621 (289)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core	multcore	work
|	272 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder	mpar_add	work
	32 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	lpm_add_sub	work
	32 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	addcore	work
	32 (32)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work
	28 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	lpm_add_sub	work
	28 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	addcore	work
	28 (28)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work
	24 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	lpm_add_sub	work
	24 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	addcore	work
	24 (24)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work
	20 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	lpm_add_sub	work
	20 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	addcore	work
	20 (20)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work
	16 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	lpm_add_sub	work
	16 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	addcore	work
	16 (16)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work
	12 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	lpm_add_sub	work
	12 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	addcore	work
	12 (12)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work
	8 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	lpm_add_sub	work
	8 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	addcore	work
	8 (8)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work
	4 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	lpm_add_sub	work
	4 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	addcore	work
	4 (4)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work

4 (4)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|lpm_add_sub:a	a_csnbuffer	work
128 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	mpar_add	work
30 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
30 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
30 (30)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
22 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
22 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
22 (22)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
14 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
14 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
14 (14)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
6 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
6 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
6 (6)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
56 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	mpar_add	work
26 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
26 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
26 (26)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
10 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
10 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
10 (10)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
20 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	mpar_add	work
18 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
18 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
18 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
18 (18)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
2 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	mpar_add	work
2 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
2 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
2 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
2 (2)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
60 (60)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mul_boothc:booth_enc	mul_boothc	work

26 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
26 (26)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
10 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
10 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
10 (10)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
20 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	mpar_add	work
18 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
18 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
18 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
18 (18)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
2 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	mpar_add	work
2 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	lpm_add_sub	work
2 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
2 (0)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	addcore	work
2 (2)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mpar_add:padder|mpar_add:sub	a_csnbuffer	work
60 (60)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|ALUnitHW:alu|lpm_mult:Mult0|multcore:mult_core|mul_boothc:booth_enc	mul_boothc	work
11 (11)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|BranchSel:bs	BranchSel	work
11450 (11	16384 (16	0	0	0	0	0	0	|try|Datapath:dp|Datamemory:Dm	Datamemory	work
1 (1)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|Decoder3by3:dc	Decoder3by3	work
42 (42)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|InstructionMemory:Imem	InstructionMemor	work
26 (26)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|Microprogramming:CU	Microprogrammin	work
33 (33)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|Mux2input:muxImmSel	Mux2input	work
31 (31)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|Mux2input:muxJALR	Mux2input	work
74 (74)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|Mux2input:muxrdsel	Mux2input	work
316 (316)	352 (352)	0	0	0	0	0	0	|try|Datapath:dp|RegFiles:regF	RegFiles	work
30 (30)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|adder_32:adder1	adder_32	work
64 (64)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|adder_32:adder3	adder_32	work
10 (10)	0 (0)	0	0	0	0	0	0	|try|Datapath:dp|imm_Gen:immGen	imm_Gen	work
0 (0)	32 (32)	0	0	0	0	0	0	|try|Datapath:dp|pc:pcreg	pc	work
7 (7)	0 (0)	0	0	0	0	0	0	|try|hexdigit:H2	hexdigit	work
7 (7)	0 (0)	0	0	0	0	0	0	|try|hexdigit:H3	hexdigit	work

Codes for physical implementation on FPGA

```
module try(
    input logic clk, reset,
    input logic [4:0] swval,
         input logic [8:0] extmemaddress,
    output logic [6:0] HEX4,
    output logic [6:0] HEX5,

         output logic [6:0] HEX1,
    output logic [6:0] HEX2



);

logic [31:0] rdval, extmemdata;


logic [7:0] bcdrd;
logic [7:0] bcddata;

Datapath dp (
    .clk(clk1),
    .reset(reset),
    .swval(swval),
         .extmemaddress(extmemaddress),
    .rdval(rdval),
         .extmemdata(extmemdata)
);

Clock_divider cd(
        .clock_in(clk),
        .clock_out(clk1)
);

Binary_to_BCD uut (
    .bin(rdval),
    .bcd(bcdrd)
  );
```

```
Binary_to_BCD uut (
    .bin(rdval),
    .bcd(bcdrd)
  );
Binary_to_BCD uut1 (
    .bin(extmemdata),
    .bcd(bcddata)
  );


hexdigit H4 (bcdrd[3:0], HEX4);
hexdigit H5 (bcdrd[7:4], HEX5);

hexdigit H1 (bcddata[3:0], HEX1);
hexdigit H2 (bcddata[7:4], HEX2);



endmodule
```

```
module Binary_to_BCD(
   input logic [31:0] bin,
   output logic [7:0] bcd

   );

integer i;

always @(bin) begin
    bcd=0;
    for (i=0;i<32;i=i+1) begin                              //Iterate once for each bit in input number
        if (bcd[3:0] >= 5) bcd[3:0] = bcd[3:0] + 3;         //If any BCD digit is >= 5, add three
        if (bcd[7:4] >= 5) bcd[7:4] = bcd[7:4] + 3;
        bcd = {bcd[6:0],bin[31-i]};                         //Shift one bit, and shift in the proper bit from input
    end
end
endmodule
```

```
module hexdigit(
        input logic [3:0] in,
        output reg [6:0] out
);

always @*
begin
        out= 7'b1111111;

        case(in)

                4'h0: begin
                out = 7'b1000000;
                end

                4'h1:begin
                out = 7'b1111001;
                end

                4'h2: begin
                out = 7'b0100100;
                end

                4'h3: begin
                out = 7'b0110000;
                end

                4'h4: begin
                out = 7'b0011001;
                end

                4'h5: begin
                out = 7'b0010010;

                end
```

Task Allocation

Himeka H.S.D.	Pipeline (EX/MEM, MEM/WB) Data forwarding
Wijethunga W.L.N.K.	Branch prediction
Arukgoda A.M.O.	Pipeline (IF/ID, ID/EX)