Design of a 5-stage Pipelined RISC-V Processor

Sri Sai Nomula
SR No: 22986, MTech ESE,

I OBJECTIVE

The Processor is a 32-bit RISC CPU with 32-bit Address space with RISC type instructions. Hazard detection, Forwarding (from EX, MEM and WB stage outputs) and the Stall unit for Load and Branch instructions are implemented. The Instruction Cache is Direct-Mapped with a Block Size of Four Words. The main Instruction Memory is implemented in Block RAM. The Processor supports basic Arithmetic and Logic instructions, LUI, Branch Instructions, and Load-Store instructions.

II SPECIFICATIONS

- 32-bit Data width and 32-bit Instruction width.
- 32 General Purpose Registers- R0 to R31 (32 bit each).
- Instructions supported- ADD, SUB, AND, OR, LOAD, STORE, BGE, BEQ, LUI, ADDI.

III INSTRUCTION FORMAT

The first 7 bits of the instruction are used to define the instruction, and the rest of bits are used to define the function or immediate values and Register addresses. The table below shows how each instruction is divided.

Instruction	[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
R-Type	Funct7	RS2	RS1	Funct3	RD	OpCode
Load	Imm(11:5)	Imm(4:0)	RS1	000	RD	OpCode
Store	Imm(11:5)	RS2	RS1	000	Imm(4:0)	OpCode
ADDI	Imm(11:5)	Imm(4:0)	RS1	000	RD	OpCode
BEQ	Imm(11:4)	RS2	RS1	000	Imm(3:0|10)	OpCode
BGE	Imm(11:4)	RS2	RS1	101	Imm(3:0|10)	OpCode
LUI	Imm(31:25)	Imm(24:20)	Imm(19:15)	Imm(14:12)	RD	OpCode

Table 1: Instruction Set Encoding

1. **R-Type Instruction:** The OpCode is 0110011 and the Funct3 field describes the operation ALU should perform (000 - ADD or SUB based on Funct7, 010 - AND, 011 - OR , 100 - XOR). RS1, RS2 and RD fields provides the Register Values which are of 5-bits each. Example - SUB R1 R2 R3 [R1 <- R2 - R3]. Here, Funct3 = ADD(000) and Funct7 = SUB(0100000),RS2 = 00011 , RS1 = 00010 and RD = 00001.
2. **Load-Type Instruction:** The OpCode is 0000011 and the Imm field gives the Value which needs to added to RS1 and access that particular memory location. RS1 and RD fields provides the Register Values which are of 5-bits each. Example - LOAD R2 R1 101 [R2 <- mem[R1 + 101]]. Here, Imm = 101, RS1 = 00001 and RD = 00010.
3. **Store-Type Instruction:** The OpCode is 0100011 and the Imm field gives the Value which needs to added to RS1 and store into that particular memory location. RS1 and RD fields provides the Register Values which are of 5-bits each. Example - STORE R2 R1 101 [R2 -> mem[R1 + 101]]. Here, Imm = 101, RS1 = 00001 and RD = 00010.
4. **ADDI-Type Instruction:** The OpCode is 0010011 and the Imm field gives the Value which needs to added to RS1 and store into RD. RS1 and RD fields provides the Register Values which are of 5-bits each. Example - ADDI R2 R1 101 [R2 <- R1 + 101]. Here, Imm = 101, RS1 = 00001 and RD = 00010.
5. **BEQ(Branch on Equal)-Type Instruction:** The OpCode is 1100011 and the Imm field gives the Value where PC needs to points in case condition satisfies. RS1 and RS2 fields provides the Register Values which are of 5-bits each. Example - BEQ R3 R4 1101 [If R3=R4 then Fetch Instruction from Memory Location 1101]. Here, Imm = 1101, RS1 = 00011 and RS2 = 00100.

6. **BGE(Branch on Greater)-Type Instruction:** The OpCode is 1100011 and the Imm field gives the Value where PC needs to points in case condition satisfies. RS1 and RS2 fields provides the Register Values which are of 5-bits each. Example - BGE R3 R4 1101 [If R3 > R4 then Fetch Instruction from Memory Location 1101]. Here, Imm = 1101, RS1 = 00011 and RS2 = 00100.

7. **LUI-Type Instruction:** The OpCode is 0110111 and the Imm field gives the Upper 12-bits Value that needs to be stored in RD. Example - LUI ABCDE R3[R3 = ABCDE000]. Here, Imm = ABCDE, RD = 00011

IV DATA PATH AND CONTROL PATH

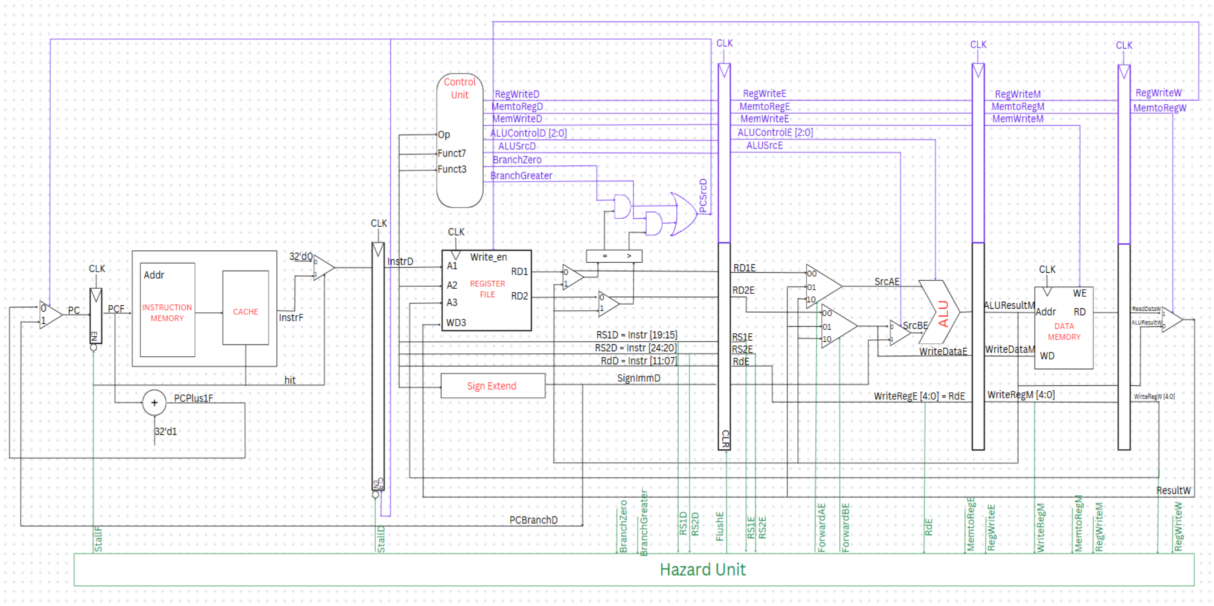


Figure 1: Pipelined Processor with Full Hazard Handling

The Functionality of various control signals are described in the Table Below. The values of these signals change based on the OpCode and the Task they are performing described by the Controller. Most of the control information comes from the opcode, but R-type instructions also use the funct7 and funct3 field to determine the ALU operation. Thus, we will simplify our design by factoring the control unit into two blocks of combinational logic. The main decoder computes most of the outputs from the opcode. It also determines a 2-bit ALUOp signal. The ALU decoder uses this ALUOp signal in conjunction with the funct7 field to compute ALUControl.

Signal	Function
RegWriteD	To Write the data into Register File
MemtoRegD	To Select the data to be written into Register File between ALUResult and Data from Memory.
MemWriteD	To Write the data into Data Memory.
ALUControlD	To Perform the desired operation of ADD, SUB, AND, OR.
ALUSrcD	To Select Operand-2 to ALU between Register Value and Immediate Value.
BranchZero	Used while performing Branch Equal Operation. So, that when Branch is Equal the Zout is set.
BranchGreater	Used while performing Branch Greater than Operation. So, that when Branch is Greater then Gout is set.
PCSrcD	To Select the PCNext Value between PC + 1 and Immediate Address specified for Branch and Jump Instructions.

Table 2: Description of Control Signals

The control signals for each instruction were described as we built the datapath. Table 3 is a truthtable for the main decoder that summarizes the control signals as a function of the opcode.

Instruction	OpCode	MemtoRegD	MemWriteD	BranchZ	BranchGr	RegWriteD	ALUSrcD
R-Type	0110011	0	0	0	0	1	0
LOAD	0000011	1	0	0	0	1	1
STORE	0100011	0	1	0	0	1	1
ADDI	0010011	0	0	0	0	1	1
BEQ	1100011	0	0	1	0	0	0
BGE	1100011	0	0	0	1	0	0
LUI	0110111	0	0	0	0	X	1

Table 3: Main Decoder Truth Table

V ASSEMBLY PROGRAM

The below assembly program is used to find the maximum number from a set of ten random numbers.This assembly code is written using the instructions declared for this 32-bit RISC-V architecture. The instructions are stored in the first 9 locations of the Instruction Memory and the 10 random numbers are stored in the 10 locations of Data Memory and the final result(Maximum Number) is stored in the R3 of Register File.

Address	Instruction	Representation[HEX Format]
0000	ADDI R1 R0 1	00100093
0001	ADDI R2 R0 10	00A00113
0002	LOAD R3 R0 0	00000183
LOOP : 0003	BEQ R1 R2 STOP	00208E63
0004	LOAD R4 R1 0	00008203
0005	ADDI R1 R1 1	00108093
0006	BGE R3 R4 LOOP	0041D563
0007	ADD R3 R4 R0	000201B3
0008	BEQ R1 R1 LOOP	00108563
STOP : 0009	HALT	

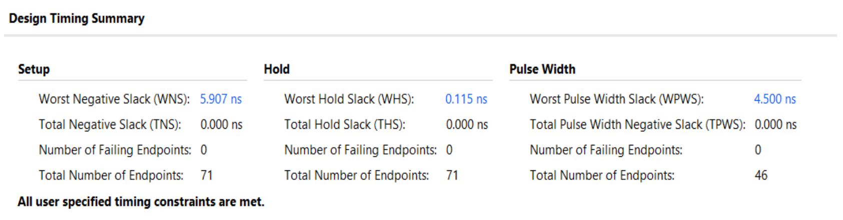
Table 4: Assembly Program

VI TIMING REPORT

The Worst Negative Slack (WNS) is 5.907ns at an operating Time Period of 15ns. Therefore,the Maximum Operating Frequency can be calculated as:

$$f_{max} = \frac{1}{15ns - 5.907ns} = \frac{1}{9.093} = 109.97MHz. \tag{1}$$

The Timing Report is shown in Fig.2



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.907 ns	Worst Hold Slack (WHS):	0.115 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	71	Total Number of Endpoints:	71	Total Number of Endpoints:	46

All user specified timing constraints are met.

Figure 2: Timing Report for Pipelined Processor

VII SIMULATION RESULTS

The Waveform after Simulation is shown in Fig.3

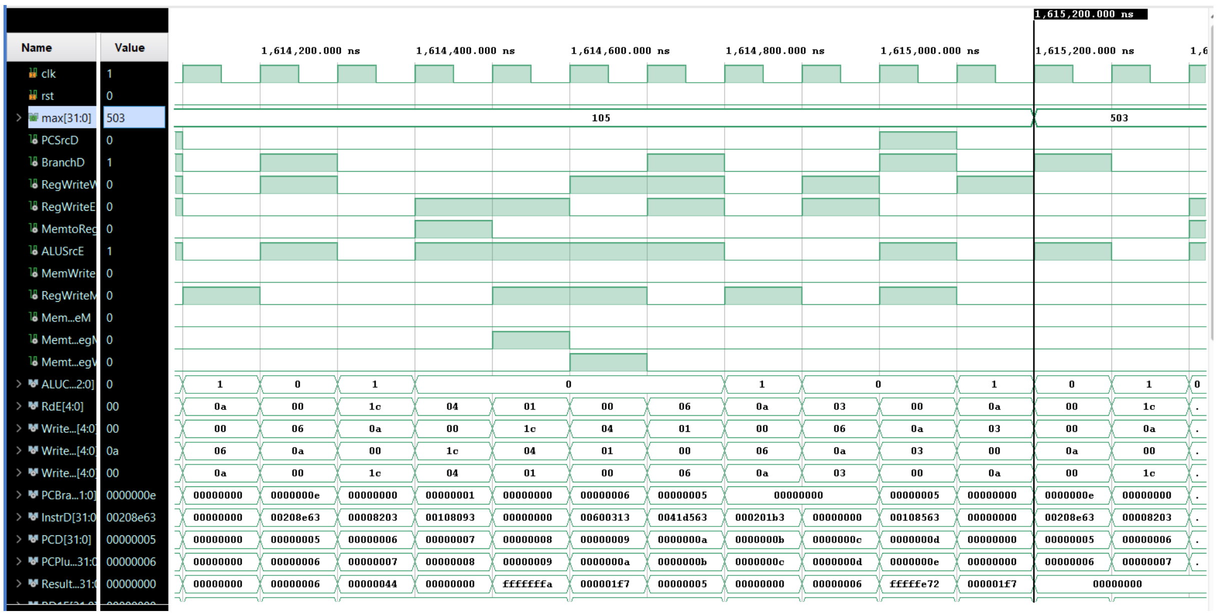
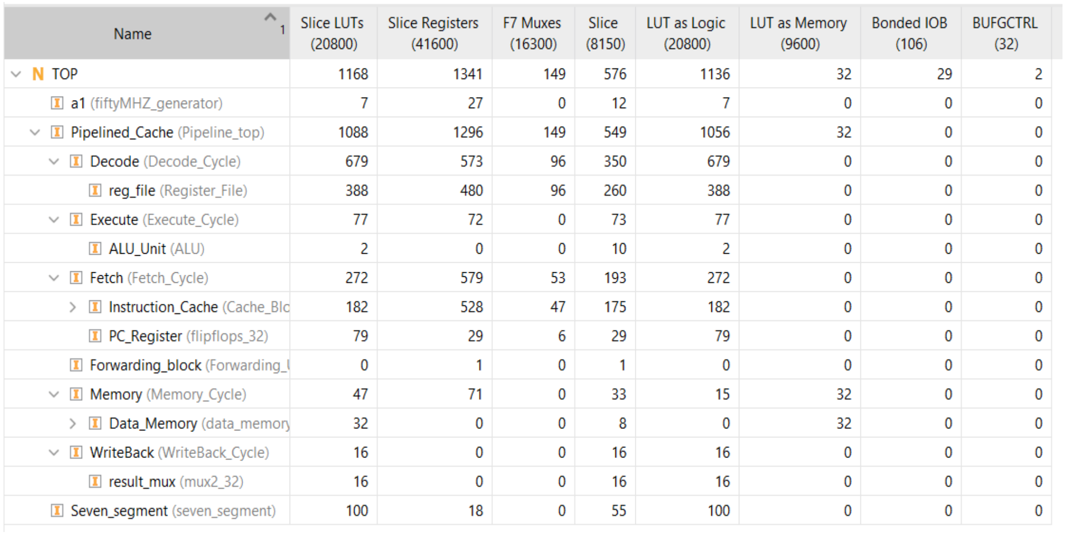


Figure 3: Simulation for Pipelined Processor

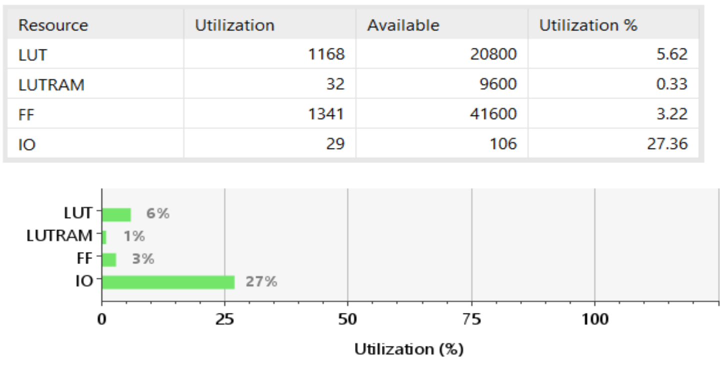
VIII RESOURCE UTILIZATION

The component statistics of the Pipelined Processor designed using Verilog Code gets synthesized into the Hardware as shown below in Fig.4 and Fig.5



Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Bonded IOB (106)	BUFGCTRL (32)
TOP	1168	1341	149	576	1136	32	29	2
a1 (fiftyMHZ_generator)	7	27	0	12	7	0	0	0
Pipelined_Cache (Pipeline_top)	1088	1296	149	549	1056	32	0	0
Decode (Decode_Cycle)	679	573	96	350	679	0	0	0
reg_file (Register_File)	388	480	96	260	388	0	0	0
Execute (Execute_Cycle)	77	72	0	73	77	0	0	0
ALU_Unit (ALU)	2	0	0	10	2	0	0	0
Fetch (Fetch_Cycle)	272	579	53	193	272	0	0	0
Instruction_Cache (Cache_Blc	182	528	47	175	182	0	0	0
PC_Register (flipflops_32)	79	29	6	29	79	0	0	0
Forwarding_block (Forwarding_	0	1	0	1	0	0	0	0
Memory (Memory_Cycle)	47	71	0	33	15	32	0	0
Data_Memory (data_memory	32	0	0	8	0	32	0	0
WriteBack (WriteBack_Cycle)	16	0	0	16	16	0	0	0
result_mux (mux2_32)	16	0	0	16	16	0	0	0
Seven_segment (seven_segment)	100	18	0	55	100	0	0	0

Figure 4: Components statistics of Pipelined Processor

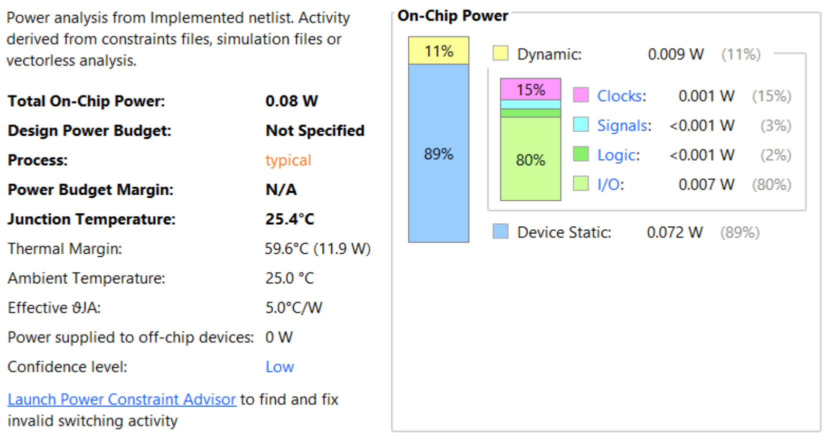


Resource	Utilization	Available	Utilization %
LUT	1168	20800	5.62
LUTRAM	32	9600	0.33
FF	1341	41600	3.22
IO	29	106	27.36

Figure 5: Components Summary of Pipelined Processor

IX Power Report

The Power Report of the Pipelined Processor is shown in Fig.6



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.08 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (11.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	0.009 W	(11%)	
Clocks:	0.001 W	(15%)	
Signals:	<0.001 W	(3%)	
Logic:	<0.001 W	(2%)	
I/O:	0.007 W	(80%)	
Device Static:	0.072 W	(89%)	

Figure 6: Power Report of Pipelined Processor

X Conclusion

The designed 32-bit RISC Pipelined CPU was then verified using the assembly code for finding the maximum value in a set of 10 random Numbers and output was displayed on Basys3 FPGA Board using its 7-Segment display. Algorithm to find maximum number was written in C and got the required assembly/binary code using Compiler Explorer and then used RVCodec to get the Hex Representation of the RISC-V Instructions.

References

[1] Digital Design and Computer Architecture 2nd edition" - David Mooney Harris and Sarah L Harris

[2] "Compiler Explorer to Convert C-Code to RISC-V 32 bit Instructions"

[3] "RVCodec to get the Hex representation of RISC-V Instructions"