5 Stage Pipelined RISC-V Processor in RTL

P V Y N Sai Teja,
Mtech VLSI &Embedded systems,
IIIT Delhi,
venkata21221@iiitd.ac.in

Akanksha Tripathi,
Mtech VLSI &Embedded systems,
IIIT Delhi,
akanksha21184@iiitd.ac.in

Sudhanshu Trivedi,
Mtech VLSI &Embedded systems,
IIIT Delhi,
sudhanshu21212@iiitd.ac.in

***Abstract*—This report contains a detailed implementation of a 32-bit processor based on a subset of instructions from RISC-V ISA. The Micro-architecture contains 5 pipeline stages namely Fetch, Decode, Execute, Memory and Write back. Additionally, it also has a MAC execution unit to handle MAC instructions which are not part of RISC-V ISA. The processor can handle data and structural hazards with the use of forwarding and stalling logic whenever required. The processor is implemented using Verilog HDL and the operating frequency has found to be 100 MHz**

Keywords— RISC-V, RV32I, Micro-architecture, Pipeline, RTL.

I. Introduction

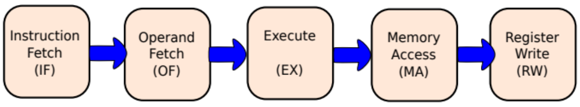
RISC-V is an instruction set architecture (ISA) that was originally designed to support computer architecture research and education which now has become a standard free and open architecture for industry implementations. RISC-V has been designed to support for 32-bit, 64-bit and 128-bit address spaces. The ISA is separated into a small base integer ISA, which is a minimal set of instructions adequate to provide customized accelerators for educational purposes, optional standard extensions and reasonable target for assemblers, compilers and operating systems. The base integer ISA can be extended with one or more optional instruction-set extensions, but the base integer instructions cannot be redefined.[1]

RISC based architectures have been used in both low level applications and mobile systems by the beginning of the 21st century. The low power and low cost embedded market is dominated by the RISC based ARM architectures. Most of the android based devices, Apple iPhone an iPad and most hand-held devices uses the ARM architecture. The MIPS line can currently be found in games like PlayStation Portable game consoles, Nintendo 64 and personal residential gateways like Linksys WRT54G series. Modern mobile phones, different gaming and low level application systems are using RISC based architecture. [3]

A. *Why Pipelining?*

In single cycles processors (unpipelined) We need to wait for all individual sub-operations to complete, before we can begin processing the next instruction, we need to wait for all of these individual sub-operations to complete, before we can begin processing the next instruction. Suppose when opcode fetch unit is working, rest of the units have to be in idle state. If we assume that each of the five stages (IF,ID,EX,MA,WB) takes the same amount of time, then at any instant, about 80% of our circuit is idle! This represents a waste in computational power, and idling resources is definitely not a good idea.[2]

B. *Advantages of a pipelined processor*



In a pipelined processor, we have divided the data path into five stages, where each stage processes a separate instruction. In the next cycle, each instruction passes on to the next stage and so on. By doing this, we can achieve a CPI which is close to one, which is far better than an unpipelined processor. It is not possible to arbitrarily increase the performance of a processor by increasing the number of pipeline stages. In fact, after a certain point, adding more stages is counterproductive. However, there are certain limitations for pipelined processors also, and once such limitation is Hazard. Two important types of Hazards are Data and structural hazards. Data hazard occurs due to data dependencies and structural hazards occurs due to hardware dependencies. To handle these hazards, one must schedule their instructions properly to avoid dependencies and also by implementing proper stalling and bypassing techniques, these hazards can be resolved.[2]

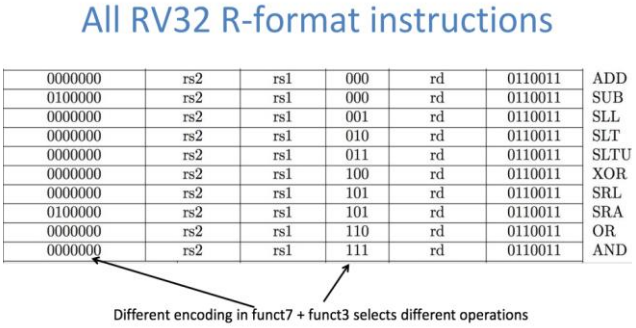
This work presents a hardware design architecture to realize a subset of RV32I base integer instruction set for 32-bit address space. It is a pipelined processor with 5 pipeline stages namely Fetch, Decode, Execute, Memory and Write back. The processor also contains a MAC (Multiply and Accumulate) execution unit which in not part of RV32I ISA. The processor has a separate Instruction memory from where it can fetch the instructions which are stored in form of a binary and it also has a data memory where the data is stored. It also includes stalling and forwarding units which deals with data and structural hazards to ensure timely functioning of the processor.[2]

II. RISC-V (RV32I) Instruction set

RISC-V has 32 registers in register file indexed from 0 to 31, out of which 31 are general purpose registers (indexed 1 to 31) and register indexed 0 is hardwired to constant 0. The program counter is another user visible register in RISC-V. There are four core instruction formats in the RV32I base integer instruction set: R, I, S and B.[5]

A. *R-Format*

These are register based instructions in which all three operands are registers, and are identified by the 7-bit opcode of the instruction. Some of the R format instructions include ADD, SUB, OR, XOR, AND, SRA, etc.



All RV32 R-format instructions

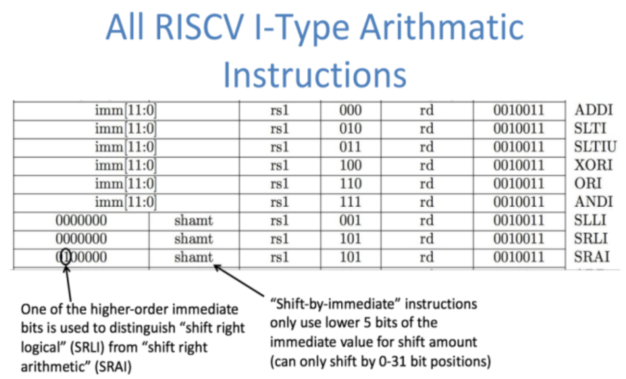
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Different encoding in funct7 + funct3 selects different operations

The different functionalities of R format instructions is identified by the function fields funct7 and funct3 as shown in above figure.[4]

B. I-Format

These are immediate type instructions, where one of the operands is an immediate bit which is directly decoded and computed from the instruction.



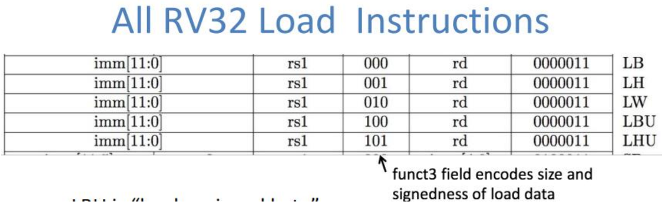
All RISCV I-Type Arithmatic Instructions

imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI

One of the higher-order immediate bits is used to distinguish "shift right logical" (SRLI) from "shift right arithmetic" (SRAI)

"Shift-by-immediate" instructions only use lower 5 bits of the immediate value for shift amount (can only shift by 0-31 bit positions)

The encoding of various I format instructions is as shown above. Load instructions are also comes under I format instructions whose encoding style is shown belwo.[4]



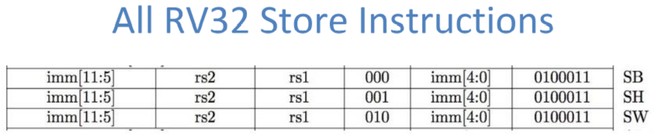
All RV32 Load Instructions

imm[11:0]	rs1	000	rd	0000011	LB
imm[11:0]	rs1	001	rd	0000011	LH
imm[11:0]	rs1	010	rd	0000011	LW
imm[11:0]	rs1	100	rd	0000011	LBU
imm[11:0]	rs1	101	rd	0000011	LHU

funct3 field encodes size and signedness of load data

C. S-Format

These are Store instructions. Store needs to read two registers, rs1 for base memory address, and rs2 for data to be stored, as well as need immediate offset. It cannot have both rs2 and immediate in same place as other instructions![4]

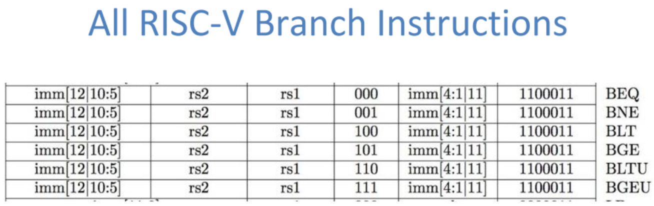


All RV32 Store Instructions

imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

D. B-Format

B-format is mostly same as S-Format, with two register sources (rs1/rs2) and a 12-bit immediate. But now immediate represents values -2^12 to +2^12-2 in 2-byte increments. The 12 immediate bits encode even 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it) [4]

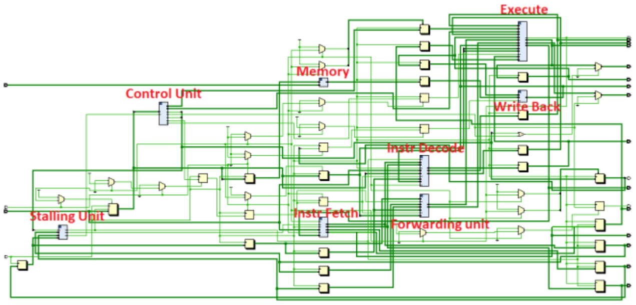


All RISC-V Branch Instructions

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

III. Architectural Implementation

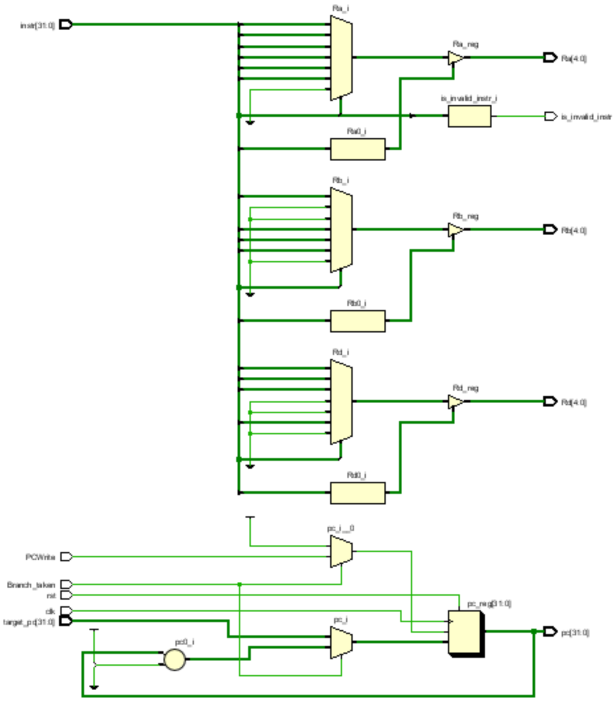
A. Top Level Overview



The above schematic gives an overall hardware implementation of the processor with all the units. It also has a separate Instruction memory, Data memory and a MAC execution unit. The hardware implementation of those units will be shown in the further discussion.

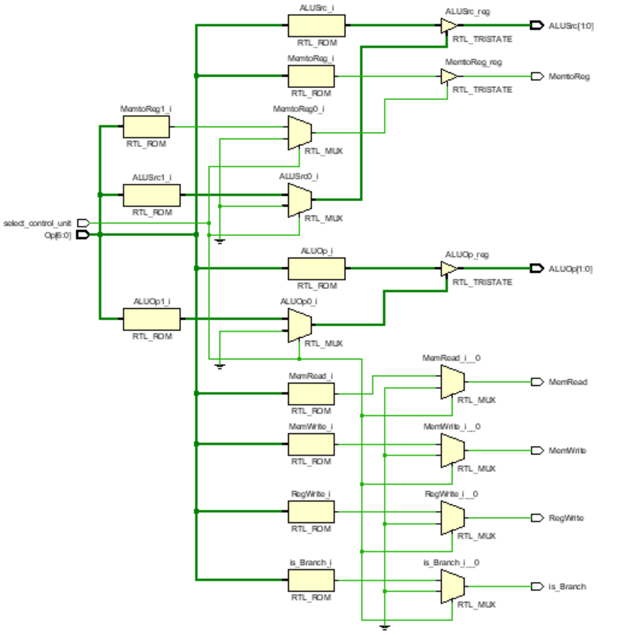
B. Instruction Fetch stage :

Fetches the instruction from the instruction binary specified in the instruction memory. PC increments by 4 every time to point out to the next instruction speculatively assuming no Interrupts or exceptions occur. It has an early decode unit which decodes the indices of the register operands and provide them to the decode stage to tells what values are to be decoded.



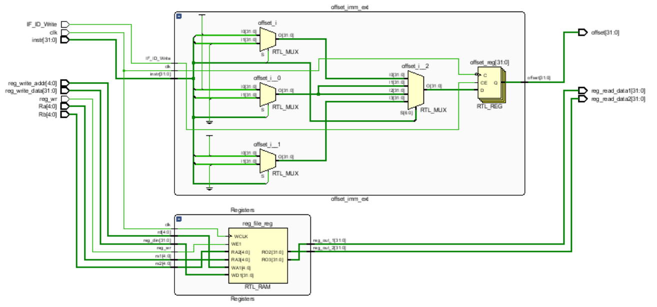
C. Control Unit :

The control unit generates all types of control signals that are needed by various stages at some instance. Basically, these control signals control the data flow in the pipeline.



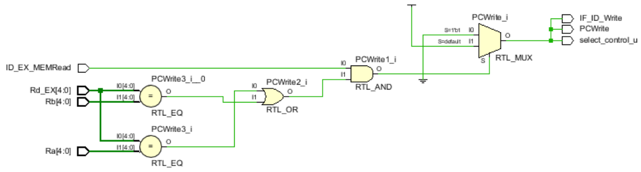
D. Instruction Decode Stage

It decodes the values of registers and sends it to the execution unit to perform various operations on the operands depending on the instruction. This stage has two sub modules namely Register file and Offset unit. Register file contains all the contents of 32 registers that are being used in the processor. Offset has to be calculated whenever there are I, S or B type instructions, and for this purpose we have an offset unit inside the decode stage.



E. Stalling Logic

Stalling is required whenever there is a structural Hazard or hardware dependency. Stalling logic will stall the instruction at the respective stage when the next stage is busy handling the previous instruction. The stalling logic here is implemented in such a way that if any of the source registers are being used by previous instruction, then it stalls the instruction until the register is available.

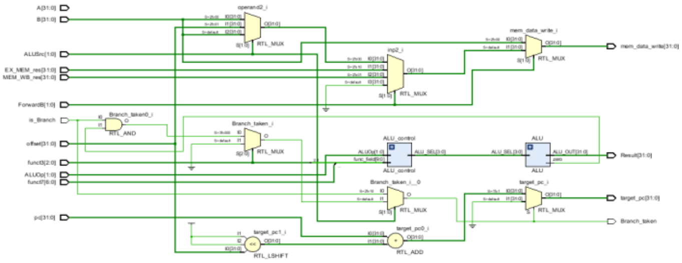


F. Execute Stage

This unit comprises of two sub modules –ALU and ALU Control. In the Execute Module, the ALU and ALU control unit are instantiated. The logic for selection of forwarding unit is also given in the Execute unit. The logic for branch instruction is also present in the execute unit which states whether the branch is being or not being taken and if branch is taken then the target instruction calculation is also given in the Execute Module.

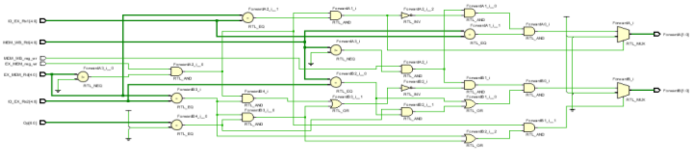
ALU Control module is for selecting a particular operation out of the other operations present in ALU module. ALU module in this processor handles only R-type, I-type instruction and branch instruction.

ALU module consists of the logic for operations to be performed by the processor and case function is used so that a number is allotted to each operation so by using that number a particular operation in the ALU Module is selected by the ALU Control Module.



G. Forwarding unit

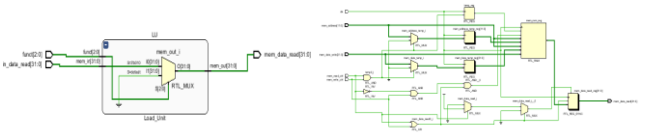
A separate forwarding unit holds the bypassing logic in case occurrence of data hazards. It forwards the results of ALU operations back to execute stage to resolve data dependencies and avoid stalling.



H. Memory stage

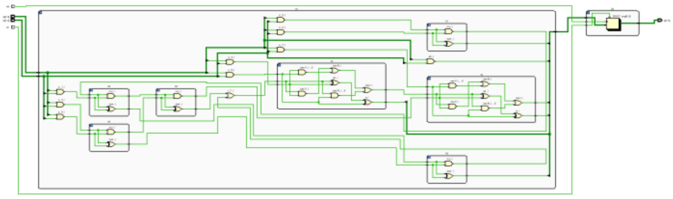
Memory stage is responsible for performing all types of memory operations such as Loads and stores. It takes the offset address from Execute stage and operand values in order to perform load and store on those memory locations. It has a sub module called load unit which is used for load instructions.

A separate unit called Data memory unit is there in order to access data from the memory locations.



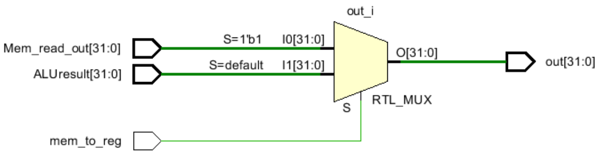
I. MAC Execution Unit

We have included a MAC execution unit which is used to execute MAC instructions (an additional feature to this processor not included in RISC-V). A MAC operation is in general Multiply and Accumulate. For this we have used a method of adding partial products using various half adders and full adders in the implementation.



The above schematic shows the hardware implementation of MAC execution unit.

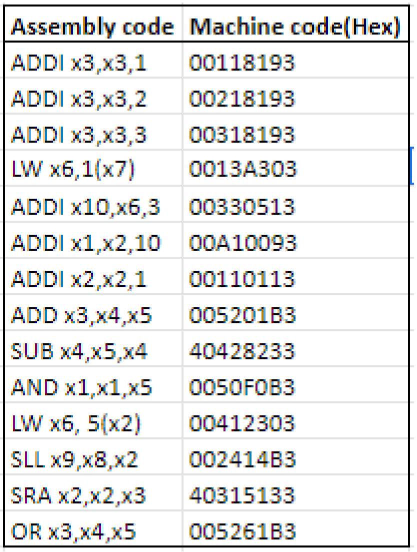
J. Write Back stage



The last stage is Write back stage which writes back the updated register values from the ALU result into respective registers.

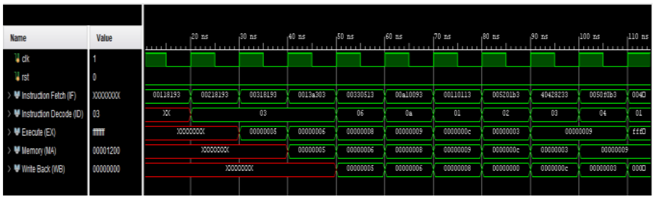
IV. Simulation and Results

The processor has been implemented using Verilog Hardware description Language and simulations have been done on Xilinx Vivado 2019.1. Below is the Instruction binary that has been used for the simulations.



Assembly code	Machine code(Hex)
ADDI x3,x3,1	00118193
ADDI x3,x3,2	00218193
ADDI x3,x3,3	00318193
LW x6,1(x7)	0013A303
ADDI x10,x6,3	00330513
ADDI x1,x2,10	00A10093
ADDI x2,x2,1	00110113
ADD x3,x4,x5	005201B3
SUB x4,x5,x4	40428233
AND x1,x1,x5	0050F0B3
LW x6, 5(x2)	00412303
SLL x9,x8,x2	002414B3
SRA x2,x2,x3	40315133
OR x3,x4,x5	005261B3

The pipelined execution of all the instructions will result in the following simulation waveforms.



V. Conclusion

We have designed:

- A 5-stage pipelined architecture.
- RISC-V 32-bit processor that is able to perform a subset of available instructions that are present in the RISC V ISA.
- Simulations have been done on Vivado and the operation of processor has been verified to be correct.

References

[1] A. Waterman, Y. Lee, D. A. Patterson, K. Asanovic, V. I. U. level Isa, A. Waterman, Y. Lee, and D. Patterson, "The risc-v instruction set manual," 2014.

[2] Basic Computer Architecture Version 2.1 by Smruti R. Sarangi.

[3] S. P. Dandamudi, *Guide to RISC Processors: For Programmers and Engineers*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[4] Great Idea in Computer Architecture, RISC-V Instruction Formats by StevenHo https://inst.eecs.berkeley.edu/~cs61c/resources/su18_lec/Lecture7.pdf

[5] E. Farquhar and P. Bunce, *The Mips Programmer's Handbook*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.