EN3551 Digital System Design

RISC-V Processor Design

Group:	Computer Squad
Module:	EN3551
Index Numbers:	210236P, 210253N, 210272V
Date:	2024.07.12

1 Introduction

RISC-V is an open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles. It was initially developed at the University of California, Berkeley. Unlike other ISAs, RISC-V is provided under royalty-free open-source licenses, which has led to its easier acceptance by a wider community. In this paper, we design a custom RISC-V processor supporting a subset of the instructions of the ISA.
In this report, we discuss the individual instruction types, the formats and parallelly develop the datapath and control path.

2 Specifications

The following are the required specifications of the proposed design

1. The processor should be a 32 bit processor
2. The processor should support all R type instructions.
3. The processor should support all I type instructions.
4. The processor should support all S type instructions.
5. The processor shall be made as simple as possible

3 Components in the Design

The following components are part of the design. In this section, the functionalities of each of the components is explained.

3.1 Control Unit

The control unit will serve as the main controller of the processor. It receives the instruction and based on the instruction controls all other components such as the register file, ALU, multiplexers and data memory. All control paths originate from the control unit.

3.2 Register File

The register file is where all immediately accessibel data is stored. RISC-V is a load-store architecture. Therefore, any operands would need to be stored in the register file first. The register file contains 32 registers, each 32 bits wide. The details of the registers are shown in table 1.

The purposes of the registers are explained as follows:

- x0 (zero): Always zero. This register is hardwired to the value zero
- x1 (return address): Holds the return address for function calls.
- x2 (stack pointer): Points to the top of the stack.
- x3 (global pointer): Points to the location where global variables are typically stored.
- x4 (thread pointer): Points to the thread-local storage area, used in multithreading environments.

- x5 to x7 (temporary registers): Used for temporary values within functions. These registers do not need to be preserved across function calls.
- x8 (saved register/frame pointer): Used as a saved register across function calls or as a frame pointer to reference function parameters and local variables.
- x9 (saved register): Another saved register, used across function calls.
- x10 to x11 (function arguments/return values): Used to pass arguments to functions and return values from functions.
- x12 to x17 (a2 to a7 - function arguments): Used to pass additional arguments to functions.
- x18 to x27 (saved registers): Saved registers that must be preserved across function calls.
- x28 to x31 (temporary registers): More temporary registers. These do not need to be preserved across function calls.

Register	**Description**
x0	Constant 0
x1	Return address
x2	Stack pointer
x3	Global pointer
x4	Thread pointer
x5-x7	Temporary
x8	Saved register & frame pointer
x9	Saved register
x10-x11	Function argument & return values
x12-x17	Function argument
x18-x27	Saved register
x28-x31	Temporary

Table 1: Registers and their functions

There are 4 inputs to the register file. They are rs1 address, rs2 address, rd address and data in. The addresses are 5 bits wide and the register file outputs the contents of whatever is stored in registers rs1 and rs2 2 output ports. rd is provides the 5 bit address to the destination register where data in the data input will be written if the controller enables a register write.

3.3 ALU

The Arithmetic and Logic unit is responsible for all the computations in the processor. It has 2 inputs where 2 32 bit operands are given and 1 32 bit output. The function it needs to perform is controlled by the control unit. For the proposed design, 4 bits are sufficient to determine the correct operation.

3.4 PC adder

This is a much simpler ALU which adds 4 to the program counter. It only performs addition and one input is permanently set to 4

3.5 Branch Adder

This is also another adder but it both input can be set. One input is the current PC value and the other is the byte offset to jump to. It is used to calculate new addresses to jump to.

3.6 Immediate Generator

This is a component that generates the immediate in a format suitable for the rest of the components as stated in the RISC-V ISA. Immediates are either immediates for string in registers upon an ALU operation or for the branch offset calculation

3.7 Instruction Memory

The instruction memory stores all the instruction that the processor needs t execute. It has 2 ports. A 32 bit address input port and a 32 bit data output port which outputs the instruction stored at the address which is input.

3.8 Data Memory

The data memory is used to store data. Data can come after calculation by th ALU. It also has an address port, data input port, data output port and a read/write port. The read/write port determines what operation needs to be performed at the address given at the address port. The read/write port is controlled by the control unit.

3.9 Multiplexers

When 2 signals need to be sent to the same location, but at different times, multiplexers can be used to select which signal can pass through. These are controlled by the control unit.

4 R type instructions

The R type instructions have the following structure: Going through the RISC-V documenta-

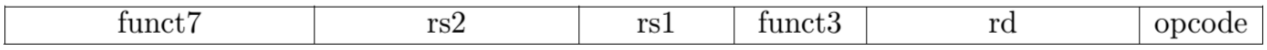

Figure 1: R Type instruction format

tion, the opcode is the same for all R-type instructions. Therefore, it is possible to differentiate if an instruction is R-tpye based solely on the opcode.
These instructions consist of operations between 2 registers. rs1 and rs2 indicated above are the source registers and rd is the destination register where the result is stored.
In order to facilitate these instructions, the following components are necessary:

- Register file: The collection of registers
- ALU: To perform the calculations
- Instruction Memory: To provide instructions

Additionally, a control unit is also required for controlling the register file and ALU.

Inst	Name	FMT	Opcode	funct3	funct7
add	ADD	R	0110011	0x0	0x00
sub	SUB	R	0110011	0x0	0x20
xor	XOR	R	0110011	0x4	0x00
or	OR	R	0110011	0x6	0x00
and	AND	R	0110011	0x7	0x00
sll	Shift Left Logical	R	0110011	0x1	0x00
srl	Shift Right Logical	R	0110011	0x5	0x00
sra	Shift Right Arith*	R	0110011	0x5	0x20
slt	Set Less Than	R	0110011	0x2	0x00
sltu	Set Less Than (U)	R	0110011	0x3	0x00

Figure 2: R Type instructions

The datapath and control path for R type instructions can be designed as shown in figure 3. The program counter outputs the address of the next instruction to be executed. The instruction memory takes in an instruction address as an input and outputs the instruction stores at the relevant address. All paths are 32 bits. From that point onwards, the paths are separated, one going to the rs1 address port, one to the rs2 address port, one to the rd address port and the rest to the control unit. Since only R- type instructions are available and since all R-type instructions have the same opcode, the opcode can be ignored when sending to the control unit for this stage. This however will be added in the next stages.

Since there are 10 operations in the ALU, 4 bits are sufficient to control the ALU. The control signals can be designed as in table 2

Control Signal	**Operation**
0000	ADD
0001	SUB
0010	XOR
0011	OR
0100	AND
0101	SLL
0110	SRL
0111	SRA
1000	SLT
1001	SLTU

Table 2: ALU control signals

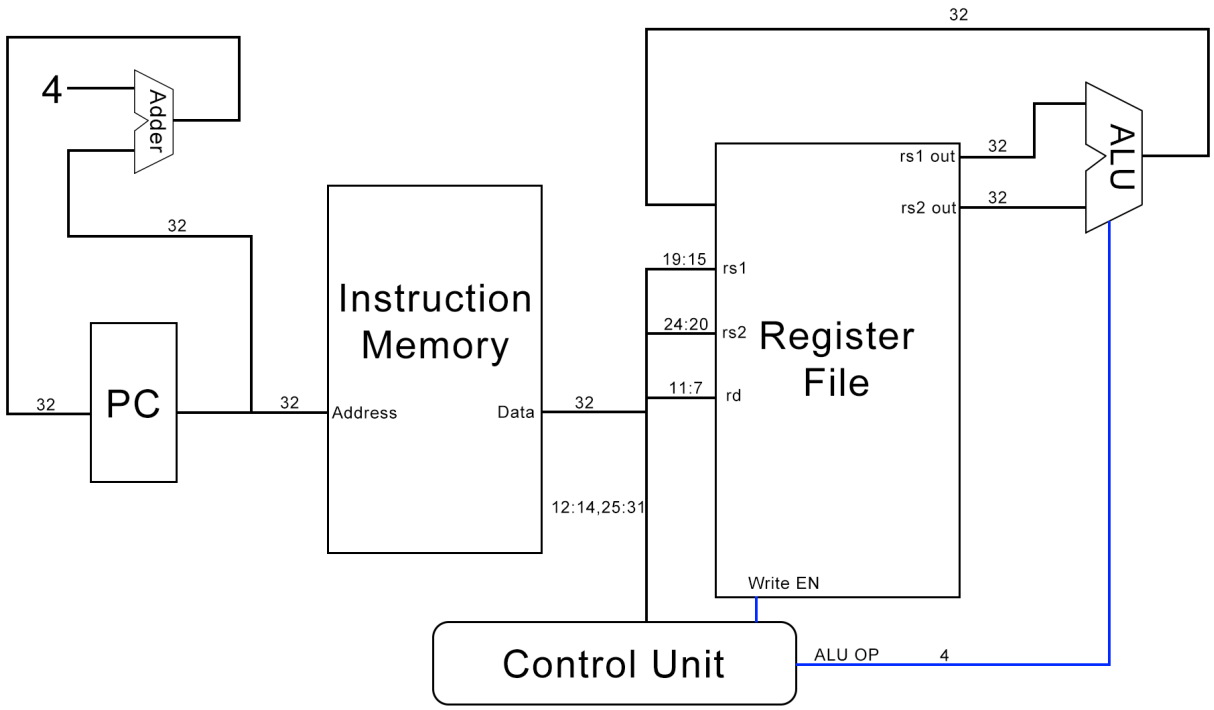

Figure 3: R Type instruction data and control paths

5 I type instructions

These instructions are instructions with immediates. Immediates can be either operands for calculation or offests for addresses. I type instructions have the following structure: I-type

imm[11:0]	rs1	funct3	rd	opcode

Figure 4: I Type instruction format

instructions take in 1 source register and a 12 bit immediate and perform a calculation on these and stores the output in the rd register. In addition to the components required for R-type instructions, I-type instructions would also require an immediate generator. This is because, all calculations need to be performed in 32 bit integers and since the immediate is 12 bits wide, it is necessary to sign extend the immediate to 32 bits. Comparing R and I type instruction formats, it is seen that the fields: opcode, rd, func3 and rs1 are aligned in both instruction types. This is to simplify the microarchitecture design. All the I-type instructions are given in figure 5

As seen in figure 5 and figure 3, the only difference in the opcodes is the 5^{th} bit. Therefore, it is possible to only send this bit additionally to the control unit to differentiate between R and I instructions. The rest of the bits on the I-type opcodes are the same.

The proposed datapath for R and I type instructions is shown in figure 7. In addition to the components in the R-type datapath, a unit has been added for sign extension of the immediate. This is because when going to the ALU, all operands need to be 32 bits wide. It is a sign extended version of the immediate. That is, if the leading bit of the immediate is 1, it is extended with ones and if it is 0, it is extended with zeros. This is beacuse sign integers

addi	ADD Immediate	I	0010011	0x0
xori	XOR Immediate	I	0010011	0x4
ori	OR Immediate	I	0010011	0x6
andi	AND Immediate	I	0010011	0x7
slli	Shift Left Logical Imm	I	0010011	0x1
srli	Shift Right Logical Imm	I	0010011	0x5
srai	Shift Right Arith Imm	I	0010011	0x5
slti	Set Less Than Imm	I	0010011	0x2
sltiu	Set Less Than Imm (U)	I	0010011	0x3

Figure 5: All I-type instructions

are signed integers and sign needs to be preserved. Another addition is the multiplexer. The multiplexer selects between the rs2 value and the immediate to be sent to the ALU. The multiplexer is controlled by the control unit.

The ALU controls are the same as that of the R-type instructions. One thing to note is the lack of a subtract immediate instruction. This is because any value that needs to be subtracted can be sent as a two's complement as the immediate and addition can be performed.

Load instructions are also I-type instructions and hence, the processor would need ts support these too. Load instructions are also considered I-type because the address from which to load data from is calculated as an offset from a base register. The instruction of these types are given in figure 6. Now, the control unit would need bit 4 as well to determine the instruction.

lb	Load Byte	I	0000011	0x0
lh	Load Half	I	0000011	0x1
lw	Load Word	I	0000011	0x2
lbu	Load Byte (U)	I	0000011	0x4
lhu	Load Half (U)	I	0000011	0x5

Figure 6: I-type load instructions

There are different types of load instructions all of which load different widths of data to registers and whether they are signed or unsigned. For these instructions, rs1 needs to have the base address of the memory from whixh the offset given by the immediate is calculated. rd is the register where the loaded data is stored. The datapath and control path would need to be modified as shown in figure 8 to support loads.

The main additional component is the data memory. The ALU calculates the address as a sum between rs1 and the immediate. This address is passed to the data memory which then puts out the data on its data output port. Additional control signals are to control the multiplexer between ALU output and data memory output. This is to decide whether to send the ALU output or the data memory output to the rd register. Another introduced control signal is the wr/rd (write/read) for the data memory. This decides whether the data memory should write or read data. In this section only reads are considered and writes are considered in a latter section. It is possible to set this control signal to read for all other instructions as the output is blocked by the multiplexer.

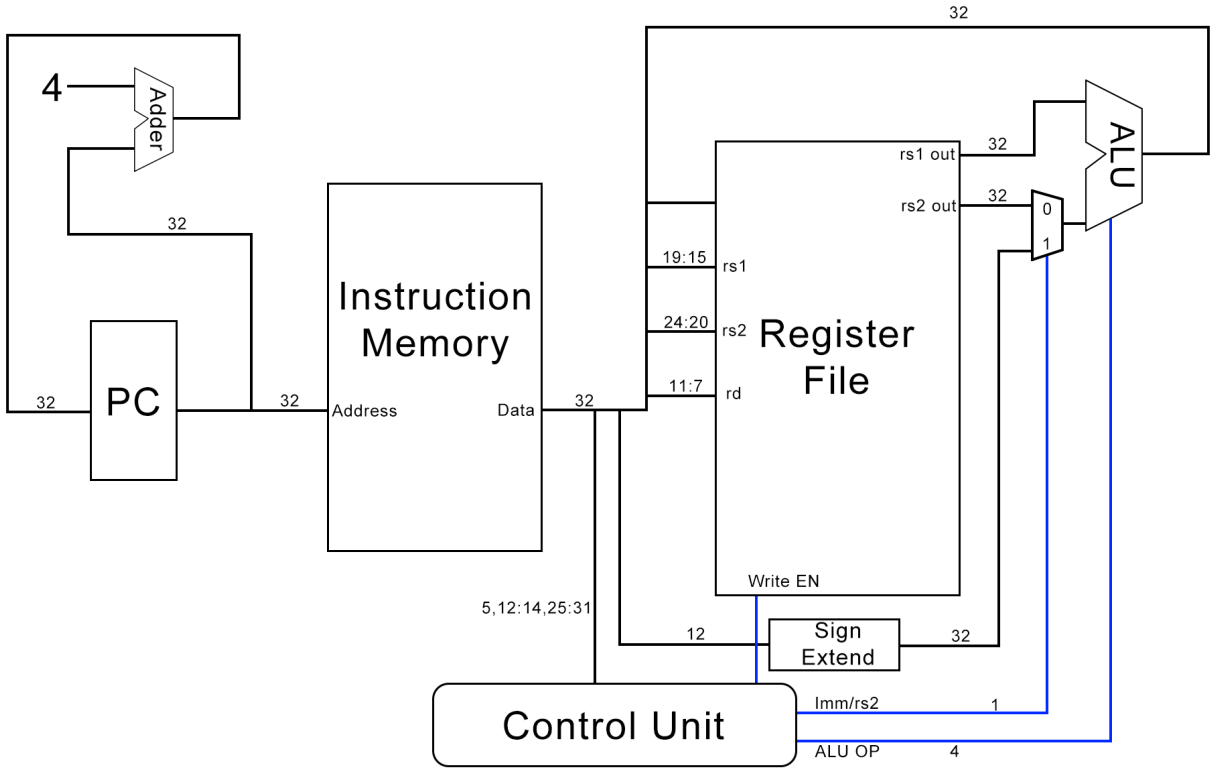

Figure 7: I-type data and control path

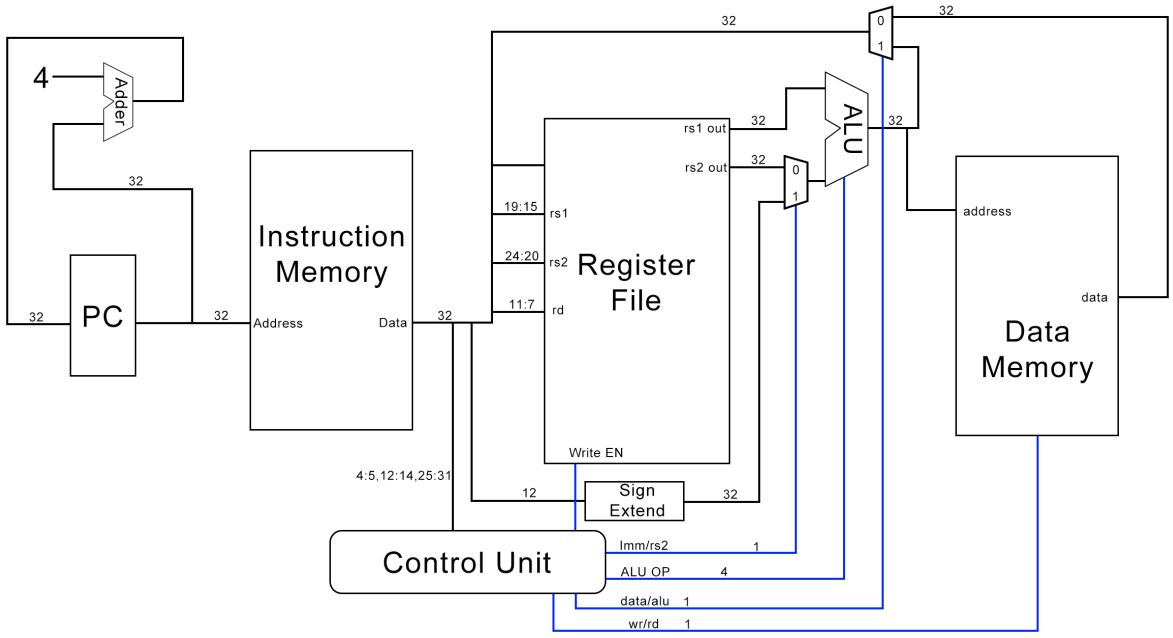

Figure 8: I-type data and control path with loads

6 S type instructions

S-type instructions are store instructions. They are used for storing data to the data memory. The following is the structure of S-type instructions. As seen in the figure, S-type instructions

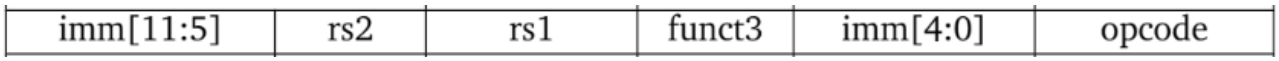

Figure 9: S Type instruction format

take in 2 registers rs1 and rs2, a 12 bit immediate and a function 3 bits wide. The functionalities of these are given below:

- rs1: rs1 is the base address register. This register stores the base address of from which an offset is calculated to store the data.
- immediate: The immediate gives the offset from the rs1 value at which the data needs to be stored. That is, the byte address where data is stored is given by: $address = rs1 + immediate$
- rs2: This register holds the data that needs to be stored. Since RISC-V is a load/ store architecture, the data that needs to be stored must first be in s register. Immediates cannot be directly sent to data memory.
- function3: These 3 bits indicate the width of what needs to be stored. Specifically whether to store a byte, half word or word.

sb	Store Byte	S	0100011	0x0
sh	Store Half	S	0100011	0x1
sw	Store Word	S	0100011	0x2

Figure 10: All I-type instructions

The S-type instructions are shown in figure 10.

Comparing all 3 instruction formats discussed so far, it is noted that opcode, functtion3, rs1 and rs2 are aligned in order to simplify the microarchitecture design. In order to stick with this approach, in S-type instructions, the immediate is broken down into 2 parts. The lower 5 bits are encoded together while the remaining 7 bits are encoded together.

In addition to the existing control and data paths, now the data memory would need a data input port as well. The data and control paths are shown in figure 11.

The architecture is similar to before as no new components were added. However, now the ALU output feeds into the address port of the data memory as well as address needs to be calculated by the ALU. Additionally, a path has been added from rs2 output to the data in the port of the data memory in order to get the data that needs to be written to the memory.

It is still sufficient to just use bits 4 and 5 to determine the type of instruction and hence of the 7 opcode bits only those 2 are sent onto the control unit. The sign extend unit now, in addition to the 12 bits sent due to I-type instructions also needs the 5 bits in positions 7:11. Therefore, these bits are also sent to the sign extend unit for a total of 17 bits. Since immediate are of 2 different forms for S and I type instructions, now the control unit needs to

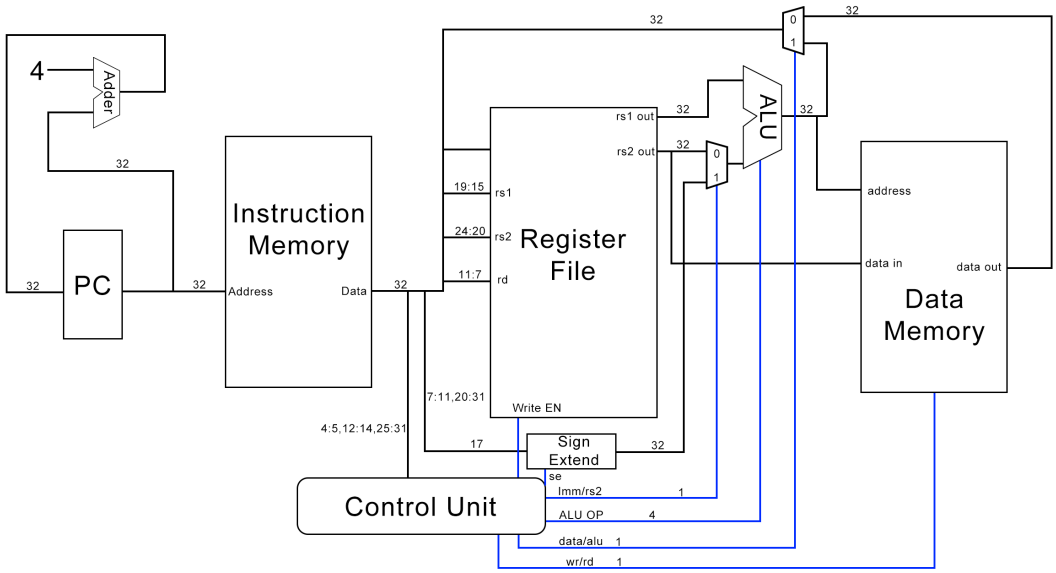

Figure 11: R, I and S type data and control paths

give a control signal to the sign extend unit to tell which type of immediate to generate. This was not necessary for only I-type instructions as the unit just had to generate the same type of immediate.

The write/read enable signal to the data memory makes sure that data is written to the address at the address port and no reading is done. Any possible writing to the register file is prevented by the write enable signal to the register file. The ALU needs only to add in the case of S-type instructions.

7 U type instructions

U type instructions are used for storing large numbers. The load instructions in the I-type instructions can only load 12 bit immediates, but the registers are 32 bits wide. Therefore, these instructions are designed to load the upper 20 bits as immediates. The following is the structure of S-type instructions. rd is the register where the immediate needs to be loaded. The

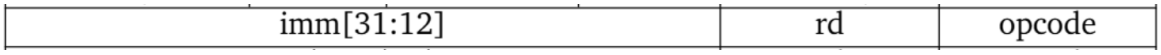

Figure 12: U type instruction format

immediate should be the upper 20 bits. The instructions in this category are shown in figure 13.

lui stands for load upper immediate. It stores the 20 immediate bits in register rd and

lui	Load Upper Imm	U	0110111
auipc	Add Upper Imm to PC	U	0010111

Figure 13: All U type instructions

clears any pre-existing data in rd. In order to store a 32 bit value in the register, it is possible to then use an addi I type instruction to add an immediate to the register value and store. auipc is used to add the upper immediate to the value of the program counter. This is used for PC relative addressing.

The data and control paths are shown in figure 14. The main addition is a multiplexer to

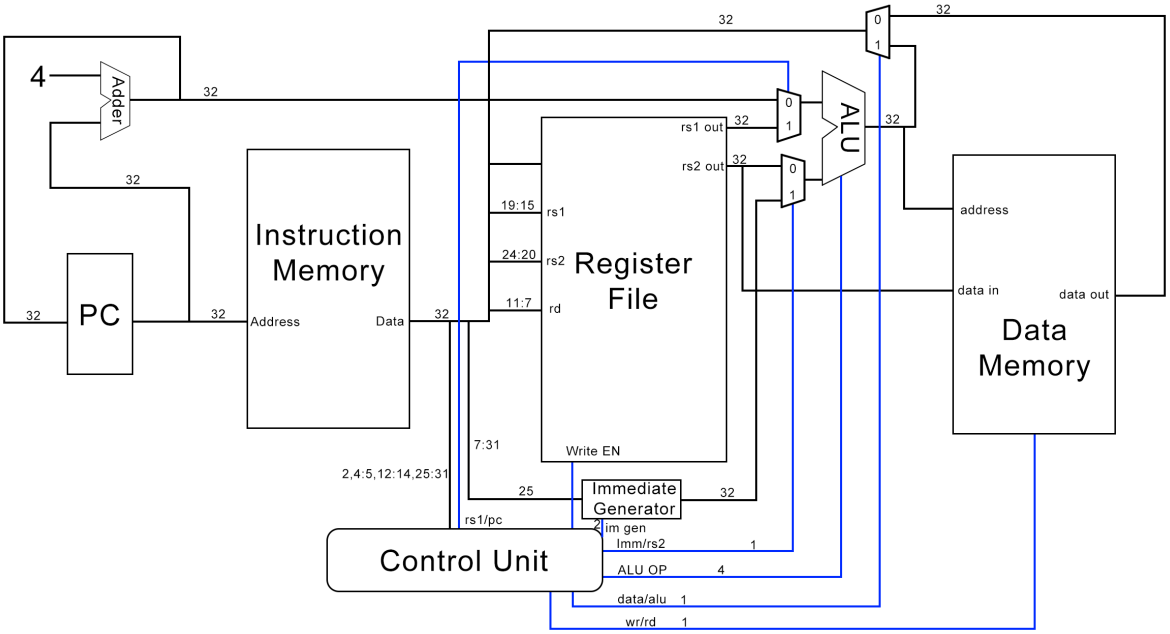

Figure 14: Data and control paths for R, I, S and U type instructions

select between rs1 and the program counter. When a U-type instruction is sent, the PC value is sent to the ALU. This is to support the AUIPC instruction. Additionally, now the control unit needs bit 2 as well to determine the type of instruction. Moreover, the immediate generation unit now needs the 20 bits as well thus making it's input 25 bits wide.

8 Conclusion

The above highlighted our paper design of a RISC-V processor supporting R,T,S and U type instructions of the RV32I architecture. It can be implemented using a suitable HDL to create a synthesizable processor on an FPGA. This implementation is left to be done.

9 References

1. J. Zhu,"RISC-V Reference Card," GitHub, [Online].
   Available: on GitHub
   Accessed: 12-Jul-2024.

2. UC Berkeley,"CS61C: Great Ideas in Computer Architecture (Machine Structures)," [Online].
   Available: at the course website.
   Accessed: 12-Jul-2024.