Dean&Francis
ISSN 2959-6157

Tinyriscv-based Processor Core Design

Jingbo Gao[1, *]

[1]School of Information Science and Engineering, Shandong University, Qingdao, China
*Corresponding author: 202200120157@mail.sdu.edu.cn

Abstract:
Microelectronics and integrated circuit technologies are undergoing continuous evolution, resulting in increasingly powerful processor chips. Presently, the predominant processors available in the market are based on the X86 and ARM architectures, both of which are proprietary and associated with substantial patent licensing constraints. In contrast, RISC-V is an entirely open-source instruction set architecture that is accessible to anyone. Given the diverse, personalized, and differentiated demands within the chip industry, the exploration and utilization of the RISC-V instruction set for processor design hold significant importance.This study first examines the RISC-V instruction set architecture, focusing on its instructional characteristics and format. Subsequently, a three-stage pipeline design is implemented based on the tinyriscv framework, incorporating resource optimizations. The study also addresses the pipeline conflict issues and proposes a partial solution. Finally, the pipeline design's efficacy and the processor core's fundamental instructions are validated through ModelSim simulations and the development of test programs.

Keywords: RISC-V; processor design; FPGA

1. Introduction

Integrated circuits are employed in a multitude of fields, including high-performance computing, consumer electronics, and wireless communications. The growing demand for data processing and the increasingly complex communications equipment functions on the chip's integration degree have led to more stringent requirements. Over the past decade, the rapid advancement of semiconductor processing technology has enabled a single chip to accommodate an increasing number of body tubes, depending on the required functionality. An integrated circuit chip can now contain anywhere between 10 and 100,000 body tubes, which has not only facilitated the continued evolution of integrated circuit design but has also prompted a surge in innovation within the broader system structure design domain [1].

The evolution of processor chips is inextricably linked to the advancement of the instruction set architecture. In the early days of computing, there were fewer computer instructions and computer programs were simpler, with fewer types of operations to perform. At that time, the prevailing trend was towards the development of Complex Instruction Set Computers (CISC) [2], which aimed to achieve the simplicity of software programming through the use of complex instruction operations and complex hardware implementations, as well as the reduction of registers. The implementation of a single CISC instruction that performs multiple operations did, however, result in an improvement in the efficiency of software programming at that time. The advancement of Moore's law has led to a surge in the demand for sophisticated chip functions, which has in turn increased the complexity of the design. The instruction set has also undergone a process of derivation, necessitating the performance of an ever-increasing number of operations. Consequently, the complexity of the instructions has risen. The shortcomings of the CISC are gradually revealed. The average number of clock cycles required by the instruction is considerable, and the CISC instruction is complex and irregular, making it challenging to insert into the pipeline. This results in a long clock cycle, which in turn makes the execution of the processor very difficult. Consequently, the execution time of the processor is considerable, and it is challenging to enhance its speed. Furthermore, due to the necessity of implementing numerous intricate operations, the hardware implementation is inherently complex, necessitating the allocation of additional hardware resources and chip area, along with an increase in manufacturing time and cost for the chip [3].

In response to the limitations of CISC, the Reduced Instruction Set Computer (RISC) was developed. Unlike CISC, RISC has a more streamlined set of instructions, simpler hardware implementations, a larger number of registers, and a greater number of lines of software program code. The pursuit of high performance in processor design is based on the acceleration of recurring events and the minimisation of the length of time taken to perform these events. The RISC design, following the insertion of the pipeline, can be executed every clock cycle of an instruction, with the pipeline partitioned to insert it into the critical path, resulting in a very short clock cycle. In accordance with the processor performance formula for comparing RISC and CISC, despite the greater number of instruction lines in RISC software, the CPI and clock cycle time are significantly smaller than CISC, resulting in a reduced actual processor execution time. These advantages have led to an increasing interest and acceptance of RISC, which has subsequently become the dominant approach in instruction set development.

Furthermore, processors and system-on-chips are confronted with an expanding array of novel challenges as technology advances and market demands evolve. By optimising the design of the processor core and introducing extended instruction sets, it is possible to meet the market requirements for higher performance, lower power consumption, smaller size, and a lower cost. Such optimisations can assist RISC-V processors in more effectively competing and responding to demand in disparate markets and applications. It is therefore of great academic and practical significance to conduct research on the mature RISC-V processor architecture, to optimise the processor core design and to investigate how to implement an extended instruction set under the RISC-V architecture in order to meet the acceleration needs of specific scenarios. To enhance the versatility and flexibility of RISC-V processors, align them with the evolving market demands for processors and system-on-chips, and facilitate chip autonomy and control, controllable has significant academic and practical implications [4].

This paper presents a study of the RISC-V instruction set architecture and a design of a three-stage pipeline processor core based on tinyriscv. The conflict problem inherent to the three-stage pipeline is analysed, and a targeted partial solution is proposed to divide the processor core modules and optimise the resources to some extent. The designed processor core was then validated through the use of Modelsim simulation and the manual creation of test programs based on the official test set developed by the RISC-V community.

2. RISC-V Instruction Set

2.1 RISC-V Instruction Set Features

The regularity of the RISC-V architecture is reflected in the coding style of the instruction set[5].The RISC-V instruction set is available in a number of standard formats, including 16-bit, 32-bit, 64-bit, and 128-bit. The 16-bit and 32-bit instruction sets are well-suited for embedded processors, whereas the 64-bit and 128-bit instruction sets are more appropriate for personal computers or servers. In contrast to traditional instruction set architectures, such as X86 and ARM, the RISC-V instruction set is characterised by an extensible and modular structure.

RISC-V employs a modular approach, wherein each instruction set module is represented by a letter of the alphabet. The basic integer instruction, represented by the letter "I," serves as the foundational instruction set for users to extend. The extended instructions include the multiply-divide instruction set (M) and the atomic instruction set. The single-precision floating-point instruction set (F), the double-precision floating-point instruction set (D), and the compression instruction set (C) are also available. Users may select different extended instruction sets to meet the specific requirements of their applications. To illustrate, in scenarios where a smaller area and lower power are required for embedded applications, the RV32IC instruction set may be selected. Conversely, in scenarios where high-performance is desired for application systems, the

ISSN 2959-6157

RV32IMFDC instruction set may be selected. It can be seen that the selection of more extended instruction modules will result in a more comprehensive processor, which will in turn lead to enhanced performance [6]. This paper implements a subset of the RV32I instruction set, comprising 32-bit instructions in four basic formats (R, I, S, U) and two variants (SB, UJ). The RV32I instruction coding format is illustrated in Figure 1.

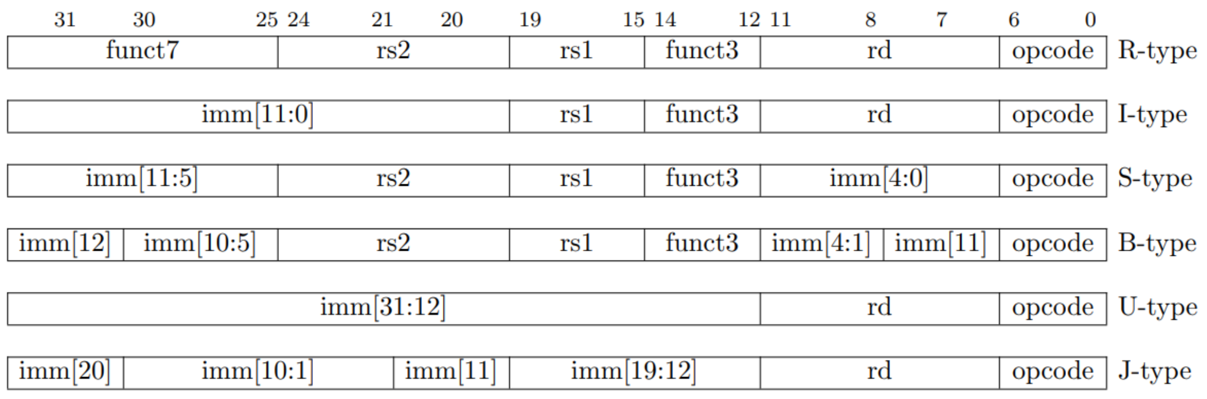


Fig.1 Basic Instruction Formats of RISC-V

The RISC-V ISA keeps the source (rs1 and rs2) and destination (rd) registers at the same position in all formats to simplify decoding. Immediates are always sign-extended, and are generally packed towards the leftmost available bits in the instruction and have been allocated to reduce hardware complexity. In particular, the sign bit for all immediates is always in bit 31 of the instruction to speed sign-extension circuitry [5].

3. RISC-V Processor

3.1 Architecture

3.1.1 Three-stage-pipeline

Pipeline is a technique that enables the overlapping execution of multiple instructions and constitutes a fundamental element of processor microarchitecture. The pipeline technique enhances system performance by augmenting the throughput of the system, thereby reducing the time required to complete an entire task. It does not, however, diminish the execution time of individual instructions [6]. The tinyriscv processor employs a straightforward three-stage pipeline architecture comprising fetch, decode, and execute modules. Figure 2 illustrates the three-stage pipeline designed in this paper, with reference to the tinyriscv.

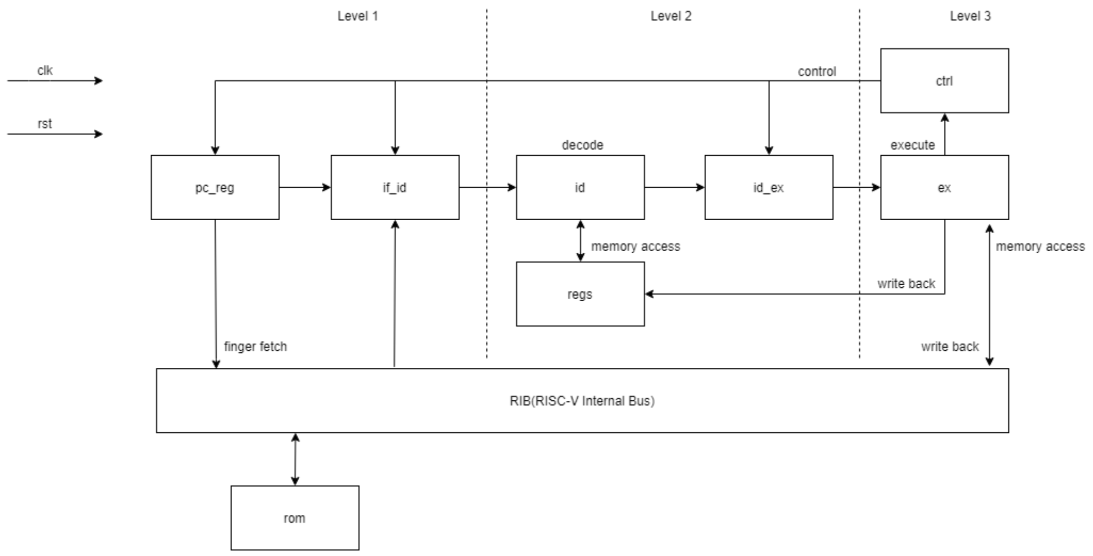


Fig. 2 Three-stage pipeline structure

Throughout the pipeline stage, the program counter is employed to generate the value of pc_reg, which is utilised as the address signal of the program memory. Given that tinyriscv is a three-stage pipeline structure, the CPU performs finger fetching and decoding. The value of pc_reg is therefore simultaneously coded, and executed. At this moment, the value of pc_reg represents the address of the instruction being executed. Consequently, the address of the instruction being finger fetched is pc_reg+8, and the address of the instruction being decoded is pc_reg+4. The address of the instruction to be decoded is pc_reg+4. In the event of an interrupt, the current value of pc_reg is stored directly. Upon the conclusion of the interrupt, if the program continues to return to the stored address of pc_reg, the interrupt will be re-executed, resulting in the program entering a dead loop. To circumvent this issue, the software implements a modification to the return address of the interrupt, whereby the current value of pc_rcg is augmented by four and the current address is bypassed. The mepc register is employed to store the return address of the interrupt, and upon the conclusion of the response, four is appended to the return address and transmitted back to the mepc register.

The tinyriscv does not possess a dedicated module for finger-fetching; rather, this operation is integrated into the decoding process. The output of the program counter (pc_reg) is connected to the input of the program memory (sim_ram). Given that the reading of sim_ram is a combinational logic operation, the instruction output from sim_ram is transferred to the input of the module situated between the finger fetch and decode stages, namely if_id, prior to the occurrence of each clock pulse. The module situated between the stages of fetching and decoding, designated if_id, is responsible for transmitting the instructions generated by sim_ram to the decoding module, designated id, following the occurrence of a clock pulse. Given that the computer is unable to discern which instruction is currently being fetched, it is essential for the module to be prepared for any instruction and to convey all potentially beneficial information throughout the pipeline.

The decoder module id is a combinational logic circuit that decodes the instructions transferred from the module id. Once decoding is complete, a signal is generated indicating whether the register is to be read or written. As the registers are read asynchronously, the corresponding register data can be obtained immediately upon sending the read register signal. This data is transmitted together with the write register signal to the execution module ex, situated between decoding and execution. The module then transmits the write register signal and register data to the execution module ex after a beat.

The execution module ex is also a combinational logic circuit. In accordance with the decoding information transferred from the ID module, the corresponding operation is executed. For instance, the add instruction executes the addition operation, the SUB instruction executes the subtraction operation, and if it is a memory access instruction, it reads and writes the memory in an asynchronous manner. Subsequently, the register write signal, the register address signal and the register data signal are conveyed to the general-purpose register group, designated as regs, while the memory write signal, the memory address signal and the memory data signal are transferred to the RIB bus, which assigns the module to be accessed.

The tinyriscv processor lacks a store-and-write-back stage; instead, the store-and-write operation is situated within the execution module. Following the result of the execution, the data is written back to the registers or memory at the subsequent rising edge of the clock.

In the modern era, the number of CPU pipeline stages has surpassed three, yet tinyriscv maintains a three-stage structure. This is primarily due to the simplicity of the three-stage pipeline, which is straightforward to implement in hardware. Furthermore, the three-stage sequential launch pipeline allows the processor to consider power consumption and performance simultaneously, thereby meeting the design requirements while also prioritising low-power consumption[7].

3.1.2 Pipeline hazards

The presence of overlapping instructions within a pipeline can result in the emergence of pipeline data hazards, which are characterised by their correlation with one another. There are three principal categories of hazard. A structural hazard arises when a component is requested by more than one instruction, resulting in insufficient hardware resources in the pipeline. A data hazard can be caused by an instruction requiring an operand that is the result of a write-back of the instruction that precedes it [8]. Finally, a control hazard can be caused by a branch or jump instruction that changes the PC value of the instruction address, thereby determining the flow of program execution based on the result of the previous instruction.

1) Structural hazards and solution measures

In a Von Neumann architecture, instructions and data may be subject to structural hazards due to their sharing of a common memory [3]. The processor core designed in this paper employs a Harvard architecture to circumvent this issue. By separating the storage of instructions and data, each memory is addressed and accessed independently, thereby meeting the requirements of a processor that is capable of parallelised data processing.

2) Data hazards and solution measures

In consideration of data hazards, it can be observed that the processor cores designed in this paper are single scalar processor cores with sequential execution. Consequently, the data hazards that arise are limited to those of the write-after-read variety. In accordance with the aforementioned three-level pipeline partitioning of the processor core, only adjacent instruction data hazard situations are to be expected [9]. An adjacent instruction data hazard refers to a scenario in which two instructions, designated A and B, are situated in a sequential order, with A preceding B. In this configuration, B is responsible for reading an operand register, while A is tasked with writing the destination register. However, if B accesses the operand and A is still in the third stage of the pipeline, it may not have stored the most recent results in the destination register. Consequently, the value of the registers read by B may not be the most up-to-date.

The occurrence of data hazards can be identified during the decoding phase. In the decode phase, the number of all registers to be utilised in the instruction is determined. In the event that the register to be read in the decode phase is identical to the register to be written in the preceding instruction, a data hazard is identified. This paper proposes a solution to the aforementioned problem by verifying whether the number of the register to be read subsequent to the instruction is identical to the number of the register to be written initially. In the event of a match, the data to be written to the register is simultaneously written to the register to be read subsequent to the instruction, thereby circumventing the intermediate process of storing and subsequently reading.

3) Control hazards and solution measures

The occurrence of control hazards in the pipeline is attributable to the execution of transfer class instructions by the program. Transfer class instructions, including unconditional transfers, conditional transfers, subroutine calls, and interrupts, which are branching instructions, have the potential to alter the direction of the program during execution, thereby resulting in pipeline breaks. In this paper, the resolution of control hazards entails the pausing of the pipeline, a measure that has the consequence of significantly reducing the efficiency of the pipeline and increasing the average number of clock cycles consumed by the instruction CPI.

Branch delay slots and dynamic branch prediction are employed in numerous processor architectures to mitigate pipeline branching losses and address control hazards. However, the utilisation of branch delay slots can render hardware design intricate and unwieldy, and software programming complex and challenging to comprehend. Furthermore, in practical software programs, it is challenging to identify instructions that are not influenced by branching outcomes and suitable for placement in delay slots. The dynamic branch prediction approach necessitates greater hardware resource utilisation and is more suited to high-performance superscalar processors.

3.2 Module Division and Design

3.2.1 Finger fetch module

In this context, the term ‘finger fetch module’ is used to refer to both the pc_reg module and the if_id module as a single entity. The function of the finger fetch module is to provide the address value of the instruction and the machine code of the instruction, retrieved from the instruction memory at the precise moment in time defined by the system clock, for subsequent decoding by the next stage of the pipeline. The design of the finger fetch module is illustrated in Figure 3.

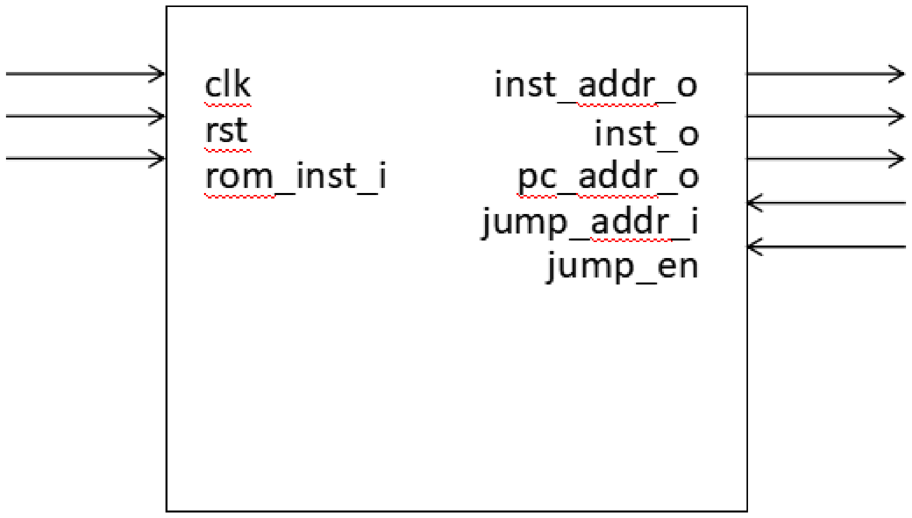


Fig.3 Finger fetch module

3.2.2 Decoding module

The decoding module generates the corresponding control information in accordance with the type of instruction, reads the value of the source operand register within the instruction, and generates the immediate number [10]. The design of the decoding module is illustrated in Figure 4.

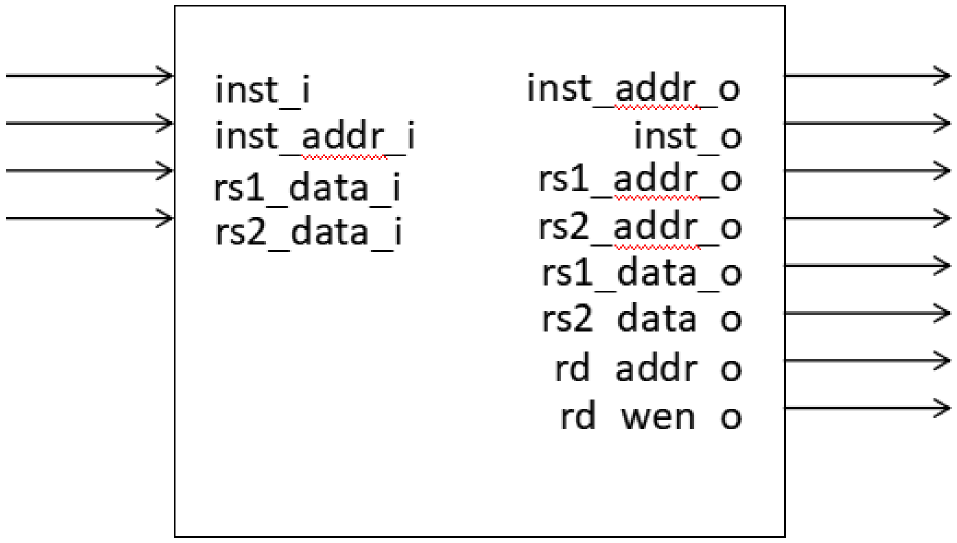


Fig.4 Decoding module

3.2.3 Execution Module

The execution module is tasked with determining the success or failure of the branch transfer, selecting two operands to participate in the operation, and obtaining the result of the operation. The result of this operation can be the destination address of the branch transfer, the data written to the destination register, or the address of the memory access, depending on the control signal. The design of the execution module is illustrated in Figure 5.

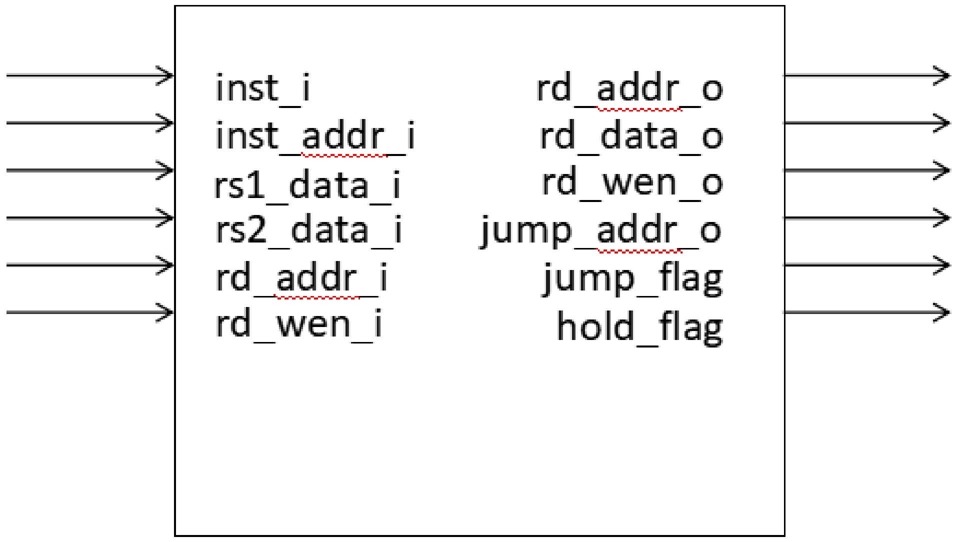


Fig.5 Execution module

3.3 Resource Optimisation

In the case of the ex module, it is typical for a number of logic gates, including those of the AND gate, the OR gate and the NOT gate, to be reused internally. As a result, it is possible to extract the logic gates and design a separate ALU module to be called during the operations in the ex module.

The specific method is to declare the result variables of the addition operation, the and operation, the different-or operation, the or operation, the left-shift operation, the right-shift operation, and the calculation of the address unit in advance in the ALU part. Subsequently, the assign statement is used to assign them with a logical operation.

ISSN 2959-6157

In this manner, during the execution phase, upon the issuance of the corresponding instruction from the decoding module, the result variable previously declared can be directly assigned to rd_data_o, thereby enabling the rd output data to directly invoke the calculated result within the ALU. This approach reduces the number of logic gates required in the ex module.

The utilisation of resources was observed using Quartus, with the selected device identified as EP4CE15F23C8. Prior to optimisation, a total of 3091 logic gates were employed. Following optimisation, the resources accounted for 3041 logical gates used.

4. Simulation Results

The limitation of pipeline size precludes the implementation of all basic instructions of RISC-V in this processor. The implemented instructions are: add, addi, andi, auipc, beq, bge, bgeu, blt, bltu, bne, jal, jalr, lui, or, ori, simple, sll.

4.1 Pipeline Validation

The design of the three-stage pipeline of this processor core and the implementation of each module have been previously described. In order to ascertain the correctness of the implementation, the processor core must be verified. Verification is the process of ensuring that the design implementation of the chip meets the expectations set out in the pre-silicon stage. It is the primary method of guaranteeing the correct functioning of the chip and reducing the risk of the chip's functionality being compromised during the manufacturing process [3].

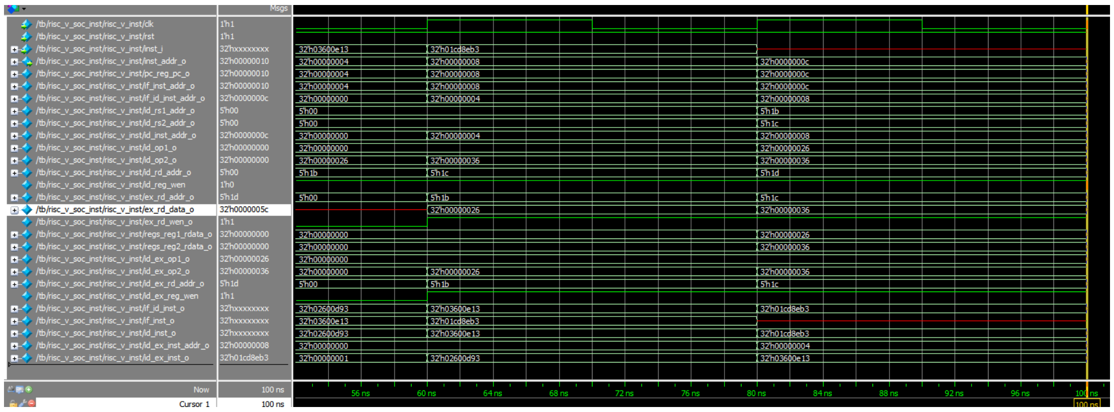
In order to check that the structure of the three-stage pipeline is correctly constructed, it is necessary to observe the running waveforms using Modelsim software simulation and to trace the execution of the instructions in each clock cycle.

MOV x27,12'd 38

MOV x28,12'd 54

ADD x29,x28 x27

The following is an example to illustrate the execution of the instruction in the pipeline. Figure 6 shows the results



	Msgs
/tb/risc_v_soc_inst/risc_v_inst/clk	1'h1
/tb/risc_v_soc_inst/risc_v_inst/rst	1'h1
/tb/risc_v_soc_inst/risc_v_inst/inst_i	32'hxxxxxxxx
/tb/risc_v_soc_inst/risc_v_inst/inst_addr_o	32'h00000010
/tb/risc_v_soc_inst/risc_v_inst/pc_reg_pc_o	32'h00000010
/tb/risc_v_soc_inst/risc_v_inst/if_inst_addr_o	32'h00000010
/tb/risc_v_soc_inst/risc_v_inst/if_id_inst_addr_o	32'h0000000c
/tb/risc_v_soc_inst/risc_v_inst/id_rs1_addr_o	5'h00
/tb/risc_v_soc_inst/risc_v_inst/id_rs2_addr_o	5'h00
/tb/risc_v_soc_inst/risc_v_inst/id_inst_addr_o	32'h0000000c
/tb/risc_v_soc_inst/risc_v_inst/id_op1_o	32'h00000000
/tb/risc_v_soc_inst/risc_v_inst/id_op2_o	32'h00000000
/tb/risc_v_soc_inst/risc_v_inst/id_rd_addr_o	5'h00
/tb/risc_v_soc_inst/risc_v_inst/id_reg_wen	1'h0
/tb/risc_v_soc_inst/risc_v_inst/ex_rd_addr_o	5'h1d
/tb/risc_v_soc_inst/risc_v_inst/ex_rd_data_o	32'h0000005c
/tb/risc_v_soc_inst/risc_v_inst/ex_rd_wen_o	1'h1
/tb/risc_v_soc_inst/risc_v_inst/regs_reg1_rdata_o	32'h00000000
/tb/risc_v_soc_inst/risc_v_inst/regs_reg2_rdata_o	32'h00000000
/tb/risc_v_soc_inst/risc_v_inst/id_ex_op1_o	32'h00000026
/tb/risc_v_soc_inst/risc_v_inst/id_ex_op2_o	32'h00000036
/tb/risc_v_soc_inst/risc_v_inst/id_ex_rd_addr_o	5'h1d
/tb/risc_v_soc_inst/risc_v_inst/id_ex_reg_wen	1'h1
/tb/risc_v_soc_inst/risc_v_inst/if_id_inst_o	32'hxxxxxxxx
/tb/risc_v_soc_inst/risc_v_inst/if_inst_o	32'hxxxxxxxx
/tb/risc_v_soc_inst/risc_v_inst/id_inst_o	32'hxxxxxxxx
/tb/risc_v_soc_inst/risc_v_inst/id_ex_inst_addr_o	32'h00000008
/tb/risc_v_soc_inst/risc_v_inst/id_ex_inst_o	32'h01cd8eb3
Now	100 ns
Cursor 1	100 ns

Fig.6 Waveform Simulation Results

The initial value of the pc register is 0x00. It can be observed that the value of the PC register progresses through 0x04, 0x08 and 0x0C in the final several clock cycles. This indicates that three instructions are read and continuously sent to the decoder module. Following decoding, the values of reg1_rdata_o and reg2_rdata_o in the final clock cycles are 32'h00000026 and 32'h00000036, respectively. The value of the register then successively goes through 32'h00000026, 32'h00000036, 32'h0000005c, which are the decimal numbers 38, 54, 92. This indicates that the execution module successfully added the two numbers and that the three-stage pipeline was correctly implemented.

4.2 riscv_test Set

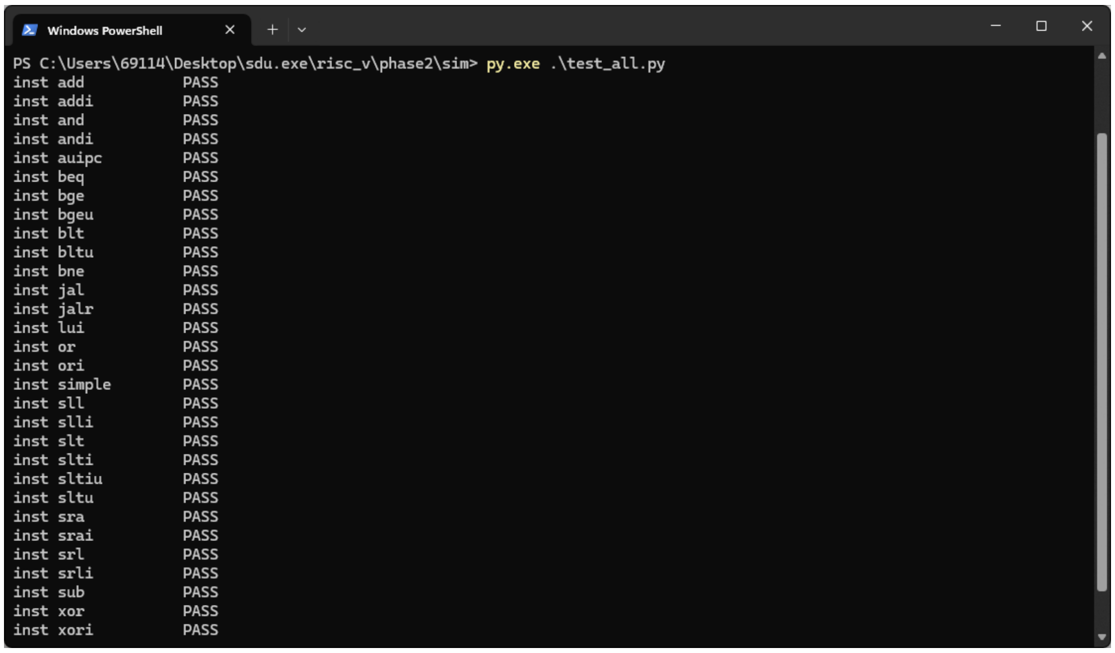
The RISC-V community has developed an official test set [11] that provides tests for each of the different RISC-V instruction variants. unit tests for each instruction are provided in riscv-tests. Below are some of the disassembled snippets from the add test cases:

```
000004cc <test_38>:
4cc: 01000093         li ra,16
4d0: 01e00113         li sp,30
4d4: 00208033         add zero,ra,sp
4d8: 00000e93         li t4,0
4dc: 02600193         li gp,38
4e0: 01d01463         bne zero,t4,4e8 <fail>
4e4: 00301863         bne zero,gp,4f4 <pass>
000004e8 <fail>:
4e8: 00100d13         li s10,1
4ec: 00000d93         li s11,0
000004f4 <pass>:
4f4: 00100d13         li s10,1
```

4f8: 00100d93 li s11,1

Here the programme will check whether the addition of two numbers is the expected result or not, and jump to the fail or pass section accordingly. That is, after the unit test is performed on the instruction, if the instruction is executed normally, it will eventually jump to the pass section and the values of registers s10 and s11 will be set to 1; otherwise, it will jump to the fail section and the values of registers s10 and s11 will be set to 1 and 0 respectively. s10 is 1 to indicate that the test is complete, s11 is 1 to indicate that the test is passed, and s11 is 0 to indicate that the test is not passed. In this way, it is only necessary to check the values of s10 and s11 to know whether the unit test has been completed and whether the unit test has passed.

As it is time-consuming to test the RISC_V basic instruction set one by one, and also for the convenience of observing the test results, this paper writes Python scripts to automatically test the basic instructions in batch mode. The result of the execution of each instruction is output as shown in Figure 7.



```
PS C:\Users\69114\Desktop\sdu.exe\risc_v\phase2\sim> py.exe .\test_all.py
inst add            PASS
inst addi           PASS
inst and            PASS
inst andi           PASS
inst auipc          PASS
inst beq            PASS
inst bge            PASS
inst bgeu           PASS
inst blt            PASS
inst bltu           PASS
inst bne            PASS
inst jal            PASS
inst jalr           PASS
inst lui            PASS
inst or             PASS
inst ori            PASS
inst simple         PASS
inst sll            PASS
inst slli           PASS
inst slt            PASS
inst slti           PASS
inst sltiu          PASS
inst sltu           PASS
inst sra            PASS
inst srai           PASS
inst srl            PASS
inst srli           PASS
inst sub            PASS
inst xor            PASS
inst xori           PASS
```

Fig.7 Results of python script

As can be seen from the figure, the instructions designed in this paper have passed the unit tests provided in riscv-tests for each instruction, verifying that the functions of each instruction of the designed processor have been realised and the basic processor core design is complete.

5. Discussion

In this paper, the verification of processor cores is done by manually writing test programs using the riscv_test set, but not by generating randomly motivated tests. Although the manually written test program traverses all instructions, it is difficult to cover the various combinations of instructions, and the random generation of test stimuli can be used to traverse all combinations of instructions to ensure the completeness of the processor verification.

There are still some areas for improvement and extension in this paper.

When solving pipeline control hazards, in addition to the pipeline suspension method adopted in this paper, further research can be conducted to adopt solution strategies such as dynamic branch prediction, data relevance checking and garbled write-back, respectively, to improve the performance of the processor.

The tinyriscv, as a scalar processor, can only decode one extended instruction at a time, which restricts the instruction performance, and the subsequent can be based on the extended instructions. Accelerator portability, the use of multi-core or multi-threaded main processor architecture to achieve parallel execution of instructions to improve computing efficiency.

6. Conclusion

This paper analyses the characteristics of the RISC-V instruction set architecture, combines the characteristics of the RISC-V architecture with the performance of the tinyriscv processor to carry out a certain pipeline data hazard analysis, and then carries out the design of the processor core, and after the completion of the design is

ISSN 2959-6157

appropriately optimised for resources. After designing the RISC-V processor core, the design of the three-stage pipeline was verified by observing the simulation waveforms, and the basic instructions of the designed processor core were verified by manually writing the test programs to achieve the basic functions of the processor core.

References

[1] Kang Songmer,YusufLeblebici,ChulwooKim. CMOS digital integrated circuits:analysis and design. Electronic Industry Press,2015.

[2] Wang, C.Y., Zhang, Chunyuan, Shen, L. et al. Computer Architecture. Beijing: Tsinghua University Press. 2015, 11-53

[3] Peng, Xiaode. Processor core design and SoC implementation based on RISC-V instruction set. Hunan University, 2022.

[4] Jia S-M. Optimisation of RISC-V processor core design and implementation of extended instruction set Shandong University, 2023.

[5] Hu Zhenbo. Hands-on teaching you to design CPU RISC-V processor chapter. Beijing:People's Posts and Telecommunications Press,2018:34

[6] Michael J. Flynn, Y.Q. Lu, and C. Zhang. Computer system design:System on a chip. Mechanical Industry Press,2015.

[7] Shi L. Design and Implementation of Extended Instruction Microprocessor Based on RISC-V Architecture. Beijing University of Chemical Technology, 2021.

[8] Jiemin,Zhang Shanfeng. Study and performance optimisation of a five-stage pipelined RISC-V processor. Microelectronics and Computing, 2022, 39(03): 78-8

[9] Lei Silei. Do-it-yourself CPU writing. Beijing: Electronic Industry Press, 2014, 107-180

[10] https://github.com/riscv-software-src/riscv-tests.git