RTL Design of the 5 Stage Pipelined Architecture of RISC V Processor

Aakash Gupta
M.Tech. ECE
Indraprastha Institute of Information Technology
New Delhi 110020
Email: aakash21150@iiitd.ac.in

Sachi Garg
M.Tech. ECE
Indraprastha Institute of Information Technology
New Delhi 110020
Email: sachi21206@iiitd.ac.in

NK Vaishnav
M.Tech. ECE
Indraprastha Institute of Information Technology
New Delhi 110020
Email: vaishnav21157@iiitd.ac.in

***Abstract*—RISC or Reduced Instruction Set Computer has instruction sets much simpler when compared to the CISC, and so are the addressing modes. This article is concerned with the design of the 32-bit RISC-V-based processor. Pipelining is a method by which we move instructions in order. With this method, we can get a good value on CPI. A five-stage microprocessor has been considered for the RISC-V (Reduced Instruction Set Architecture) ISA. This ISA has 32-bit general-purpose registers and multiple data and instruction memories. The data flow can be divided into windows in pipelining and pass them through each stage simultaneously. Each stage has buffers or latches in between them. It can be divided into five stages where every stage executes in parallel to implement pipelining. Sometimes in pipelining, the subsequent instructions can't be executed as there might be some dependency in the registers in the next clock cycle.**

***Index Terms*—Pipelining, Arithmetic logic unit, forwarding, stalling, MAC.**

I. Introduction

Any electronics hardware that comprises a processor uses either of the two architectures, i.e., CISC or RISC. CISC (Complex Instruction Set Computing Architecture) was developed by intel. It contains instructions that vary from more superficial to elaborate and take a longer execution time. It has a shorter length of codes, so it doesn't require larger memory space for storing. RISC (Reduced Instruction Set Computing Architecture) processors have an elementary addressing mode, so the fixed-in length instructions are simpler to implement and have a very customized set. The execution speed of the RISC processors is much higher due to their limited and simplified instruction sets compared to their counterpart CISC architecture. In RISC architecture, it is possible to implement some CISC architecture instructions by breaking them into simpler ones. Instructions in the RISC architecture can be executed in one clock cycle (so RISC has CPI=1), and each CPI has a fetching, decoding, and execution unit. RISC architecture comprises main memory supported by two small cache memory (instruction cache and data cache). It has a hardwired control unit to control the processor's components with the data and address bus.

Along with these, CISC architecture consists of a micro-program control unit and instruction path. The performance of the RISC architecture depends and varies according to the code being executed because any instruction that is executed presently may rely on previous instructions.

RISC is a microprocessor that improves performance by parallelizing the instructions. That means the speed of the machine is much more or faster by increasing a million instructions per second, or the known name is MIPS. If a million instructions per second are much more, the system's performance is very high. It doesn't imply that the processor is fast if a million instructions per second are high. So a million instructions per second are not the sole variable that decides the processor's speed, and it is not a good practice to use this as a parameter to distinguish between processors. Many known processors use this RISC architecture to make the system work fast. Some improvements can be made apart from improving a million instructions per second.

- The development and testing of an advanced or updated processor can be made with low effort and small-time.
- The applications and the operating systems that use ISA can be developed to establish the system. It is easy to do the work using the RISC ISA as its a smaller, lighter, and faster architecture.
- As the architecture is more flexible, it becomes easier to use a large amount of space on the chip, and it becomes an easy task to evaluate.
- The compilers of higher-level produce the code that is more efficient than the present compiler.

So these are the main reasons why a small set of architecture is used to build a primary, fast, and functional operating system.

II. Pipelining Architecture

The pipeline is the technique used for the RISC architecture, which enhances the processor's performance by reducing the execution time marginally by parallelly fetching, decoding, and executing the instructions without altering the logic of the instructions code and resulting in a reduced CPI (number of clocks per instruction). In the pipeline architecture, each segment comprises flip flops which store or hold the data, and then the combinational circuit follows it. The flip-flops transmit the data to the combinational circuit to perform

operations when the clock triggers the flip-flop. It is done at each pipeline level when the processor may isolate one stage's process from another, allowing another stage to function independently. Hence it allows running multiple instructions at the same time.

The pipeline stages used in this processor are fetched, decoded, executed, memory, and written back. Now, if the one instruction reaches the solve step after being brought, the other instruction is available and continues. With lots of benefits, some hazards come with pipelining, which are structural hazards; it occurs when another stage uses a functional unit of one location. Data hazard occurs when the register required by an instruction is already in operation. And another hazard associated with the pipeline is the controlling hazard which occurs when there is branching instruction. Several techniques are available to get over the hazards, but this processor uses stalling and forwarding for structural and data hazards.

III. Hazards In Pipelining

A. Data dependency

If a register is used to store the result in the previous instructions used in the next cycle, it forms a result dependency or RAW dependency. Hence, there are two ways to solve this problem; we can wait till the result is written back into the X2, or we can bypass the output of the next stage, so we must get a correct result, so we avoid the instruction to get the correct result. Thus, if any instructions have any dependency, it is known as data dependency.

B. Memory Delay

Generally, when an instruction is encountered, it gets searched in the Cache memory. If the required data is not found, it is a cache miss. Now again, a search is being conducted for more data that is farther and takes ten cycles or even more cycles. So, it is required for the respective pipeline to stall for a selected number of processes, a memory delay hazard. This miss of the cache also results in the delay of the further instructions.

C. Branch delay

Let's look at how the four instructions are piped together. If the 1st instruction is considered a branch instruction, it targets the instruction. The microprocessor starts working, and it fetches, decodes, then executes, and finally, it moves through memory and bypasses, but the result gets computed in the 4th cycle. The term was already fetching the other three instructions. As a result, further instructions will be branched out to different locations. All those instructions are intended to be squashed until now. Then the operations are expected to be performed, which means two instructions will be stamped into a single-stage pipeline, known as instruction branching. The corresponding delay is known as branch delay.

IV. Stages Of Operations

A. Instruction Fetch

Instruction from memory is retrieved from the location where the PC points. If there is a branch instruction, store the current PC to another NPC and increment the program counter value. The staging processor extracts the instructions from the instruction memory in the fetch. And after pulling instructions, the program counter increment by four as the processor is 32 bits.

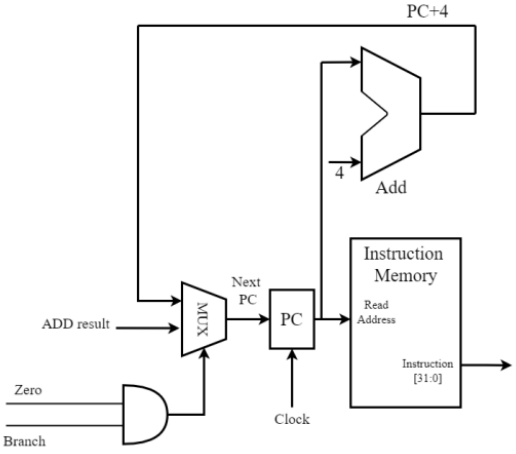

Fig. 1: Fetching and updating program counter.[1]

B. Instruction Decode

In this stage, instruction decode and register read were performed. Instruction fetched from IF is in the instruction register, which is decoded. According to the instruction format, reading register operands is done in parallel to decode. The values of load/store and arithmetic units in this stage might be skipped to utilize them in the following step. The instruction decoder decodes the instructions which are fetched. It solves the destination register, source register, and associated paths. They are transmitted to the execution stage to execute.

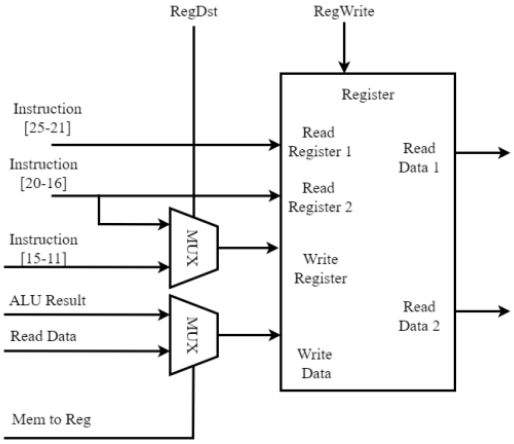

Fig. 2: Instruction Decoding.[1]

C. Instruction Execute

All instructions are executed in this stage. When required, an effective address for memory access is also calculated here. It is usually using an ALU here. The new program counter value is checked and updated for the branch instruction. The data decoded at the decode stage is received at the execution stage, and at this stage, the data processing takes place. At the execution stage, there are different modules such as arithmetic logic unit, adder, multiplier, etc.

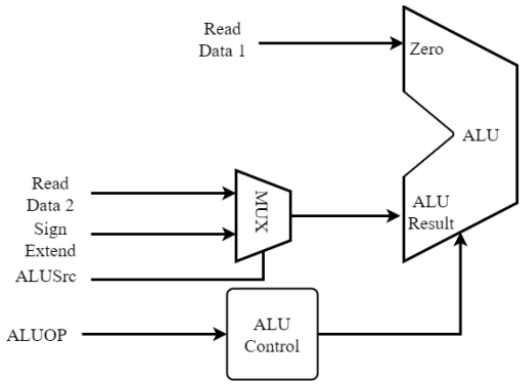

Fig. 3: Execution Stage.[1]

D. Instruction Memory

This step is not used for most of the instructions as it requires memory access, which is used by load/store instructions. For the Branch, the program counter has two addresses. When the flag is one, the program counter jumps that address. Otherwise, the following address is given for decoding.

E. Instruction Writeback

The output after this stage goes into the register file and updates the value of the register in the Writeback stage. The position of the destination register depends on the instruction, which is known after the decoding has been done.

V. System Architecture

The 32-bit processor, a Reduced Instruction Set Computing Architecture, is a Harvard architecture created on a MIPS processor with a five-stage pipelining implementation. Designing the 32-bit processor, which has a Reduced Instruction Set Computing Architecture, is a success in terms of execution speed, leading to better performance. The design proposed in the paper uses the following modules to implement a 32-bit RISC processor.

A. Program Counter

All the instructions and the data in storing element of the system have a physical address used to execute particular instructions. The program counter is the register in any processor used to extract the physical address of the following instructions specified in the instruction memory that is to be fetched. It stores the following instruction address.

B. Arithmetic logic unit

The arithmetic logic unit is the heart of any digital computer processor, enabling them to perform many mathematical operations. Its simple form can be considered a binary calculator that calculates various data operations. Arithmetic can also generate flags, simple status bits used to represent the logical arithmetic unit's operation outcome. Such as the zero status bit in the flag register being set if the process produces zero results; similarly, the logical arithmetic unit can generate many status bits. But modern computing machines are far more complex than the described, capable of performing arithmetic operations such as addition, subtraction, multiplication, dividing, etc., a logical operation greater than less than operation, a bitwise operation such as AND OR, XOR, etc.

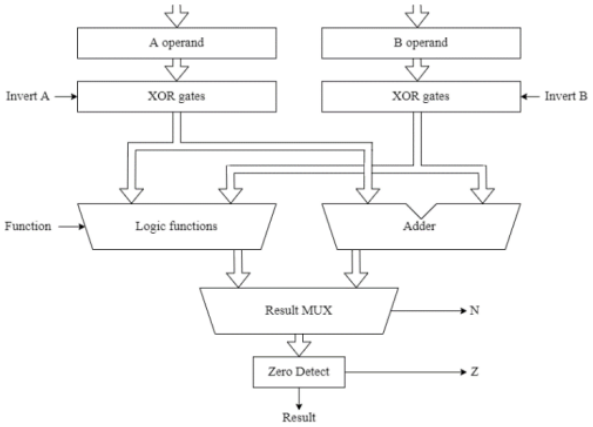

Fig. 4: ALU architecture.[1]

C. Control Unit

The Control unit for the RISC architecture controls the data flow from and into the central processing unit (CPU). Also, it is responsible for maintaining the operation of the arithmetic logic unit. It also guides the memory and input-output devices on how to perform the functions on the instruction sets. Because the architecture uses 5-stage pipelining to isolate the functional units, the same control signal may affect another set when one control signal controls one pipelining stage. As a result, it can also be possible that it doesn't work on another when the clock comes for one place. Still, the control signal is standard for both locations; therefore, even the control signal is implemented on the pipelining.

D. MAC Unit

In the processor, two stages are pipelining into the MAC. In the first multiplier stage, 32bit multiplier and 32bit multiplicand are stored in two registers. At the rising edge of the system clock, multiplicand and multiplier become inputs of Booth multiplier, which produces the output of 64-bits. In the next cycle of the clock, the accumulator register releases the result of a Booth multiplier and an accumulator's production of the previous clock. An accumulator adds two inputs through

a 64-bit carry select adder and makes two outputs. One is 1-bit overflow, and the other is the 64bit accumulation result since both multiplier and accumulator are independent, and applying the 2-stage pipelining is a more straightforward process.

E. Stalling

The RISC processor having the five-stage pipelining has its advantages, but a few of the pipeline's disadvantages are hazards. Stalling is the technique used to eliminate a few of the conflicts of pipelining. Basically, in this processor, stalling is used before implementing the forwarding. Hence the stall technique is used to eliminate the data hazard and the controlling hazard; in the control conflict, it's the skipping of the instructions which are already fetched before the execution stage could identify a branch instruction. Hence, this processor stall technique attempts to eliminate the data hazard. The stalling approach comes with its demerit as it increases the instruction's latency, increasing the CPI. Stalling becomes worst during the control hazard, as the longer stalling happens, the higher the cost of CPI. Also, the throughput of pipelining is hard to predict, making the processor more complex.

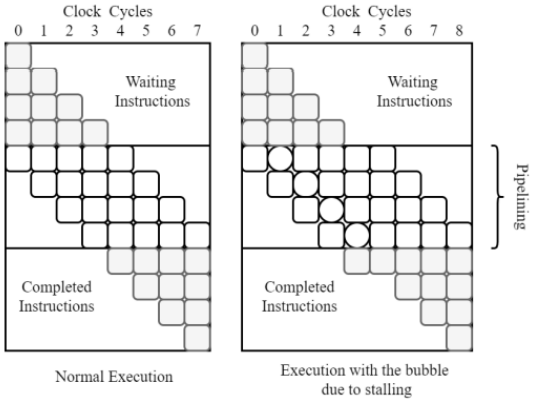

Fig. 5: Basic representation of stalling.[1]

F. Forwarding

Stalling was used to overcoming the data hazard, but the CPI increased marginally after applying the stalling technique. To avoid this, bypassing or forwarding is used to prevent this. Forwarding is feedbacking the result of the execution stage to the decode stage, so when the register is currently used, it does not have to write back. The executed value can be excessed before it.

VI. Disadvantages of pipelining

Many problems arise with the pipelines during the execution. Every designer uses various methods to overcome these issues; some of them are listed below, which are used primarily to reduce the time per instruction or CPI value and increase the speed of the processing of the execution.

- Designing a non pipelined processor is more accessible than designing a pipelined processor as it allows each instruction to complete its writeback stage. Then only the next instruction enters the pipeline, affecting the processor's CPI, speed, and performance. By using the pipelining, the CPI reduces. The speed of the processor increase, and non-pipelined instructions prevents branch delays in pipelining. All the branches are delayed, so there is some merit of hard work observed over here, so the use of pipelines improves all the required parameters.
- So generally including a flip flop between the pipeline to reduce the latency, which happened by cutting the whole instructions into pipeline stages; usage of flip flop helps us a lot in CPI and processor performance and speed increase and system throughput.
- In a non pipelined processor, all the instructions take time to execute. In pipelined structure mainly depends on many factors like dependencies, resource usage, etc.; considering all the dependencies, the throughput of a system in a pipelined system is calculated.
- Recent pipelines are of exceptionally high stages for supposing Intel Pentium uses 44 locations, so managing the flow control becomes a big task. So the advantage to the higher throughput becomes reduced as the number of the branches increases, so the processor is in tented to wait till the program leaves the pipeline out of the functional unit, which can be reduced by using the branch prediction theory built on earlier activity.
- But it's not always true that all the instructions are dependent. It's also not always true that all the instructions are independent, so simply for a pipeline to complete instruction may take five stages, but it's not always true that all the channels need five cycles. Hence, constructing the pipeline structure makes the pipeline dependent and speeds up the circuit. It is required to run four subsequent instructions in a pipeline run. Depending on these instructions and the first instructions, the pipeline should be managed until the channel's complete dependence is resolved.

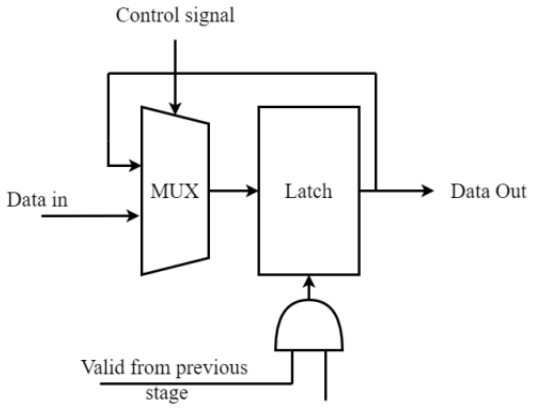

Fig. 6: Basic representation of forwarding.[1]

VII. Bypassing, Branching Load Instruction

Many problems arise with the pipelines during the execution. Every designer uses various methods to overcome these issues; some of them are listed below, which are used primarily to reduce the time per instruction or CPI value and increase the speed of the processing of the execution.

- **Memory Bypassing**: The second instruction bypasses the value from the first instruction. When there is a memory instruction(like Load) in which the processor is supposed to load data from memory, the situation may arise that before writing the data in the destination register, because of pipelining next instruction may need it to perform its operation. Now due to this situation, a RAW hazard can occur. So to prevent this, a bypassing logic can be used where the forwarding unit will bypass the value which has to be stored back into the given destination register straight from memory which will be further used by the next instruction to execute its operation. It helps in reducing the clocks required compared to bypassed logic.
- **ALU Bypassing**:The value from the first instruction is bypassed in the second. In ALU bypass, when there are two ALU instructions in which one depends on the other. So, rather than waiting for one instruction to perform its ALU operation and then write back the corresponding data into the memory, in that case. And then directly bypass the data from the execute stage where ALU operation will be performed to the following instruction operand, which needs that data to proceed with its ALU operation. So here, rather than stalling our clock and waiting for writeback to happen just directly bypassed the data and completed our execution in less no. Of clocks, hence reducing the CPI.
- **Load instruction**:It shows how it moves between the stages and updates the values. Load is a data transfer instruction used to transfer the data from a particular location in the memory to our destination register. This instruction uses a 12-bits offset address and two registers, where some register in the instruction set is the destination register where our data will be from memory. And other is the source register, whose value will be added to the offset to compute the memory address from where the data will be loaded.
- **Branch instruction**:It shows how it moves between the stages and updates the values. Branch instructions are used to change the flow of a code or to jump to some other particular location. There are two types of branch instructions, i.e., conditional and non-conditional (e.g., jump). Be A, B, offset(PC). So basically, in this instruction, there are two source registers, and alu operations are performed if the subtraction of two values is zero. The Branch will be considered taken, and our pc will move to a new location by killing one instruction fetched earlier when the Branch was getting executed.

VIII. Conclusions

The paper presents a microarchitecture model for a RISC CPU. This suggested microarchitecture outperforms previous versions. The most noticeable characteristics of RISC-V ISA are effortless to interpret and pretty much straightforward to use, which makes it superior to other ISAs, and all of the instructions are simple for scheduling and carrying out hazard detection.

The whole paper has the microarchitecture model of the RISC processor and the removal of the conflict of resources. The primary purpose is to compare the machine's efficiency when the program is run without bypassing vs. with bypassing. Suppose the same line of code is executed without bypassing. In that case, the number of clocks required per instruction will be more significant because the stall will occur for some cycles and wait for the previous instructions to complete their execution before writing it back to the register file to be used as an operand subsequent instructions. With the forwarding logic, data will be directly bypassing either from the execute stage of the memory stage for the arithmetic bypassing or memory bypassing. Hence, the number of required clock cycles is reduced, making our machine much faster for the instruction to execute. In today's world, the need for high-speed processors that can run our instructions with a minimum usage of the CLK and minimum CPI, so this bypassing gives a significant advantage to fulfill the same. The simulation shows that the stalling, forwarding, and branching operation has shown correct functionality using the 5-stage pipelining. And the forwarding was also performed from the memory and writeback to the execution stage. Finally, the MAC unit was also included in the RISC architecture, which can be used in signal processing and matrix operations. Hence, it enhances the use case of the architecture.

[]

References

[1] S. A, "Design of low power 32- bit risc processor using verilog hdl," *International Research Journal of Engineering and Technology*, vol. 6, 2019.

[2] T. McGrew, E. Schonauer, and P. Jamieson, "Framework and Tools for Undergraduates Designing RISC-V Processors on an FPGA in Computer Architecture Education," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. Las Vegas, NV, USA: IEEE, Dec. 2019, pp. 778–781. [Online]. Available: https://ieeexplore.ieee.org/document/9071132/

[3] M. Gokhale and J. Stone, "NAPA C: compiling for a hybrid RISC/FPGA architecture," in *Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines (Cat. No.98TB100251)*. Napa Valley, CA, USA: IEEE Comput. Soc, 1998, pp. 126–135. [Online]. Available: http://ieeexplore.ieee.org/document/707890/

[4] T. Gokulan, A. Muraleedharan, and K. Varghese, "Design of a 32-bit, dual pipeline superscalar RISC-V processor on FPGA," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*. Kranj, Slovenia: IEEE, Aug. 2020, pp. 340–343. [Online]. Available: https://ieeexplore.ieee.org/document/9217851/

[5] N. Albuquerque, K. Prakash, A. Mehra, and N. Gaur, "Design and implementation of low power reservation station of a 32-bit DLX-RISC processor," in *2016 International Conference on Information Science (ICIS)*. Kochi, India: IEEE, Aug. 2016, pp. 217–221. [Online]. Available: http://ieeexplore.ieee.org/document/7845330/

[6] A. Raveendran, V. B. Patil, D. Selvakumar, and V. Desalphine, "A RISC-V instruction set processor-micro-architecture design and analysis," in *2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA)*. Bengaluru, India: IEEE, Jan. 2016, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/7593047/

[7] A. Pandey, "Study of data hazard and control hazard resolution techniques in a simulated five stage pipelined RISC processor," in *2016 International Conference on Inventive Computation Technologies (ICICT)*. Coimbatore, India: IEEE, Aug. 2016, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7824864/

[8] A. Raveendran, V. Patil, V. Desalphine, P. M. Sobha, and A. David Selvakumar, "RISC-V out-of-order data conversion co-processor," in *2015 19th International Symposium on VLSI Design and Test*. Ahmedabad, India: IEEE, Jun. 2015, pp. 1–2. [Online]. Available: http://ieeexplore.ieee.org/document/7208117/

[9] T. Kanamoto, M. Fukushima, K. Kitagishi, S. Nakayama, H. Ishihara, K. Kasai, A. Kurokawa, and M. Imai, "A Single-Stage RISC-V Processor to Mitigate the Von Neumann Bottleneck," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*. Dallas, TX, USA: IEEE, Aug. 2019, pp. 1085–1088. [Online]. Available: https://ieeexplore.ieee.org/document/8884919/

[10] S. P. Katke and G. Jain, "Design and implementation of 5 stages pipelined architecture in 32 bit risc processor," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 4, pp. 340–346, 2012.

[11] V. Prasanth, V. Sailaja, P. Sunitha, and B. V. Lakshmi, "Design and implementation of low power 5 stage Pipelined 32 bits MIPS Processor using 28nm Technology," vol. 8, no. 4, p. 5.

[12] C. Chen, X. Xiang, C. Liu, Y. Shang, R. Guo, D. Liu, Y. Lu, Z. Hao, J. Luo, Z. Chen, C. Li, Y. Pu, J. Meng, X. Yan, Y. Xie, and X. Qi, "Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension : Industrial Product," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, May 2020, pp. 52–64. [Online]. Available: https://ieeexplore.ieee.org/document/9138983/

[13] G. Liu, J. Primmer, and Z. Zhang, "Rapid Generation of High-Quality RISC-V Processors from Functional Instruction Set Specifications," in *Proceedings of the 56th Annual Design Automation Conference 2019*. Las Vegas NV USA: ACM, Jun. 2019, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/3316781.3317890