IMPLEMENTATION AND EXTENSION OF BIT MANIPULATION INSTRUCTION ON RICS-V ARCHITECTURE USING FPGA

***Mr.Chekuri Mahesh*[1]*., Chinnam Neha*[2]*., P.Sravani*[3]*., V.Keerthana*[4] *., J.Navya*[5]**
1 Assistant Professor, Department of ECE., Malla Reddy College of Engineering for Women., Maisammaguda., Medchal., TS, India (chekurimahesh364@gmail.com)
2, 3, 4, 5 B.Tech ECE, (19RG1A04H8, 19RG1A04G2, 19RG1A04H4, 19RG1A04E3),
Malla Reddy College of Engineering for Women., Maisammaguda., Medchal., TS, India

Abstract

Computing systems in consumer electronics must fulfil specific characteristics in order to be dependable, cost-effective, and environmentally friendly. RISCV is a well-known ISA since it supports native hardware implementation rather than simulations and can easily accommodate large ISA extensions in specialized variants (ISA). Bit Manipulation Instructions (BMIs) were developed by ARM and Intel to reduce the overhead of running a programme and its impact on the environment, but although the RISC- V ISA is widely used, it only supports the two most basic BMIs. In this study, we introduce the simplified design of bidisc, a 32-bit synthesizable processor based on the public domain RISC- V (RV32I) instruction set architecture. In addition, we present two novel RISC-V BMIs and implement them on our developed processor, which is optimised for power, cost, and design complexity and is targeted at low-cost Embedded/IoT devices. We have prototyped the "bidisc" single-cycle processor on the "Zed Board" FPGA, which is the result of our simplified design and the usage of Verilog HDL.

INTRODUCTION

A highly efficient system with low cost and power consumption, as well as better energy efficiency and safety, is often required of computer devices for consumer electronics. Saving cash might mean reducing productivity. Processors based on the ARM architecture are widely used in IoT/Embedded systems because of the benefits they provide in terms of performance, system complexity, cost, power consumption, heat dissipation, and so on. [2] The bulk of the embedded/IoT market is driven by x86 and ARM CPU's. Several RISC-based architectures have been produced as a consequence of the need for low-cost and low-power embedded system devices, each with its own flavour to ideally serve a different set of applications. Computer architecture has been the subject of intensive study for decades, resulting in optimizations and the creation of new, more powerful ISA. Just recently, for instance, the RISC-V instruction set architecture (ISA) became available to the business and academic sectors as open-source software. It's possible to build it on the actual hardware without resorting to simulations since it provides complete support for huge ISA extensions and specialized

versions. RISC-V is an ISA that supports (32-, 64-, and 128-bit) encoding and embedded applications thanks to its four integer bases. In addition, a wide variety of Standard Extension ISAs may be used for generic software and hardware development. Intel and ARM developed the Bit- Manipulation Instruction (BMI) as an extension to the x86 and ARM ISA for microprocessors to boost the performance of bit manipulation operations [3]. Power efficiency and performance of the CPU may be greatly improved by using BMIs to streamline the processes that would otherwise create complex circuitry that requires more resources and power. Considering this is even more important when designing for Low-Cost Embedded/IoT devices if complex operations like square root etc. need to be conducted. In the areas of Machine Learning, Steganography, and cryptography, algorithms use binary data to carry out tasks like bit rotation, grouping, shifting, etc. The application of BMI in this context has the potential to greatly reduce energy and resource waste, enhancing the device's efficiency. In this study, we provide the RV32I base-instruction set, an expansion of the conventional RISC-V ISA that adds two BMI instructions to the 32-bit RISC-V processor architecture. The study goes into depth on how BMIs were integrated into the RISC- V ISA and the "bidisc" central processing unit. The "bidisc" central processing unit is a single-cycle, single-core design that only supports the RV32I instruction set. Real-Time Embedded System Applications, including the Internet of Things, Signal Processing, and More In order to develop inexpensive electronic devices, we sought to enhance the architecture in terms of power, cost, and design complexity at the expense of having precise timing constraints. The "bidisc" processor is utilized to implement the new BMI for the same reason. Using a field-programmable gate array (FPGA; Zed Board), we prototyped the processor and implemented it to boost the performance of our new BMI microprocessor. The article is structured as follows. Section II provides a brief summary of RV32I, an examination of the differences between the different ISAs, and a review of the related literature. Section III describes the structure and operation of a processor. Our newly built RISC-V BMIs and their applications are discussed in Section IV. Results, simulations, and used FPGA resources are all presented in Section V. In Section VI, the paper finishes with some last reflections and recommendations for further study.

LITERATURE REVIEW

The RISC-V ISA has seen widespread adoption around the globe due to its numerous advantages. In [4], the differences and similarities between MIPS, RISC-V, Open reach, and ARM, four popular instruction set architectures, were examined. Speed, static code size, and power efficiency are all enhanced by the RISC-V Instruction Set Architecture (ISA) since over-architecting is not required and a variable-length instruction encoding space is supported. RV32I was created to reduce the strain placed on today's computers. RV32I instructions come in six different flavour, denoted by the letters R, I, S, B, U, and J, and all of which display instantaneous variants [1]. RV32I uses a unique encoding scheme for each of its 40 different instructions. The addressing mode for all instructions is a byte, and the maximum width of the instructions is 32 bits. Although on a separate set of architectures, RV32I has been implemented in [5]. When BMIs were first introduced, it was not until 2012–2013[3] by either Intel or ARM. The advantages of x86 BMI have been thoroughly analysed, and new RISC-V BMIs have been designed [6]. Rocketpunk utilizes barrel shifters for rotl and rotor operations and a divide and conquer algorithm for calls and cuts instructions to speed up processing and consume less energy. Our findings expand the range of possible applications for body mass indices. For effective administration of DSP applications, [9] additionally provides application-specific instructions and the associated bit manipulation unit (BMU).

METHODOLOGY

The 32-bit CPU is often used in embedded systems for less demanding tasks due to its cheaper price and simpler design. The one-cycle design performed better than the other two in the preliminary performance analysis. In the next paragraphs, we'll look at the design from a more abstract level, focusing on the processor and micro-architecture.

Top module view:

A Data path Unit, Control Unit, and Memory make up the implemented single-cycle processor. Data path Unit acts as the skeleton and also stores the CPU's architecture. The Control Unit (CU) acts as a decoder and provides insight into how the various elements of the CPU will handle a new command. One's memories may be seen from two perspectives. A computer's primary memory is divided into two categories: instruction memory (where the actual programme is stored) and data memory (where information is temporarily stored). During each clock cycle, the Data path Unit's dedicated adder increases the value of the programme counter (PC), which is used by branch and jump instructions to identify their destination. The Register file module gives you access to 32 32-bit general-purpose registers, numbered R0 through R31. Since R0 is always linked to ground or logic zero, the output of any operation that employs R0 will always be zero. If the CU provides an LaFont value, the ALU module will utilise that value to carry out that function on its inputs. In response to instructions retrieved within the same time period, the control unit generates the control signal. Specifically, ALUs exclusively deal with R-R (register-register) and R-I (register-immediate) operations while processing data. Since specialized adders are used to compute their addresses, the ALU does not take part in the calculation of their addresses; instead, it establishes the condition for branch instructions and sends the result to the control unit, which then modifies the next PC value appropriately.

Micro-Architecture and its Working:

To retrieve the next instruction from programme memory, PC is increased by 4 (due to byte addressing) or by the values of peal in the Fetch and Control block at the beginning of each clock cycle. The Fetch and Control component not only delivers the command, but also forwards it to the Control Unit. A 32-bit instruction (RV32I) is evaluated by the control unit, which decides which operation to carry out based on the values of the "Opcode" (0-6 bits), "func3"(12-14 bits), and "Inst" (30th bit) [1]. The length of an instruction's Opcode (from zero to six bits) determines how it is represented in code (R, I, B, S, J, U, etc.). Two signals (instruction, branch) are sent to the Control Unit, and seven signals are returned (to the Data path Unit). It is possible to provide access to the registry in this case. It is possible to specify read/write access using the misread and miswrite switches. In the "Func3" box, you may specify the encoding of the ALU and the operation (add, sub, shift, and, or, etc.) to be carried out by the ALU. The peal line selects the next value of PC, the bell line selects the second operand to the

ALU (the B-select Block), and the sales line selects the inputs to the Register-file (the Store Logic Block). Sign-immediate the imams, immix, imp, immune, and imp values indicate the Store, Instruction, Instruction, Branch, Instruction, and Instruction encoding extensions, respectively (for J encoding). These extensions are checked for usefulness at the beginning of each cycle thanks to immediate instructions and the values of peal, bell, and sales, the select lines for their respective blocks, but are only implemented if they are deemed necessary. Different adders are employed for incrementing PC, calculating branch and jump addresses, etc. since a single-cycle processor has to be relatively simple to strike a balance between component usage between clock cycles. Instruction and Data memory may have a maximum size of 4GB and be accessed by a 32-bit address line, however for the sake of simplification and simulation, this has been reduced to 16KB. Figure 1 is an example of this concept. In order to help children with ADD, teachers might use the peal method to choose which lesson to recall (i.e., ADD),

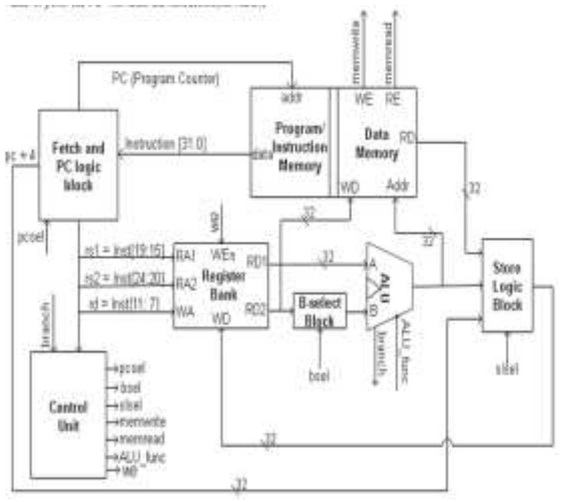
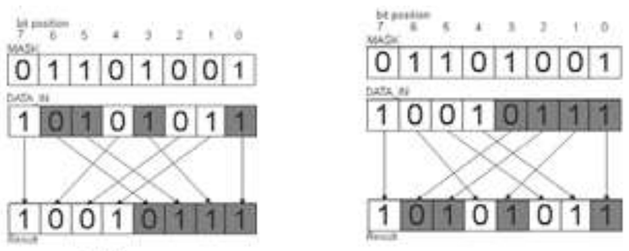


Fig. 1: Designed RISC-V Micro-Architecture.

This book presents the novel executable group and degree. By streamlining our procedures and methods, we were able to home in on hardware that did not break the bank. Instruction encoding recommendations are shown in Table 1. We settled on the R-type encoding as our preferred method since the grammar of our instructions is quite close to that of R-type data. More opcodes add layers of complexity to computers. Example: If the mask position 2 value is 1, then that value is added to the grand total. Since bits 6 and 7 of the mask are likewise 1, just the rightmost bits of the value are utilized. Similar to how a mask with all zeros in bits 1, 3, and 8 would shift the matching data in value to the left, clustering together items with the same sign. To "group" is to arrange bits into groups that match the ones and zeros in the mask. The data bits may be grouped together using the 1s and 0s in the mask bits, thanks to the group command. The first table contains some helpful data. Keeping the same bit order inside each cluster, the instruction shown in Figure 2 swaps its data input such that 1s from the mask bit are clustered on the right and 0s on the left.

TABLE I: R-TYPE ENCODING

funct7	rs2	rs1	Funct3	rd	Opcode	
0011000	rs2	rs1	000	rd	1101011	gp
0010000	rs2	rs1	000	rd	1101011	
31 25	24 20	19 15	14 12	11 7	6 0	degrup



(a) 8-bit grup (b) 8-bit degrup

Fig. 2: Operation of group and degree examples with 8-bit input data and mask

Figure 3 depicts the process flow for the implementation of the group instruction.

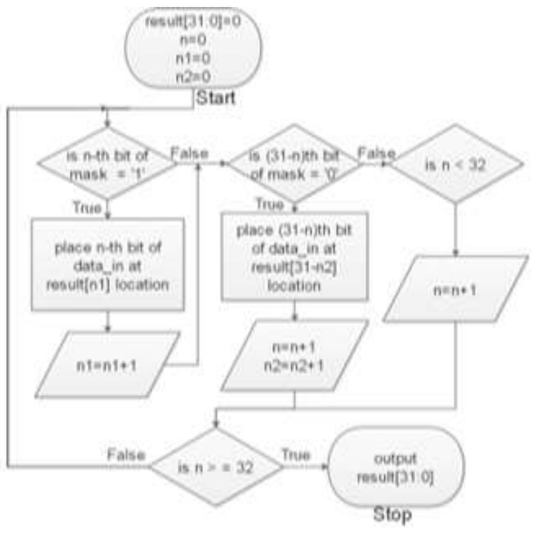

result[31:0]=0
n=0
n1=0
n2=0
Start
is n-th bit of mask = '1'
False
is (31-n)th bit of mask = '0'
False
is n < 32
True
True
place n-th bit of data_in at result[n1] location
place (31-n)th bit of data_in at result[31-n2] location
n=n+1
n1=n1+1
n=n+1
n2=n2+1
False
is n > = 32
True
output result[31:0]
Stop

Fig. 3: Flowchart of group instruction Di grouping

Similar to how group completes an action, degree undoes it. In contrast to group, the inputs and outputs are simply flipped. Figure 4b depicts data bits being positioned in the "1" places of the mask. Any bits in the data that are read from right to left are placed in the outputs for bits 1 in the mask, and likewise for bits 0 in the mask. The radix-2 sorting technique that makes advantage of the fast group instruction is helpful, as is the cryptographic primitive's field. Steganography, the study of hiding messages, may one day be used in clustering algorithms like K-means clustering, etc., as improvements are made in techniques of instruction and bit arithmetic and representation.

The "bidisc" Processor was developed, implemented, and synthesized using the behaviour modelling capabilities of Verilog HDL. Simulation and synthesis were carried out using Xilinx Viv ado 2018.3.1 and GTKW Ave, respectively. Our processor has all the hardware built in to handle the most recent RISC-V BMIs. Assembly language instructions are typically written first, then compiled or translated into machine code, and then dumped into the instruction memory before anything can be evaluated. The results of each instruction's execution by the central processing unit (CPU) are simulated and assessed using some set of criteria at the conclusion of each machine cycle. The FPGA prototype was built once the processor's functionality was shown, and the results were showed using LEDs. C code produced in the Xilinx vivid software development kit was supposed to allow the "bitkis" accelerator processor to talk to the Zed Board's programmable logic (PL) and peripheral input/output (PS) (SDK). This verification suite uses the original machine code for each instruction, which may have been produced by a C-program, a Dual-port BRAM, or even a direct memory initialization. We don't use a compiler or an assembly of any type.

TABLE II: Code snippet for our example

Assembly Code	Operation
# Code groups the bits according to mask bit placed at memory[7] from the product of two numbers placed at memory[1] and memory[2]. Output result at memory[4]	
LW R1, 1(R0)	load Reg[1] <= Mem[Reg[0] + (1)]
LW R2, 2(R0)	load Reg[2] <= Mem[Reg[0] + (2)]
ADDI R5, R0, R2	addition Reg[5] <= Reg[0] + Reg[2] or mov R5 , R2
ADDI R9, R2, -1	addition Reg[9] <= Reg[2] + SXT(-1)
ADDI R2, R0, R9	addition Reg[2] <= Reg[0] + Reg[9] or mov R5 , R9
loop : ADD R6, R5, R1	addition Reg[6] <= Reg[1] + Reg[5]
ADDI R5, R0, R6	addition Reg[5] <= Reg[0] + Reg[6] or mov R5 , R6
ADDI R7, R2, -1	addition Reg[7] <= Reg[2] + SXT(-1)
ADDI R2, R0, R7	addition Reg[2] <= Reg[0] + Reg[7] or mov R2 , R7
BNE R2, R0 , loop	branch if not equal to Zero branch <= (Reg[0] != Reg[2]) ? 1 : 0
SW R5, 3(R0)	store Mem[Reg[R0] + (3)] <= Reg[5]
LW R1, 7(R0)	load Reg[1] <= Mem[Reg[0] + (7)]
grup R8, R5, R1	group R5 according to R1 and store in R8
SW R8, 4(R0)	store Mem[Reg[R0] + (4)] <= Reg[8]

System 170 relied on accurate results from the processor's design, which were verified by individual module test benches. To the extent that your licence allows it, you are restricted to the following activities: Murdoch University is where I earned my degree. The download from IEEE Xplore completed on June 15, 2020, at 19:05:32 UTC. These constraints improve the strength of the model. The development of a square series up to a particular number, multiplication by repeated addition, the sum of the natural numbers up to N, etc. were all created as examples for the final testing phase. The hardware and software shown in Fig. is what is needed to create an FPGA prototype. The synthesized was simulated using RTL-logic level simulation. In this case, a ZED Board FPGA was used to actualize the concept. Both code and data were stored in the 16 kilobytes of on-chip BRAM. In addition, chips had built-in support for general-purpose I/O like lights and buttons. For the purpose of this tutorial, we will utilise the group command to conceal irrelevant information while multiplying two values by repeated addition. The assembly language for the following is provided in Table II, which may be accessed via the associated links. Table III displays the results of this study's hardware. It has been determined that a clock frequency of 100 MHz or more is doable with our architecture. The chip itself is expected to have a power consumption of 1.109 watts.

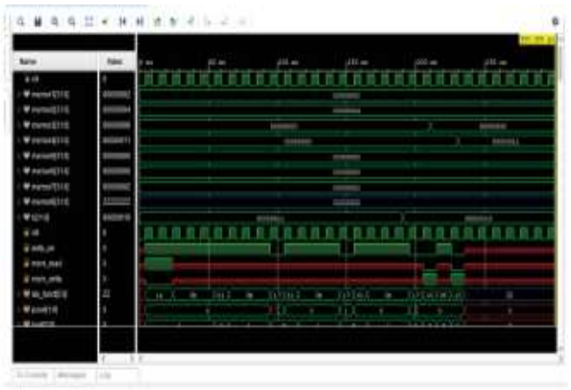


Fig. 4: Simulation output memory

The conclusion may be seen in both the most current table and Figure 4. The predicted results of this code fragment are shown in Figure 4. This experimental configuration on a Zed Board is shown in Figure 5.

TABLE III: Resource Utilization

Resources	Utilization	Available	Utilization %
LUT	2312	53200	4.35
LUTRAM	62	17400	0.36
FF	2035	106400	1.91
IO	21	200	10.50
BUFG	13	32	40.63

Fig 5: Setup for Implementation

CONCLUSION

Two novel RISC-V Basic Machine Instructions, group and degree, are presented in this research together with the ground-breaking bitkis RISC-V processor. The ultimate objective is to use the current and presented BMIs to incorporate more RISC-V extensions like M (Multiplication and Division unit), F (Single precision Floating-Point Unit), etc., into an accelerator on FPGA for a Software Defined Radio (SDR) filtering application. These results might inform future research that leads to practical, usable solutions.

REFERENCES

1. ***"The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Douc- mint Version 20191213", Editors- Andrew Water man and Krste Saanvi c, RISC-V Foundation, December 2019.***

2. ***Wikipedia, "Arm architecture — Wikipedia, the free encyclopedism- die", [Online; last accessed 31-Janurary-2020]. [Online]. Available: https://en.wikipedia.org/wiki/ARMarchitecture.***

3. ***Wikipedia, "Bit Manipulation Instruction Sets —Wikipedia, the free encyclopaedia", [Online; last accessed 3-Febuary-2020] [Online]. Available: manipulations instructions eats.***

4. ***K. Saanvi and D. A. Patterson, "Instruction sets should be free: The case for risk-v," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, Aug 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/ EECS-2014-46.HTML.***

5. ***Don Kurian Dennis, Ayushi Priyam, Sukhpreet Singh Virk, Sajal Agrawal, Tanuj Sharma, Arijit Mondal, Kailash Chandra Ray." Single cycle RISC-V micro architecture processor and its FPGA prototype", 2017 7th Interna- tonal Symposium on Embedded Computing and System Design (ISED), 2017.***

6. ***Bastian Koppelmann, Peer Adult, Wolfgang Mueller, Christoph Scheft." RISC-V Extension for Bit Manipulation Instructions", 2019 29th Inter- national Symposium on Power and Timing Modelling, Optimization and Simulation (PATMOS), 2019.***

7. ***A. Traber et al., "Pulping: A small single-core risk-v soc," in 2016 RISC- V Workshop, 2016.***

8. ***A. Menon et al., "Shakti-t: A risk-v processor with light weight security extensions," in Proceedings of the Hardware and Architectural Support for Security and Privacy, ser. HASP '17. New York, NY, USA: ACM, 2017.***

9. ***Sugg H. Jingming H. Summonsing K. Oh "Bit Manipulation Accel- orator for Communication Systems Digital Signal Processor" in EURASIP Journal on Applied Signal Processing 2005:16, 2655–2663.***

10. ***GS Tomar, Marcus George, "Modified Binary Multiplier Architecture to Achieve Reduced Latency and Hardware Utilization", Wireless Personal Communication, 2018, Vol.98, No.4, pp.3554-3561***