High Performance Data Level Parallelism based Instruction Set Architecture in RISC-V

C Sri Vardhan,
Department of ECE,
IIITDM Kurnool - 518007, India.
Email: 121ec0018@iiitk.ac.in

G Joel Israel,
Department of ECE,
IIITDM Kurnool - 518007, India.
Email: 121ec0032@iiitk.ac.in

Mohamed Asan Basiri M,
Department of ECE,
IIITDM Kurnool - 518007, India.
Email: asan@iiitk.ac.in

2023 IEEE 7th Conference on Information and Communication Technology (CICT) | 979-8-3503-0517-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/CICT59886.2023.10455642

***Abstract*—Performance is an indispensable factor in processes related to computer architecture. Data Level Parallelism (DLP) is useful to operate on multiple data streams under a single instruction multiple data (SIMD) instructions for improving the performance of the operations on the data. It has a great scope in computing huge chunks of data in an accelerated time, thus finding its applications in scientific calculators, big data processors, and fast computing applications. In this paper, we have proposed an instruction set architecture (ISA) in reduced instruction set computing (RISC)-V using GEM5 that provides faster computational results than the conventional instruction set. Karatsuba algorithm for the 64×64 bit multiplication can be carried out in several ways including thirty-two 16×8 bit multiplications, sixty-four 8×8 bit multiplications, sixteen 32×8 bit multiplications, and so on. When all the above computations are executed together there is a decrease of 71.55% latency with the proposed implementation in comparison to the conventional implementation. The latency of eight 8-bit additions is decreased by 22.31%, four 8×8 multiplications is decreased by 11.34% and for one 8×8 bit multiplication and six 8 bit additions, latency is reduced by 19.98%.**

Index Terms- Data Level Parallelism, GEM5, Instruction Level Parallelism, RISC-V ISA, Superscalar, VLIW

I. INTRODUCTION

Performance is one of the best quantification mechanisms to describe a process. When the term performance is used in computer architecture, it is mainly related to the execution time of the process. The performance of a process is inversely proportional to its execution time. There are several ways to achieve high performance and one such way is the exploitation of parallelism. In a conventional method, processes are executed one after another, just like the queue in a departmental store. From the name parallelism itself, it is understood that multiple processes are being done simultaneously in parallel, thus reducing the overall execution time. There are different types of parallelism such as bit-level parallelism, instruction level parallelism, thread level parallelism, and data level parallelism.

Instruction-level parallelism [1] conveys the idea of executing multiple instructions in parallel. An instruction can either be dependent or independent. Based on the dependency of the instruction, the execution of the instructions can be overlapped. For example, in a process involving two instructions, the fetching of one instruction and the decoding of another instruction can be done in the same clock cycle. This type of parallelism introduces the concept of pipelining [2], in which the system is partitioned into multiple stages, with added buffers that store the computations of the previous stages between the stages. Also, both the addition and multiplicaiton instructions can be processed in parallel using instruction level parallelsim (ILP) when the adder and multiplier of the data path are idle in Very Large Scale Instruction Word (VLIW) [3] or superscalar processors [4].

Data level parallelism [5] is one of the methods of parallelism in which multiple data streams can be operated under a single instruction. This reduces the execution time of the instruction in comparison to the conventional method, in which multiple instructions are used to execute multiple data streams. In vector processors [6], multiple functional units are used to enhance the performance of an instruction. This method is used in multimedia extensions and graphics processing units.

According to the Amdahl's Law, the gain in the improvement of the performance of the computer using an enhancement is limited by the duration of the enhancement exploited.

$$Performance \propto \frac{1}{Execution\,Time}$$

$$Speedup = \frac{Performance\,(Using\,Enhancement)}{Performance\,(Without\,Enhancemnt)}$$

$$Speedup = \frac{ExecutionTime\,(Without\,Enhancement)}{ExecutionTime\,(With\,Enhancement)}$$

In this work, we use the instructions are of type of RV64 of RISC-V processor [7], where the lengths of the instruction and data register are 32-bit and 64-bit respectively. Each instruction can hold only two 64-bit source registers (rs1 and rs2) and one 64-bit destination register (rd1). The instruction format of RV64 of RISC-V is shown in Fig. 1. In the existing works [8] and [9], the custom instructions for convolution neural network and Module-LWE applications are defined in RISC-V architecture. In the similar way, we do define our custom instructions for the data level parallelism based additions and multiplications.

The rest of the paper is organized as follows. Section II explains the proposed DLP based ISA of RISC-V. The implementation details and results are shown in Section III followed by the conclusion in Section IV.

979-8-3503-0517-3/23/$31.00 ©2023 IEEE

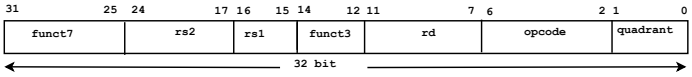

Fig. 1. The format of RV64 instruction of RISC-V

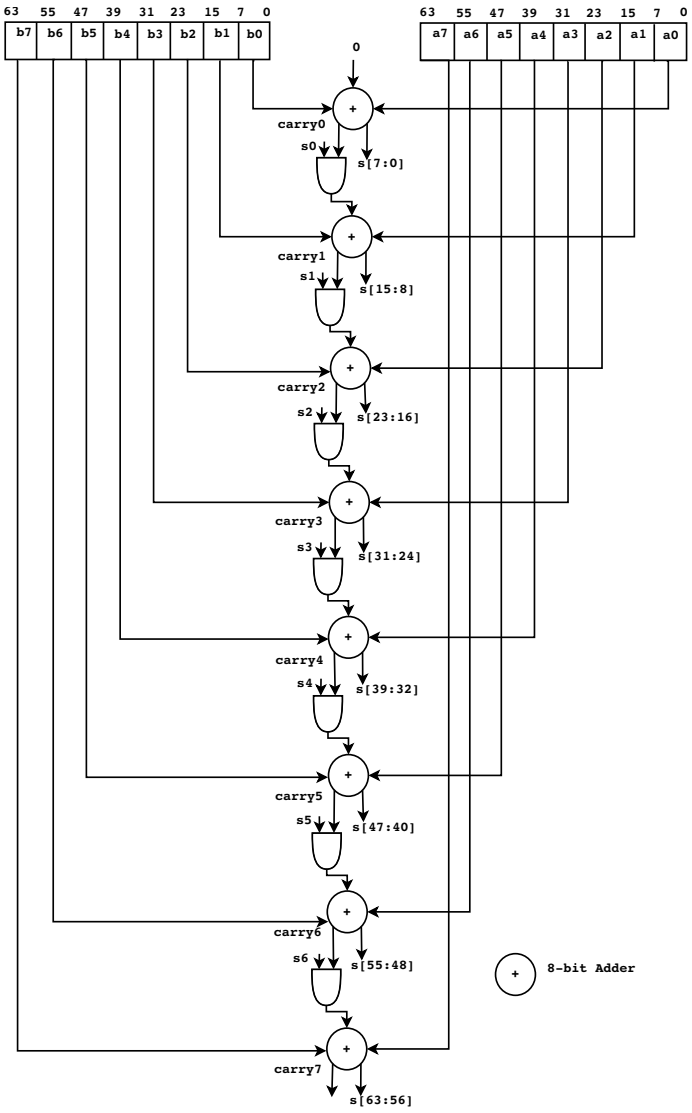

Fig. 2. Implementation of 64-bit adder to perform mutliple additions

II. The proposed DLP based ISA of RISC-V

Fig. 2 shows the implementation of 64-bit adder [10] to perform mutiple additions, where two 64-bit inputs are used. The control lines $s0$, $s1$, ...$s6$ are used to perform eight 8-bit additions OR four 16-bit additions OR two 32-bit additions OR one 64-bit addition. If $s0 = s1 = ..s6 = 0$, then eight 8-bit additions can be performed in parallel. If $s0 = s2 = s4 = s6 = 1$ and $s1 = s3 = s5 = 0$, then four 16-bit additons can be performed in parallel. If $s0 = s1 = s2 = s4 = s5 = s6 = 1$ and $s3 = 0$, then two 32-bit additions can be performed in parallel. If $s0 = s1 = ..s6 = 1$, then one 64-bit addition can be performed. Fig. 3 shows the

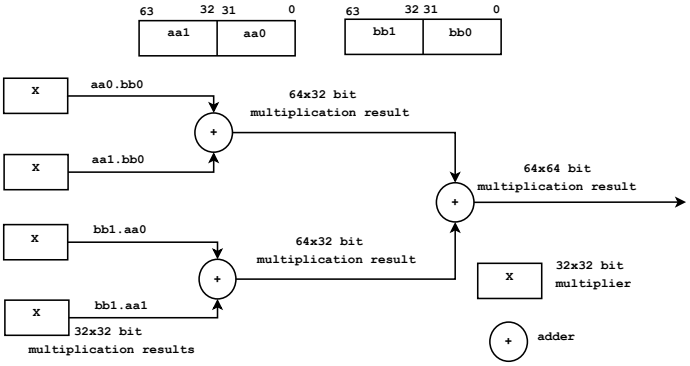

Fig. 3. Implementation of 64×64 bit multiplier to perform mutliple multiplications

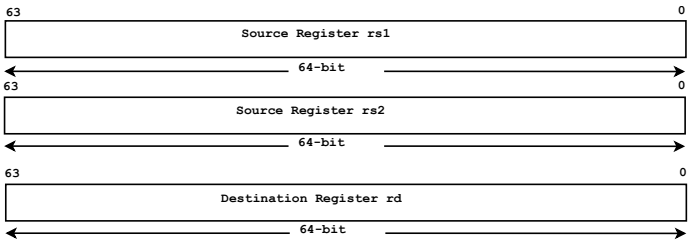

Fig. 4. Instruction to perform one 64-bit addition

implementation of 64×64 bit multiplier [11] [12] to perform mutiple multiplications, where four 32×32 bit OR two 64×32 bit OR one 64×64 bit multiplications can be performed using Karatsuba algorithm [13]. This algorithm is based on the divide and conquer technique and is particularly useful when dealing with large numbers (n-bit). The conventional way of multiplying two large n-bit numbers has a time complexity of $O(n^2)$. However, the Karatsuba algorithm divides the two n-bit numbers into four n/2-bit numbers, resulting in four n/2×n/2 bit multiplications instead of a single n×n bit multiplication. The ability to perform multiple operations in Fig. 2 and 3 gives the motivation to propose the DLP based instructions of RISC-V.

In Fig. 4, there are two 64-bit source registers containing 64-bit data each and one 64-bit destination register. Both the 64-bit data are added using binary addition and the addition result is given to the destination register. This is the normal functioning of an addition instruction. But the addition of four

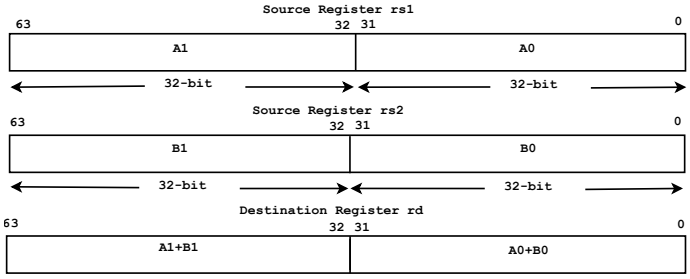

Fig. 5. The proposed instruction to perform two 32-bit additions

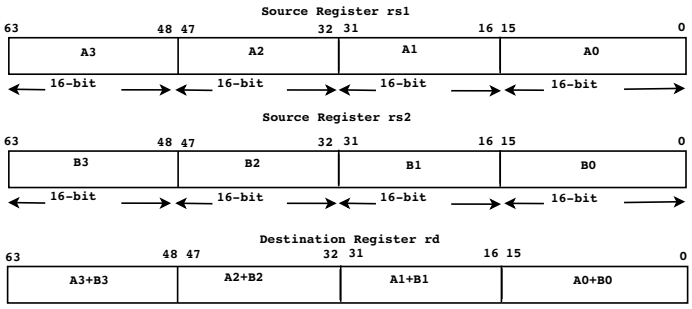

Fig. 6. The proposed instruction to perform four 16-bit additions

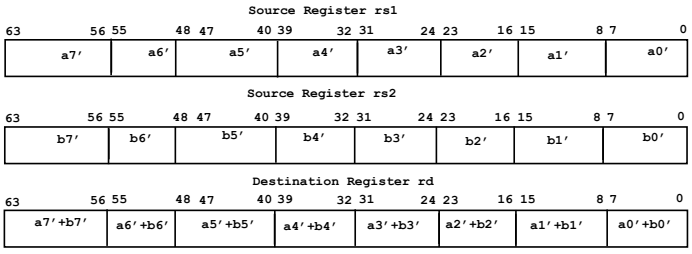

Fig. 7. The proposed instruction to perform eight 8-bit additions

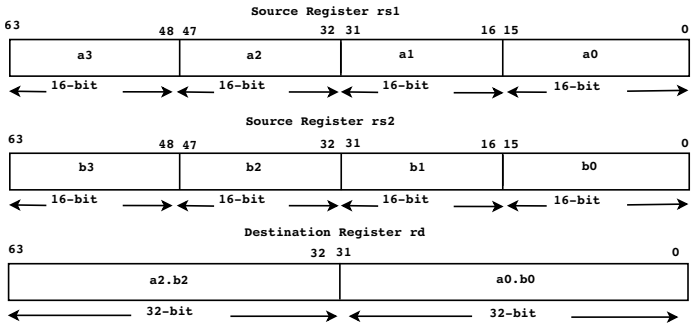

Fig. 9. The proposed instruction to perform two 16×16 bit multiplications

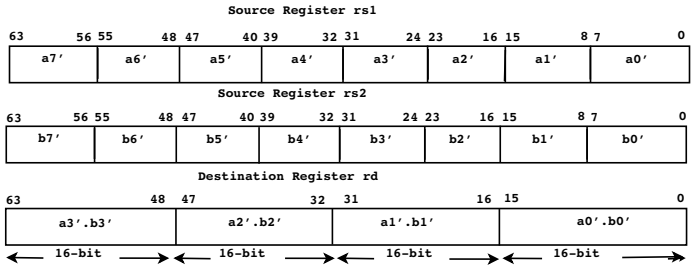

Fig. 10. The proposed instruction to perform four 8×8 bit multiplications

32-bit numbers requires four 32-bit numbers stored in four 64-bit registers incurring a loss of 32-bits in each register and it requires two addition instructions. To avoid such loss of data, two 32-bit numbers are declared in a single source register of size 64-bit and four 32-bit data are operated under a single addition instruction as shown in Fig. 5. In the similar way, four 16-bit additions and eight 8-bit additions can be performed in single instruction as shown in Fig. 6 and 7 respectively.

In a similar fashion, two 32×32 bit multiplications are done using two 64-bit registers storing two 32-bit numbers each and two instructions. Here, the first instruction is to store the first 64-bit result of the binary multiplication and the second instruction is to store another 64-bit result of the binary multiplication as shown in Fig. 8. Fig. 9 shows the implementation of two 16×16 bit multiplications using one instruction, where both the 32-bit results a0.b0 and a2.b2 can be observed from the destination register of the instruction. In the conventional case, we need two different instructions to achieve this with the loss of 48 bits in each of the source registers. In the same way, four 8×8 bit multiplications can be performed using single instruciton as shown in Fig. 10, whereas the conventional case needs two instructions to do the same. Fig. 11 shows the implementation of two 16×8 bit multiplications and one 8×8 bit multiplication using single instruction, whereas the conventional case needs three instructions to do the same. Fig. 12 shows the implementation of one 32×8 bit multiplication and one 8×8 bit multiplication using single instruction, whereas the conventional case needs two instructions to do the same. Fig. 13 shows the implementation of one 8×8 bit multiplication and six 8-bit additions using single instruction, whereas the conventional case needs seven instructions to do the same. Fig. 14 shows the implementation of one 8×8 bit multiplication and three 16-bit additions using single instruction, whereas the conventional case needs four instructions to do the same. Fig. 15 shows the implementation of one 16×8 bit multiplication and five 8-bit additions using

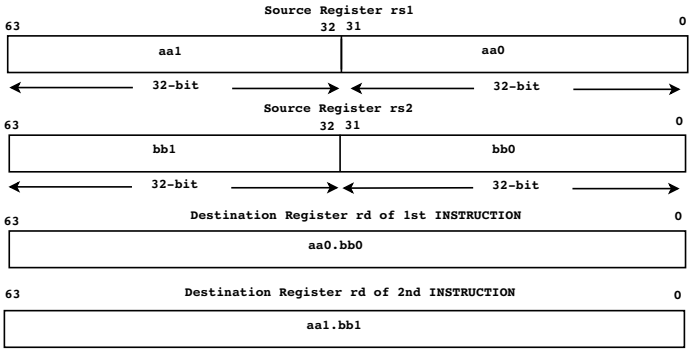

Fig. 8. Instruction to perform two 32×32 bit multiplications

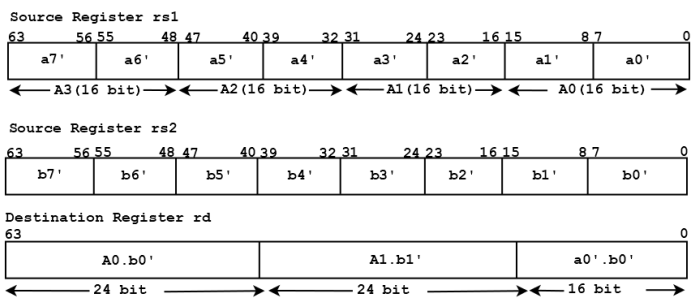

Fig. 11. The proposed instruction to perform two 16×8 bit multiplications and one 8×8 bit multiplication

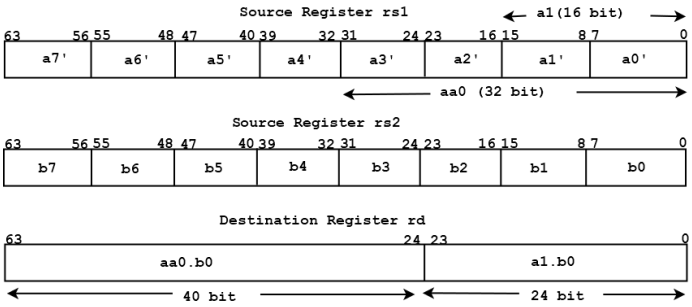

Fig. 12. The proposed instruction to perform one 32×8 bit multiplication and one 16×8 bit multiplication

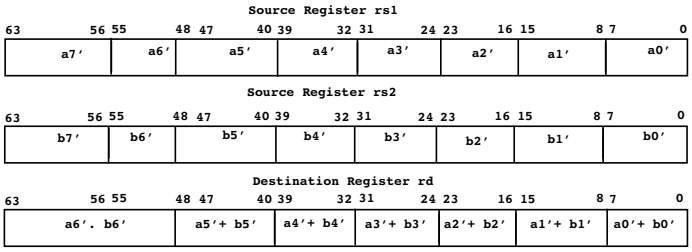

Fig. 13. The proposed instruction to perform one 8×8 bit multiplication and six 8-bit additions

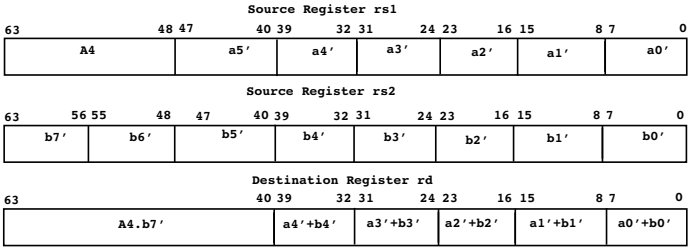

Fig. 15. The proposed instruction to perform one 16×8 bit multiplication and five 8-bit additions

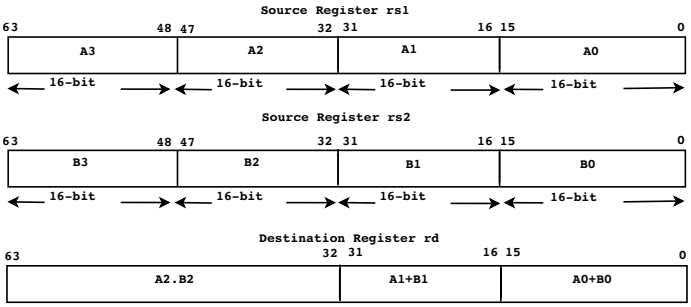

Fig. 16. The proposed instruction to perform one 16×16 bit multiplication and two 16-bit additions

single instruction, whereas the conventional case needs six instructions to do the same. Fig. 16 shows the implementation of one 16×16 bit multiplication and two 16-bit additions using single instruction, whereas the conventional case needs three instructions to do the same. Fig. 17 shows the implementation of one 16×16 bit multiplication and four 8-bit additions using single instruction, whereas the conventional case needs five instructions to do the same.

III. IMPLEMENTATION DETAILS AND RESULTS

Table I shows the synthesis results of adders and multipliers using 45 nm CMOS technology with cadence. These results clearly show that the area and power dissipation of the DLP based adder and multiplier are greater than the conventional designs due to flexibility. Also, Table I shows the power delay product (PDP) [14] of various designs. Here, the PDP is caluclated by the multiplication of total power and delay. The total power is found with the addition of switching and leakage

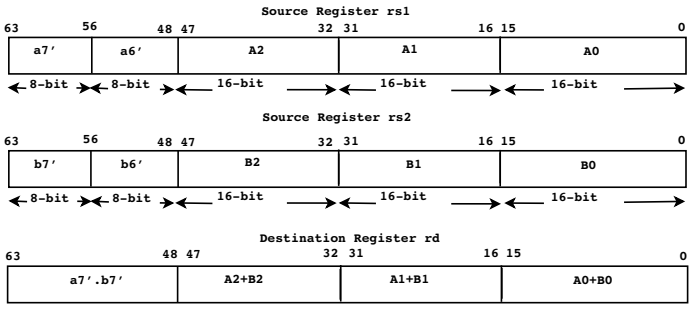

Fig. 14. The proposed instruction to perform one 8×8 bit multiplication and three 16-bit additions

power dissipations. For the implementation of the proposed instructions stated in the previous section, RISC-V ISA is used and the latency is measured using a tool called GEM5. GEM5 is an open-source tool used in computer architecture. There are two types of encoding of an instruction. One is variable-length encoding and the other is fixed-length encoding. A complex Instruction Set Computer (CISC) has variable-length instructions, which encode much information per instruction. In this computer, the main memory is considered as premium, and the decoding of the instruction is filled with complexity. A Reduced Instruction Set Computer (RISC) uses fixed-length instructions, whose decoding is simple. This computer uses processor registers extensively. Therefore in this computer architecture, register-based addressing is used. RISC-V is an ISA that employs RISC type of instructions. To add a custom instruction in GEM5, the following file must be modified according to the instruction breakdown of that particular instruction *gem5/src/arch/riscv/isa/decoder.isa*. The newly added instructions are compiled into binaries using cross-compilers. To measure the latency of each implementation, RISC-V Timing Simple CPU is used in the GEM5 simulator and also the processor is designed in such a way that there is no cache memory.

The latency for the proposed instructions is enlisted in Table II. From Table II, the number of instructions used for one 64-bit addition, one 64×64 bit multiplication, and two 32×32 bit multiplication is the same in both the proposed and conventional implementations. So there is no much change in latency for these operations in both the proposed and conventional implementations. Whereas in the case of two 32×32 bit multiplications, there is a change in the latency because

TABLE I
SYNTHESIS RESULTS OF ADDERS AND MULTIPLIERS USING 45 nm CMOS TECHNOLOGY WITH CADENCE

Design	Delay (ps)	Area (μm^2)	Number of Cells	Leakage power (nW)	Switching Power (nW)	PDP (fJ)
64 bit Conventional Adder	2765	326.6	65	20.61	12748.88	35.31
64 bit DLP based Adder (Fig. 2)	2693	352.99	100	22.17	12944.12	34.92
64 bit Conventional Multiplier	5827	17686.19	6141	1103.36	2194439.38	12793.43
64 bit DLP based Multiplier (Fig. 3)	5837	20655	7176	1159.01	2130772.37	1244.08

TABLE II
COMPARISON OF NUMBER OF INSTRUCTIONS AND LATENCY BETWEEN THE CONVENTIONAL AND DLP BASED PROPOSED ISA OF RISC-V

Type of operation	**Operation**	**Proposed Method (DLP)**		**Conventional Method**	
		Number of Instructions	**Latency (ps)**	**Number of Instructions**	**Latency (ps)**
Addition	**(1) ONE 64-bit addition**	1	18196000	1	18196000
	(2) TWO 32-bit additions	1	18147000	2	18845000
	(3) FOUR 16-bit additions	1	18147000	4	20497000
	(4) EIGHT 8-bit additions	1	18147000	8	23358000
Multiplication	**(5) ONE 64x64-bit multiplication**	2	18657000	2	18657000
	(6) TWO 32x32-bit multiplications	2	18265000	2	20133000
	(7) TWO 16x16-bit multiplications	1	18147000	2	18579000
	(8) FOUR 8x8-bit multiplications	1	18147000	4	20468000
	(9) TWO 16x8-bit multiplications & ONE 8x8-bit multiplication	1	18147000	3	19636000
	(10) ONE 32x8-bit multiplication & ONE 16x8-bit multiplication	1	18147000	2	18579000
Multiplication and Addition	**(11) ONE 8x8-bit multiplication & SIX 8-bit additions**	1	18147000	7	22674000
	(12) ONE 8x8-bit multiplication & THREE 16-bit additions	1	18147000	4	20825000
	(13) ONE 16x8-bit multiplication & FIVE 8-bit additions	1	18147000	6	22246000
	(14) ONE 16x16-bit multiplication & TWO 16-bit additions	1	18147000	3	19966000
	(15) ONE 16x16-bit multiplication & FOUR 8-bit additions	1	18147000	5	21604000

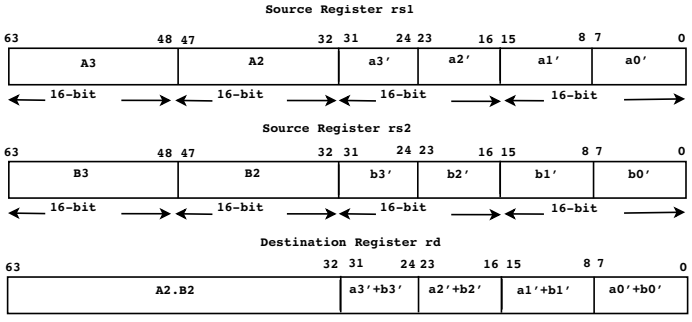

Fig. 17. The proposed instruction to perform one 16×16 bit multiplication and four 8-bit additions

only two 64-bit registers containing four 32-bit numbers are used in the case of the proposed implementation, and four 64-bit registers containing four 32-bit numbers are used in the case of the conventional implementation, incurring a 32-bit data loss in each register. Similarly, for eight 8-bit additions, all the sixteen 8-bit numbers are stored in the two 64-bit source registers, each storing eight 8-bit numbers and these two source registers are operated under a single instruction, obeying the principle of SIMD. When the same is executed by the conventional method of implementation, sixteen 64-bit source registers and eight instructions are required, which is accountable for more execution time. By utilizing the proposed instructions DLP, there is a decrease in the latency of eight 8-bit additions by 22.31%, a decrease in the latency of eight 8×8 bit multiplication by 11.34% and for one 8×8 bit multiplication and six 8 bit additions, latency is decreased by 19.98%. Karatsuba algorithm for the 64×64 bit multiplication can be carried out in several ways including thirty-two 16×8 bit multiplications, sixty-four 8×8 bit multiplications, sixteen 32×8 bit multiplications, and so on. When all the above computations are executed together there is a decrease of 71.55% latency with the proposed implementation in comparison to the conventional implementation. Fig. 18 shows the comparison of latency between the conventional and DLP based proposed ISAs of RISC-V, where the all the ten operations mentioned in Table II are represented in the x-axis.

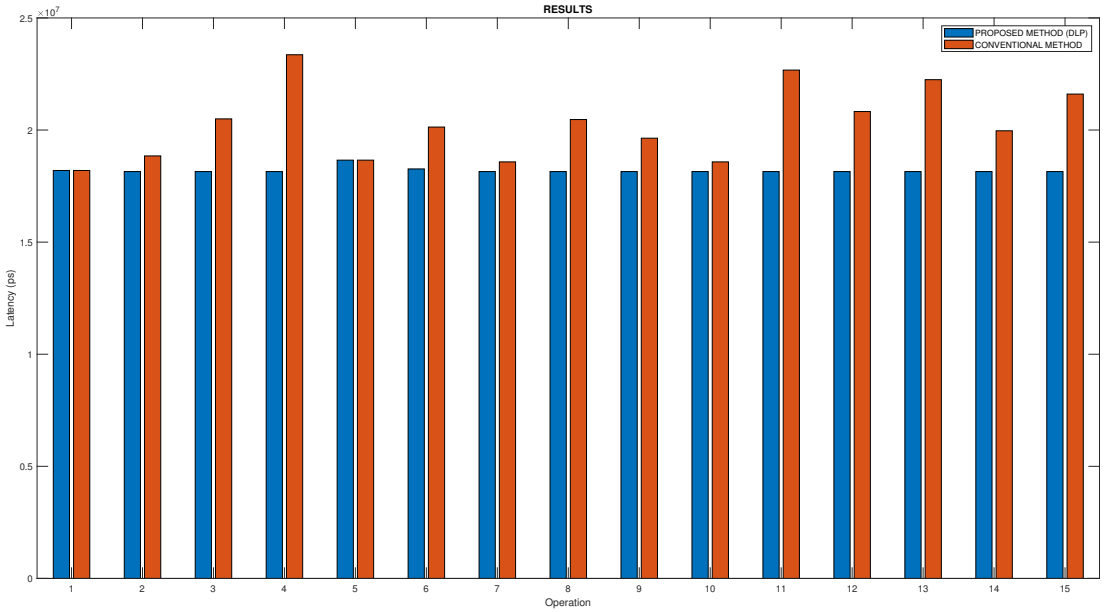

Fig. 18. Comparison of latency between the conventional and DLP based proposed ISA of RISC-V

IV. CONCLUSION

The DLP offers great scope in the field of computer architecture as it reduces the execution time of a process, thereby enhancing the performance of a processor. It is very much useful to compute operations on huge chunks of data under a single instruction, owing to a faster computational time, thus finding its application in Scientific Calculators and and Big Data Processors. For example the computation of sixteen 8×8 bit multiplications can be done using only two 64-bit registers instead of thirty-two 64-bit registers, thus producing a faster computational result. We have exploited this DLP in our proposed ISA using RISC-V to reduce the latency. The simulation of the proposed instructions is done with GEM5 compiler and the latency analysis shows that the proposed implementation achieves significant improvement than the conventional implementation.

V. ACKNOWLEDGMENTS

This work is sponsored by MeitY, GoI as a part of Category-III of C2S research project titled, "Instruction-Data level Parallelism based Hardware Accelerator Design in HPC and CPS Applications". We thank MeitY for the support.

REFERENCES

[1] D. A. Patterson and J. L. Hennessy, "*Computer Architecture: A Quantitative Approach*", MK Publications, 5th Edition, pp. 262-334, 2012.

[2] Carl Harmacher, Zvonko Vranesic, and Safwat Zaky, "*Computer Organization*", McGraw Hill Publications, pp. 454-506, 2002

[3] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri, "*A VLIW Processor with Reconfigurable Instruction Set for Embedded Applications*", IEEE Journal of Solid-State Circuits, vol. 38, no. 11, pp. 1876-1886, 2003.

[4] D.M. Cardoso, R. Tonetto, M. Brandalero, G. Nazar, A.C. Beck, and J.R. Azambuja, "*Exploring the Limitations of Dataflow SIHFT Techniques in Out-of-order Superscalar Processors*", Microelectronics Reliability, Elsevier, vol. 100-101, Sep. 2009.

[5] T. Komuro, S. Kagami, and M. Ishikawa, "*A Dynamically Reconfigurable SIMD Processor for a Vision Chip*", IEEE Journal of Solid-State Circuits, vol. 39, no. 1, pp. 265-268, Jan. 2004.

[6] C.E. Kozyrakis and D.A. Patterson, "*Scalable, Vector Processors for Embedded Systems*", IEEE Micro, vol. 23, no. 6, pp. 36-45, Dec. 2003.

[7] Andrew Waterman and Krste Asanovic, "*The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2*", University of California, pp. 1-130, May 2017.

[8] Wenqi Lou, Chao Wang, Lei Gong, and Xuehai Zhou, "*RV-CNN: Flexible and efficient instruction set for CNNs based on RISC-V processors*", International Symposium on Advanced Parallel Processing Technologies, Springer, pp. 3-14, Aug. 2019.

[9] Yifan Zhao, Ruiqi Xie, Guozhu Xin, and Jun Han, "*A High-Performance Domain-Specific Processor With Matrix Extension of RISC-V for Module-LWE Applications*", IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 69, no. 7, pp. 2871-2884, July 2022.

[10] M. Moris Mano, "*Digital Logic And Computer Design*", Pearson Publications, 2nd Edition, pp. 436-447, 2004.

[11] Mohamed Asan Basiri M, Samaresh Chandra Nayak, and Noor Mahammad Sk, "*Multiplication Acceleration Through Quarter Precision Wallace Tree Multiplier*", IEEE International conference on signal processing and integrated networks (SPIN), pp. 502-505, Feb. 2014.

[12] Mohamed Asan Basiri M and Noor Mahammad Sk, "*Memory Based Multiplier Design in Custom and FPGA implementation*", International Symposium on Advances in Intelligent Systems and Computing, Springer, vol. 320, pp. 253-265, Sep. 2014.

[13] Eyupoglu C, "*Performance Analysis of Karatsuba Multiplication Algorithm for Different Bit Lengths*", Procedia - Social and Behavioral Sciences, vol. 195, pp. 1860–1864, July 2015.

[14] Ricardo Gonzalez, Benjamin M. Gordon, and Mark A. Horowitz, "*Supply and Threshold Voltage Scaling for Low Power CMOS*", IEEE Journal of Solid State Circuits, vol. 32, no. 8, pp. 1210-1216, 1997.