Design of Extended RISC-V for Q-Learning Hardware Accelerator using HW/SW Co-Design

Infall Syafalni[‡§*], Muhammad Sulthan Mazaya[¶], Muhammad Raihan Elfazri[‡],
Eko Mursito Budi[¶], Nana Sutisna[‡§], Rahmat Mulyawan[‡§], Trio Adiono[‡§], and Makoto Ikeda[*†]
[‡]School of Electrical Engineering and Informatics, Bandung Institute of Technology, Indonesia
[§]University Center of Excellence on Microelectronics, Bandung Institute of Technology, Indonesia
[¶]Faculty of Industrial Technology, Bandung Institute of Technology, Indonesia
[*]System Design Laboratory, School of Engineering, The University of Tokyo, Tokyo, Japan
[†]Graduate School of Engineering, The University of Tokyo, Tokyo, Japan

2024 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS) | 979-8-3503-7877-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/APCCAS62602.2024.10808529

***Abstract*—In this paper, we propose a novel RISC-V ISA extension for a reinforcement learning-based hardware accelerator called RISC-Q. We introduce new instructions *i.e.,* $q.store\_constant$, $q.store$, $q.load$, $q.max$, and $q.update$. Each instruction corresponds to a hardware module. The $q.store\_constant$ sets necessary constants for the Q-learning by storing it in dedicated registers. The $q.store$ and $q.load$ instructions write and read the data to the main memory. The $q.max$ instruction controls the max seeker hardware module to find Q-max values. The $q.update$ instruction accesses the Q-updater hardware module to update the Q-value with the corresponding state, action, and reward. Experimental result shows that the RISC-V extended reinforcement learning hardware accelerator (RISC-Q), which is implemented in the Nexys FPGA board, outperforms the software implementation up to 1.5× speed up. The work is useful for edge-computing applications such as smart navigation, smart communications, and robotics.**

I. Introduction

Reinforcement Learning (RL) is one of the Machine Learning (ML) approaches used to train an entity, called an agent, to accomplish a certain task [1]. Applications of this approach have been increasing in many ways, such as crypto trading [2], autonomous driving [3], health [4], and even gaming [5]. Such applications often involve complex computations, such as large-scale matrix operations, neural network computations, and iterative calculations. One way such heavy tasks can be efficiently performed is by using a hardware accelerator. Many hardware technologies can be used in accelerating machine learning (and particularly reinforcement learning) algorithms, such as Graphics Processing Unit (GPU) and Field-Programmable Gate Array (FPGA). In particular, FPGA is a promising technology due to its low power consumption, reconfigurability, and real-time processing capability [6], [7].

On the other hand, an open-source processor *i.e.,* RISC-V has been a popular selection for both researchers and engineers in developing efficient computing platforms [8]–[11]. The work in [8] proposes a low-power IoT RISC-V processor with a hybrid encryption accelerator. The encryption accelerator is controlled by using custom instructions (ISA) for the RISC-V processor. Meanwhile, the work in [9] proposes an extension of RISC-V in a 256-bit word RISC-V for dynamically scheduled instructions. The use of multi-core RISC-V is also described in [10]. Finally, the work in [11] implements RISC-V ISA extensions for arithmetic operations. RISC-V ISA is known for its flexibility. Thus, research on implementing RISC-V extended ISA by exploiting unused opcodes for various applications is still an interesting research area.

In this paper, we propose a novel RISC-V ISA extension for a reinforcement learning-based hardware accelerator. We introduce new instructions *i.e.,* $q.store\_constant$, $q.store$, $q.load$, $q.max$, and $q.update$. Each instruction corresponds to a hardware module. The $q.store\_constant$ sets necessary constants for the Q-learning by storing it in dedicated registers. The $q.store$ and $q.load$ instructions write and read the data to the main memory. The $q.max$ instruction controls the max seeker hardware module to find Q-max values. The $q.update$ instruction accesses the Q-updater hardware module to update the Q-value with the corresponding state, action, and reward. Experimental result shows that the RISC-V extended reinforcement learning hardware accelerator (RISC-Q), which is implemented in the Nexys FPGA board, outperforms the software implementation up to 1.5× speed up with more than 0.4 instructions per cycle. The work is useful for edge-computing applications such as smart navigation, smart communications, and robotics.

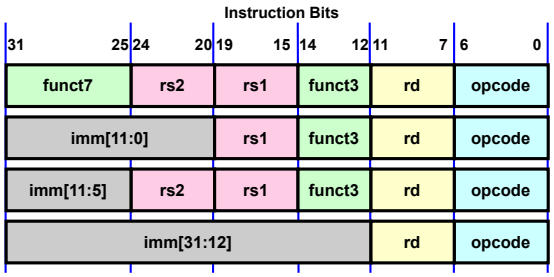

Fig. 1: Instruction Format For Each Instruction Type

The paper is organized as follows: Section I introduces the work. Section II shows the basic properties and definitions for the proposed method. Section III explains the design of an extended RISC-V reinforcement learning hardware accelerator (RISV-Q). Finally, Section IV shows the experimental results and Section V concludes the work.

II. Basic Definitions and Properties

A. RISC-V ISA and Its Extension

RISC-V is an open standard instruction set architecture based on established reduced instruction set computer principles and it is a royalty-free open-source license. Thus, it is very suitable for researchers and engineers to innovate in various cutting-edge projects using RISC-V.

Figure 1 shows several RISC-V base instructions; 1) Register/Register (R-Type), 2) Immediate (I-Type), 3) Store (S-Type), and 4) Upper Immediate (U-Type) [12]. The sources $rs1$ and $rs2$ as well as the destination rd are in 5-bit format for accessing the 16 registers. They are allocated to the same position. The $funct7$ and the $func3$ are used to specify the operation to be done in the RISC-V. Finally, the immediate value is stored in the imm field. In this work, we modify the R-Type instruction to control our extended reinforcement learning accelerator. Lemma 2.1 shows the number of ISA extensions for an R-Type instruction. In this case, for an opcode, we can extend the ISA up to $128 \times 8 = 1024$.

Lemma 2.1: **ISA Extension.** Suppose that there is an opcode in the R-Type instruction that is available for an ISA extension. If we have a 7-bit $funct7$ and a 3-bit $funct3$. The number of extended ISA is at most $2^{n_7} \times 2^{n_3}$, where n_7 is the number of bits of $funct7$ and n_3 is the number of bits of $funct3$.

B. Reinforcement Learning

A human does a learning process by observing the environment and evaluating the environment based on the rewards he/she takes from the environment. This concept is adopted to a branch of machine learning called **reinforcement learning (RL/Q-learning)**. By the rewards, the human, or in RL we call an agent, calculates the pros and cons of an action by remembering them in the brain (or memory). The pros and cons of an action are summarized by a Q-value.

1) Q-Update Calculation: To calculate the Q-value, we use the Bellman equation. This approach is derived from a well-known dynamic programming optimization:

Lemma 2.2: **Q-Update.** Let α, γ, $s[t]$, $a[t]$, $r[t]$ be the learning rate, the discount factor, the current state, the action, and the reward from the given action $a[t]$, respectively. The calculation of a Q-value is:

$$Q[s[t],a[t]] = Q[s[t],a[t]] + \alpha(r[t] + \gamma(\max_a(Q[s[t+1]] - Q[s[t],a[t]]))),$$

where $Q[s[t],a[t]]$ is the current Q-value in state $s[t]$, and $\max_a(Q[s[t+1]] - Q[s[t],a[t]]))$ maximizes the effects of the given action $a[t]$ with the reward $r[t]$.

2) State, Q-Table, and Reward Definitions: In this subsection, we define the state, the Q-value, and the reward for the shortest path problem in a maze environment.

Definition 2.1: **State Definition.** Suppose that we have a maze with $N = w \times l$ grids. The array of state for the shortest path problem is $S = [s[0], s[1], \ldots, s[N-1]]$.

Note that the state is represented by a one-dimensional array. In the case of the shortest path problem, a state represents the location of the agent in the maze environment. Figure 2 shows the illustration of the maze environment and its Q-table representation. The maze environment consists of width w and length l. Thus, the number of total mazes is N. From this, we can define the Q-table representation.

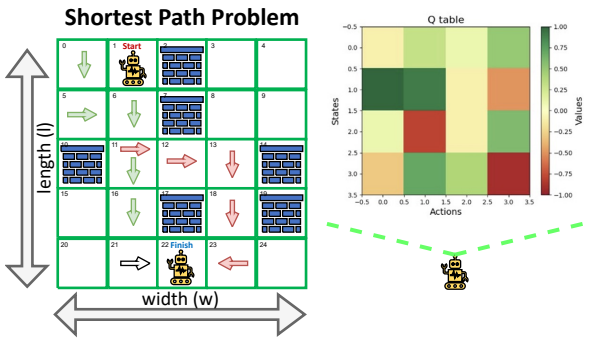

Fig. 2: Shortest Path Problem and Its Q-Table Representation

Definition 2.2: **Q-Table Representation.** The Q-table is represented as a function of a state and an action $Q(s[t], a[t])$. To store each Q-value that corresponds to a state and an action, it requires $N \times N_a$, where N is the size of the maze environment and N_a is the number of actions.

Lemma 2.3: **Reward for the Q-Learning.** Let $s[t]$ and $s[t+1]$ be the current and the next state, respectively.

$$r[t] = \begin{cases} r_w, & \text{if the agent meets an obstacle} s[t+1] \\ r_g, & \text{if } s[t+1] \text{is goal state,} \\ r_v, & \text{otherwise,} \end{cases}$$

where r_w is the obstacle reward (large negative value), r_g is the goal reward (large positive value), and r_v is the visit reward (small negative value).

III. Proposed Extended RISC-V for Q-Learning Hardware Accelerator Using HW/SW Co-Design

The processor used for this paper is RISC-V VeeR EL2 Core. We extend the Execute/Memory stage of the RISC-V to support several functions in the Q-learning algorithm in the form of specific hardware modules.

Figure 3 shows the proposed extended RISC-V for the Q-learning hardware accelerator unit. The accelerator has several main components such as a constant register hardware unit, Q-table updater hardware unit, max-seeker hardware unit, and Q-table memory hardware unit.

A. Proposed HW/SW Co-Design Partition

We propose a partition for dividing the Q-learning into hardware (highlighted in blue) and software processing (highlighted in red) as shown in Alg. 1. The hardware part accelerates the instruction using the extended RISC-V for the Q-learning hardware accelerator, while the software part uses regular RISC-V stages that are already provided by RISC-V VeeR EL2 Core. The RISC-V execute or memory stage is added with a Q-learning accelerator. Finally, we introduce custom instructions as shown in Table I.

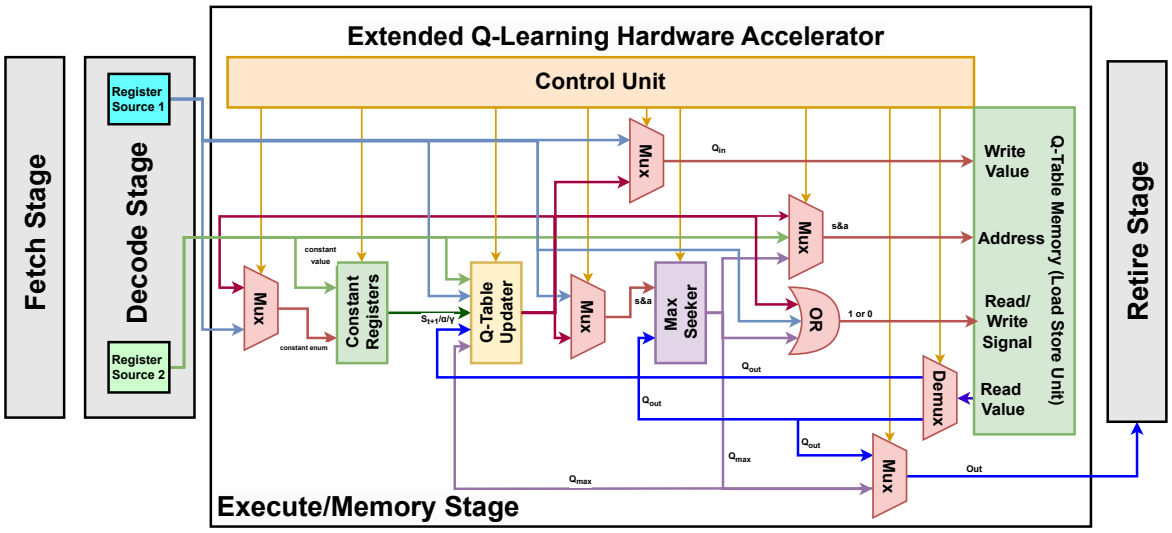

Fig. 3: Proposed Extended RISC-V Based Q-Learning Hardware Accelerator Unit

TABLE I: Custom Instruction Formats for Extended RISC-V

funct7	opcode	ISA
0000000	1010011	q.store
0001000	1010011	q.load
0001100	1010011	q.store_constant
0001110	1010011	q.max
0001111	1010011	q.update

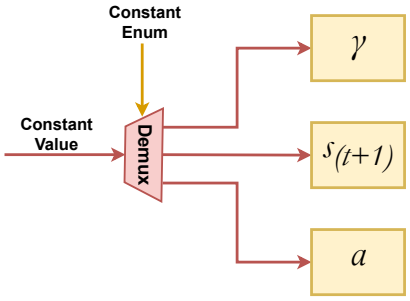

Fig. 4: Constant Registers Hardware Unit

1) Q-Table Memory Hardware Unit: First, the $q.store$ (Alg. 1, line 1) is used to initialize each Q-value by accessing the Q-table memory as shown in Figure 3. The size of the memory follows Definition 2.2. The values of the memory (Q-value) will be updated as the agent explores or exploits the environment by the corresponding state $s[t]$, action $a[t]$, and reward $r[t]$.

B. Constant Register Hardware Unit

Second, the $q.store\_constant$ (Alg. 1, line 2) is used to store the α and γ values as shown in Figure 4. It consists of a demultiplexer that chooses three registers to be filled by the values of γ, $s[t+1]$, and α.

C. Max Seeker Hardware Unit

Next, the $q.max$ instruction (Alg. 1, line 6) is used for performing Q-value comparisons. The hardware module compares

Algorithm 1: RISC-Q HW/SW Co-Design Algorithm

Input: The learning rate is α, the discount factor is γ and the total number of episodes is L.
Output: The Q-Table is stored in the Q-Table Memory
begin
- **[Hardware]** Initialize each Q-value $Q(s[t], a[t])$.
- **[Hardware]** Store α and γ in the constant registers.
- **while** $k < L$ **do**
  - **[Software]** $s[t] \leftarrow s[0]$.
  - **while** $s[t] \neq s[N]$ **do**
    - **[Hardware]** Choose an action $a[t]$ for the state $s[t]$.
    - **[Software]** Do the action $a[t]$, get the reward $r[t]$, and move to the next state $s[t+1]$.
    - **[Hardware]** Store $s[t+1]$ at the accelerator.
    - **[Hardware]** Update the current Q-value $Q(s[t], a[t])$.
    - **[Software]** $s[t] \leftarrow s[t+1]$.
  - **[Software]** $k \leftarrow k+1$ and update the cumulative reward $C_r[k]$.
- **[Software]** Return the Q-Table.

the Q-values $Q(s[t], a[t])$ by performing $\max Q(s[t+1], a)$ for all actions as shown in Figure 3.

For flexibility, the reward calculation (Lemma 2.3) uses the RISC-V software process (Alg. 1, line 7). And, here is values of the reward $r[t]$ depending on the environment: 1) Hitting the wall: -1, 2) Out of bounds: -1, 3) Retracing traversed path: -0.2, 4) Traversing new tile: -0.01, 5) Reaching the goal: 1. In

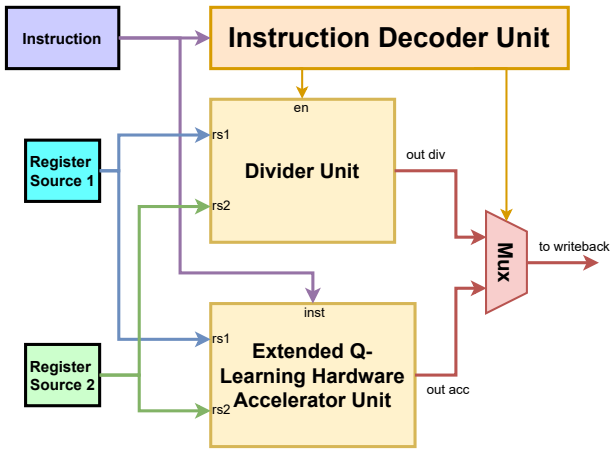

Fig. 5: Extended Controller for Q-Learning HW Accelerator

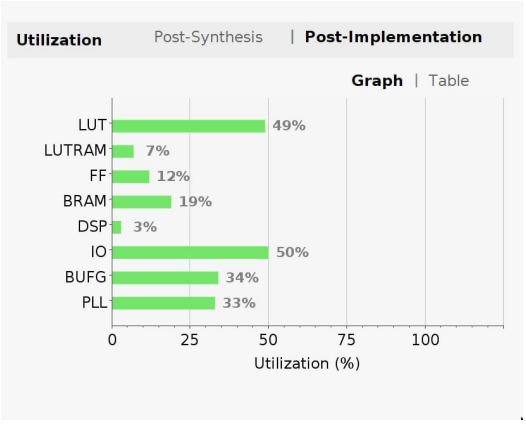

Fig. 6: Resource Utilization of RISC-V Q-Learning Hardware Accelerator

the Alg. 1, line 8, the $q.store\_constant$ is also used to store the $s[t+1]$ using the hardware part.

D. Q-Updater Hardware Unit

The Q-updater hardware unit calculates the Bellman equation as Lemma 2.2.

Lemma 3.1: **Reconstructed Q-Update.** The Bellman equation for finding updated $Q[s[t], a[t]]$ can be rewrite as:

$$(1-\alpha)Q[s[t],a[t]] + \alpha(r[t] + \gamma \max_{a}(Q[s[t+1]])).$$

The Q-updater calculates the Lemma 3.1 in several steps; step 1) $(1-\alpha)$, step 2) loading $Q(s[t], a[t])$ using $q.load$, step 3) $(1-\alpha)Q(s[t], a[t])$ and $\max Q(s[t+1], a)$ concurrently, step 4) $\gamma \max Q(s[t+1], a)$, step 5) $r[t] + \gamma \max Q(s[t+1], a)$, step 6) $\alpha(r[t] + \gamma \max Q(s[t+1], a))$, and step 7) the whole Lemma 3.1.

E. Extended Instruction Decoder Unit

Finally, we locate the extended RISC-V hardware unit for the Q-learning at the out-of-pipe divider module in VeeR EL2 Core. The new instructions are passed to the extended RISC-V core together with register source 1 ($rs1$) and register source 2 ($rs2$). The output of the extended core is multiplexed with the divider unit.

IV. Experimental Results

In this section, we show some experimental results regarding our proposed extended RISC-V for a Q-learning hardware accelerator. The experiments are implemented in the Nexys AMD (previously Xilinx) FPGA board. First of all, we implement the new instructions in the execution module.

TABLE II: Hardware Accelerator Performance Comparison

	RISC-Q(T.W.)	RISC-V	Improve
Cycles	24393528	36846206	1.51×
Instructions	9889007	16180437	1.63×
IPC	0.40539470141	0.43913441183	0.92×

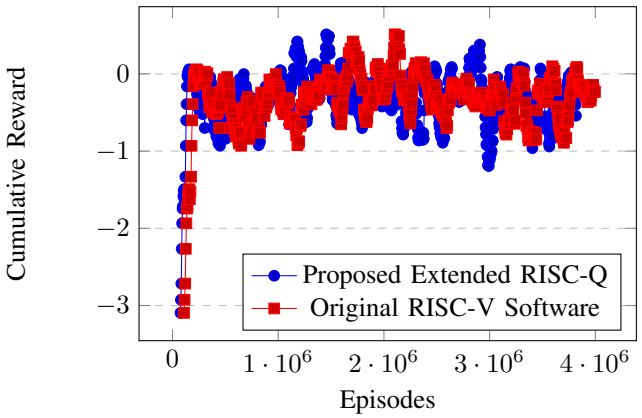

Fig. 7: Cumulative Reward Performances for Both Extended RISC-V (proposed) and Original RISC-V Software

Next, in Table II, we compare the instruction per cycle of the extended RISC-V core with the original RISC-V software implementation. As shown, our proposed processor has a smaller number of instructions with 9889007 (40% smaller than the original RISC-V implementation). Figure 6 shows the resource utilization. Finally, Figure 7 confirms the functionality of the Q-learning on the implemented processor.

V. Conclusion

In this paper, we proposed a novel RISC-V ISA extension for a reinforcement learning-based hardware accelerator (RISC-Q). We introduced new instructions *i.e.,* $q.store\_constant$, $q.store$, $q.load$, $q.max$, and $q.update$. Each instruction corresponds to a hardware module. Experimental results showed that the RISC-V extended reinforcement learning hardware accelerator (RISC-Q), which is implemented in the Nexys FPGA board, outperforms the original RISC-V software implementation up to 1.5× speed up. The work is useful for edge-computing applications such as smart navigation, smart communications, and robotics.

ACKNOWLEDGEMENT

This work is supported by 2023 International Research Program provided by LPPM, Bandung Institute of Technology under grant no. [LPPM.PN-16-22-2023].

REFERENCES

[1] R. S. Sutton, F. Bach, and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press Ltd, 2018.

[2] G. Borrageiro, N. Firoozye, and P. Barucca, "The recurrent reinforcement learning crypto agent," *IEEE Access*, vol. 10, pp. 38590–38599, 2022.

[3] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2022.

[4] K.-L. A. Yau, Y.-W. Chong, X. Fan, C. Wu, Y. Saleem, and P.-C. Lim, "Reinforcement learning models and algorithms for diabetes management," *IEEE Access*, vol. 11, pp. 28391–28415, 2023.

[5] K. Shao, Y. Zhu, and D. Zhao, "Starcraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 3, no. 1, pp. 73–84, 2019.

[6] N. Sutisna, A. M. R. Ilmy, I. Syafalni, R. Mulyawan, and T. Adiono, "Farane-q: Fast parallel and pipeline q-learning accelerator for configurable reinforcement learning soc," *IEEE Access*, vol. 11, pp. 144–161, 2023.

[7] I. Syafalni, M. I. Firdaus, A. M. Riyadhus Ilmy, N. Sutisna, and T. Adiono, "Mazecov-q: An efficient maze-based reinforcement learning accelerator for coverage," in *2023 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, pp. 01–06, 2023.

[8] S. Yang, L. Shao, J. Huang, and W. Zou, "Design and implementation of low-power iot risc-v processor with hybrid encryption accelerator," *Electronics*, vol. 12, p. 4222, Oct 2023.

[9] N. M. Qui, C. H. Lin, and P. Chen, "Design and implementation of a 256-bit risc-v-based dynamically scheduled very long instruction word on fpga," *IEEE Access*, vol. 8, pp. 172996–173007, 2020.

[10] A. Kamaleldin, S. Hesham, and D. Göhringer, "Towards a modular risc-v based many-core architecture for fpga accelerators," *IEEE Access*, vol. 8, pp. 148812–148826, 2020.

[11] M. Reichenbach, J. Knödtel, S. Rachuj, and D. Fey, "Risc-v3: A risc-v compatible cpu with a data path based on redundant number systems," *IEEE Access*, vol. 9, pp. 43684–43700, 2021.

[12] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual, volume i: User-level isa, version 2.0," Tech. Rep. UCB/EECS-2014-54, May 2014.