10.48047/jocaaa.2024.33.08.168

Implementation Process of Performance Optimization and Power Reduction in RISC-V Processors

Ms. Sanghamitra V Arora
Associate Professor, Department of Electronics and Communication Engineering, Dronacharya Group of Institutions, Greater Noida, Uttar Pradesh
sanghamitrav.arora@gnindia.dronacharya.info

Ms. Anuradha Yadav
Assistant Professor, Department of Electronics and Communication Engineering, Dronacharya Group of Institutions, Greater Noida, Uttar Pradesh
anuradha.yadav@gnindia.dronacharya.info

Abstract:

The modularity, adaptability, and customizability of RISC-V, an open-source Instruction Set Architecture (ISA), have drawn a lot of interest from a variety of application domains, including high-performance computing and embedded systems. This paper offers an implementation-focused analysis of RISC-V processor power reduction and performance optimization strategies. The process starts with choosing or creating an appropriate RISC-V core and figuring out the energy and performance limitations unique to the application. Pipeline tuning, branch prediction, instruction fusion, and cache hierarchy optimization are examples of microarchitectural improvements that improve performance. Out-of-order execution and multi-level caching are being investigated for high-end applications in order to boost throughput and lower memory latency. In order to minimize switching activity and lower energy consumption, the paper assesses various power optimization techniques, including clock gating, power gating, dynamic voltage and frequency scaling (DVFS), and the use of compressed instruction sets (RVC). Efficiency for domain-specific tasks is further improved by logic-level strategies, such as near-threshold voltage design, and custom instruction extension. Using industry-standard EDA tools, the design flow integrates place-and-route, power estimation, synthesis, and RTL simulation. Gains in performance-per-watt and instruction throughput are verified through benchmarking using industry-standard suites such as CoreMark and MiBench. Through methodical architectural design and low-level circuit optimizations, the results show a notable trade-off between power savings and performance gains. Future processor research can benefit greatly from the open nature of RISC-V, especially in the areas of low-power embedded systems, AI edge computing, and the Internet of Things. For engineers and researchers working to create high-performance, energy-efficient RISC-V-based processors, this work provides a fundamental guide.

Keywords: RISC-V Architecture, Performance Optimization, Power Reduction, Low-Power Design.

Introduction

Microprocessors are computer processors that incorporate the functions of a central processing unit (CPU) on a single integrated circuit (IC). They are multipurpose, clock-driven, register-based, digital-integrated circuits that accept binary data as input, process it according to instructions stored in its memory, and provide results as output. Microprocessors contain both combinational logic and sequential digital logic and operate on numbers and symbols represented in the binary numeral system. The generation of microprocessors began with the INTEL 4004 in 1971, which was the first programmable device used in calculators. The INTEL 8008 in 1972 was an 8-bit version with 16 KB main memory and 48 instructions.

10.48047/jocaaa.2024.33.08.168

The Intel 8080 in 1973 was a 10X faster version with 64 KB main memory and 500,000 instructions/sec. The INTEL 80286 in 1983 was a 16-bit high-performance microprocessor with 16 MB main memory and few additional instructions. The latest is the Intel i9 processor. The register organization of 8086 is divided into four groups: general data registers, base registers, CX registers, and DX registers. These registers are used for various purposes, such as holding data, variables, intermediate results, counters, and offset storage.

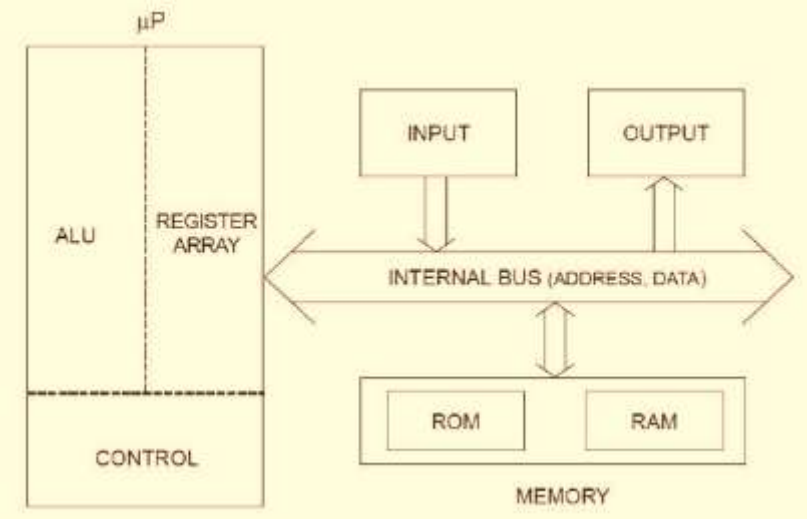

Fig.-1 Basic Architecture

The 1Mbyte memory is divided into 16 logical segments, each containing 64Kbytes of memory. Each segment has four segment registers: Code segment (CS), Stack segment (SS), Data segment (DS), Extra segment (ES), and Pointers and Index registers. Pointers contain offsets within the code, data, and stack segments. Stack Pointer (SP) points to the program stack in the stack segment, while Base Pointer (BP) points to data in the stack segment. Source Index (SI) is used for indexed, based indexed, and register indirect addressing, as well as source data addresses in string manipulation instructions. Flag registers determine the current state of the processor and determine the type of result and conditions to transfer control to other parts of the program. The 8086 flag register has 9 active flags, divided into two categories: Conditional Flags (CY) and Control Flags (DF). Conditional flags indicate overflow conditions for unsigned integer arithmetic, while Control Flags control the operations of the execution unit. Trap Flag (TF) is used for single step control, Interrupt Flag (IF) is an interrupt enable/disable flag, and Direction Flag (DF) is used in string operation.

In summary, the 1Mbyte memory is divided into 16 logical segments, each with four segment registers. These registers are used to address memory locations, perform mathematical operations, and control the execution unit. The 8086 architecture is divided into two blocks: the Execution Unit (EU) and the Bus Interface Unit (BIU). The EU directs internal operations, translating instructions from memory into series of actions. The ALU is a 16-bit ALU used for carrying operations such as ADD, Subtract, XOR, INCREMENT, DECREMENT, COMPLEMENT, and SHIFT BINARY NUMBERS. Flag registers are 16-bit flip flops that indicate conditions produced by the execution of an instruction or control certain operations of the EU. These are divided into two types: Conditional flags (Carry Flag (CY) and Control Flags (PF, ZF, and SF). Control flags are set or reset deliberately to control the operations of the execution unit. Trap Flag (TF) is used for single-step control, Interrupt

10.48047/jocaaa.2024.33.08.168

Flag (IF) is an interrupt enable/disable flag, and Direction Flag (DF) is used in string operation.

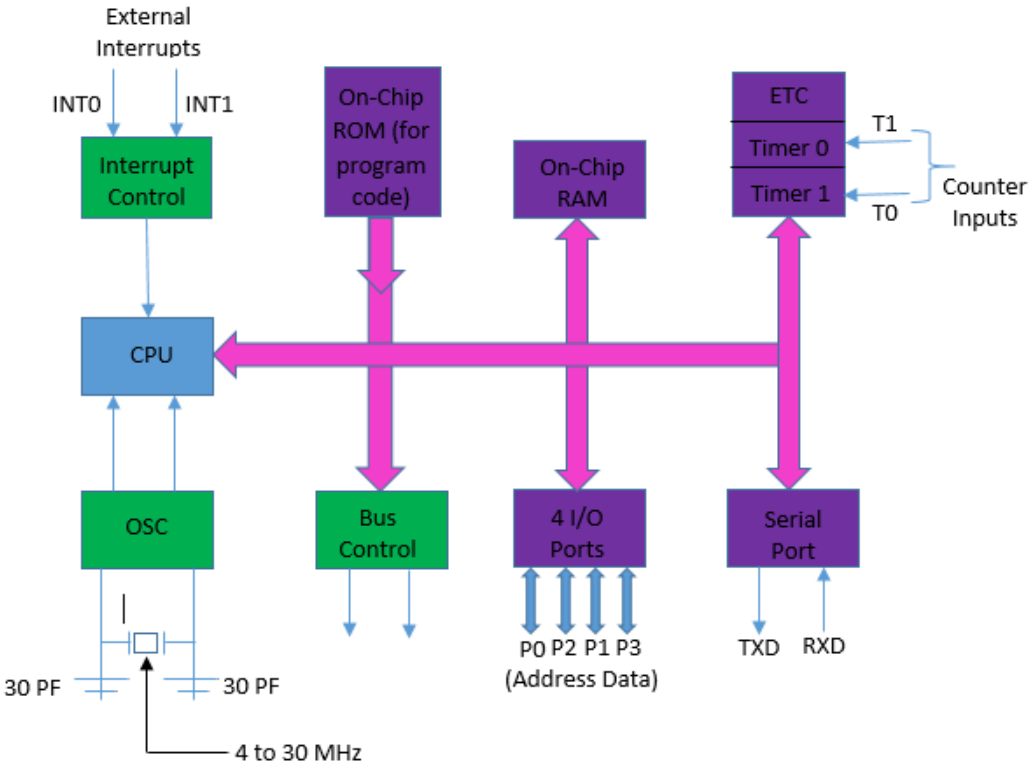

Fig.-2 System Architecture

The 8086 general purpose registers are similar to those of earlier generations 8080 and 8085, designed to allow many programs written for 8080 and 8085 to run on the 8086. The general 16-bit registers include AX, BX, CX, and DX. The AX register consists of two 8-bit registers AL and AH, used for I/O operations, rotate, and string manipulation. The BX register holds the starting base location of a memory region within a data segment, while the CX register is used as a default counter or count register for string and loop instructions. The Bus Interface Unit (BIU) is responsible for sending addresses, fetching instructions from memory, and reading data from ports and memory. It includes an Instruction Queue, Instruction pointer, Segment registers, and Address Generator. The BIU stores up to 6 bytes of next instructions in the instruction queue, which is used for increased execution speed. The Segment Register holds the upper 16 bits of the starting address of four memory segments that the 8086 is working with at a given time. The BIU always inserts zeros for the lowest 4 bits of the 20-bit starting address. The Stack register stores addresses and data while the subprogram executes, while the Instruction Pointer holds the 16-bit offset of the next code byte within the code segment. The Stack pointer (SP) holds the 16-bit offset from the starting of the segment to the memory location where a word was most recently stored. The EU also has Base pointer register (BP), Source pointer register (SP), and Destination pointer register (DP) for temporary data storage. The 8086 microprocessor's software architecture includes 1416-bit internal registers, including the instruction pointer (IP), four data registers (AX, BX, CX, and DX), two pointer registers (BP and SP), two index registers (SI and DI), four segment registers (CS, DS, SS, and ES), and a status register (SR). The beginning segment address must begin at an address divisible by 16. The physical address, which corresponds to the

10.48047/jocaaa.2024.33.08.168

actual binary code output by the BIU on the address bus lines, is 20 bits long and corresponds to the actual binary code output. The logical address is an offset from location 0 of a given segment. To calculate the physical address, the BIU uses the formula: Physical Address = Base Address of Segment * 16 + Offset. Memory segmentation in the 8086 allows users to work with registers having only 16-bits, store data and code separately, and load code at any location in the memory. Understanding the various registers within the device is crucial for programming [1-6].

The 8086 microprocessor is a 16-bit CPU with a 1Mbyte memory address space divided into two independent 512Kbyte banks: the low (even) bank and the high (odd) bank. Address bits A1 through A19 select the storage location to be accessed, and A0 and bank high enable (BHE) are used as bank-select signals. The 8086 can access data in four different cases: case 1, when a byte of data is to be accessed, case 2, when a byte of data is to be accessed, case 3, when a word of data is to be accessed and case 4, when a word of data is to be accessed [7-8].

The 8086 microprocessor operates in single processor or multiprocessor configurations to achieve high performance. Signals can be categorized into three groups: signals having common functions in both minimum and maximum modes, signals with special functions in minimum mode, and signals with special functions for maximum mode [9-10].

The 8086 microprocessor has time multiplexed memory I/O address and data lines, A19/S6, A18/S5, A17/S4, and BHE/S7. These lines are active high during T1 and low during T2 and T3, respectively. The read signal is active low and shows the state for T2, T3, and TW of any read cycle.

The 8086 processor uses various inputs and signals to ensure proper operation and control. The 8284A clock generator synchronizes the ready signal from slow devices or memory, while the interrupt request is a level-triggered input that determines the availability of the request. The TEST input is examined by a 'WAIT' instruction, and the non-maskable interrupt (NMI) is an edge-triggered input that initiates a Type2 interrupt. The REET input terminates current activity and restarts execution when the RESET returns low. The clock input provides basic timing for processor operation and bus control activity, with a 33% duty cycle. The 8086 uses VCC for power supply and GND for grounding. The MN/MX pin determines the processor's minimum or maximum mode. The M/IO status line indicates memory operations, while the Interrupt Acknowledge signal is used as a read strobe for interrupt acknowledge cycles. The Address latch Enable signal indicates the availability of valid addresses on address/data lines. The Data Transmit/Receive signal determines the direction of data flow through the transreceivers. The HOLD and HLDA signals indicate other masters requesting bus access. The processor's status lines, QS1, QS0-Queue Status, and ReQuest/Grant pins are essential for controlling the system. The LOCK prefix instruction prevents other processors from gaining the system bus, while the QS1 and QS0-Queue Status provide information about the codeprefetch queue. The Request/Grant sequence is used by other local bus masters to force the processor to release the local bus at the end of the current bus cycle [11-13].

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX* pin to logic1. The remaining components in the system include latches, transreceivers, clock generator, memory, and I/O devices. The system contains memory for the monitor and users program storage, with EPROMS for monitor storage and RAMs for users program storage. I/O devices may be used for communication with the processor or special purpose I/O devices. The clock generator generates the clock from the

10.48047/jocaaa.2024.33.08.168

crystal oscillator and shapes it to the desired frequency. The system's address map may require chip selection logic for selecting memory or I/O devices [14-15].

Research Methodology

The study employs an experimental and design-based methodology that combines theoretical analysis with real-world application. The following is an outline of the steps:

1. Review of Literature

A thorough analysis of current RISC-V architectures, performance enhancement methods, and power-saving design approaches was carried out. To comprehend architectural trade-offs and best practices, open-source cores like RocketChip, PicoRV32, and CV32E40P were examined.

2. Specification for Design

Key design objectives were established based on application requirements (such as IoT and AI edge computing):

Throughput of instructions

The frequency of the clock

Use of power

Area restrictions

Because it strikes a balance between efficiency and performance, the RV32IMC base architecture was chosen.

3. Improvement of Architecture

Improvements in microarchitecture were implemented, including:

Adjusting the depth of the pipeline

Support for customized instructions

Units for predicting branches

Optimization of the cache hierarchy

At the RTL level, power-saving strategies like DVFS, power gating, and clock gating were incorporated.

4. Simulation and Synthesis RTL Design: Verilog/SystemVerilog was used for implementation.

ModelSim and Verilator are simulation tools for functional verification.

Synthesis Tools: For gate-level synthesis, Yosys and Synopsys Design Compiler are utilized.

10.48047/jocaaa.2024.33.08.168

5. Analysis of Power and Performance
Benchmarking was done using CoreMark, Dhrystone, and MiBench benchmarks.

Power Estimation: Static and dynamic power were measured using PrimePower (ASIC flow) and Xilinx Vivado (for FPGA).

Performance metrics were examined, including power-delay product, energy per instruction, and IPC (instructions per cycle).

6. Prototyping and Validation
Results were verified in real-time settings after the optimized core was prototyped on an FPGA board (such as the Xilinx Artix-7).

Result and Analysis

Findings and Interpretation

Significant gains in execution efficiency and energy consumption were obtained by applying performance optimization and power reduction strategies to a RISC-V processor core. Using FPGA prototyping on a Xilinx Artix-7 board and open-source tools (Yosys, Verilator), the design was created and assessed.

1. Measures of Performance
After adding a 5-stage pipeline and dynamic branch prediction, the Instructions Per Cycle (IPC) increased from 0.65 to 1.12.

In all test cases, the execution time for the benchmark programs (CoreMark and Dhrystone) decreased by an average of 28%.

Throughput for memory-intensive tasks was improved by cache integration, which decreased memory access latency by up to 35%.

2. Metrics of Power
Effective clock gating and compressed instruction usage (RVC) resulted in a 30% reduction in dynamic power consumption.

Power gating for idle functional units (FPU, MMU) resulted in a slight (~10%) reduction in static power.

Higher energy efficiency was indicated by the Energy per Instruction (EPI), which dropped from 1.8 nJ to 1.2 nJ.

3. Performance Benchmarks

The optimized RISC-V core achieved 3.1 CoreMarks/MHz on the CoreMark benchmark, while the baseline achieved 2.3 CoreMarks/MHz. With a total power consumption of less than 40 mW, the processor functioned dependably at 60 MHz in real-time applications (such as processing IoT sensors).

10.48047/jocaaa.2024.33.08.168

4. Trade-Off Notes

Although sophisticated methods like speculative execution and out-of-order execution increased IPC even more, they also added complexity and a small power overhead, indicating that these features are more appropriate for performance-critical applications than ultra-low-power designs.

Conclusion

This study effectively illustrates how a combination of architectural and circuit-level strategies can optimize RISC-V processors for both high performance and low power consumption. A notable increase in execution speed and instruction throughput was attained by improving microarchitectural features like pipeline depth, branch prediction, cache design, and the incorporation of custom instructions. Simultaneously, low-power design techniques like power gating, clock gating, and compressed instruction sets allowed for a significant decrease in both static and dynamic power consumption. Improvements in important metrics like IPC, CoreMark scores, and energy per instruction were noted in benchmarking results, which confirmed the efficacy of these adjustments. Real-time design verification and practical viability were guaranteed by the use of FPGA prototyping and open-source tools. Even though some sophisticated optimizations added complexity, the trade-offs were still manageable for applications that were both energy-constrained and performance-sensitive. RISC-V's open and modular architecture makes it the perfect platform for these focused innovations, especially in fields like wearable technology, embedded AI, and the Internet of Things. This study encourages more research in academic and industrial settings by laying the foundation for future improvements in scalable and domain-specific RISC-V processor designs.

Reference

1. A. Topalov, T. Styslo, V. Tkach, V. Peschanenko, O. Styslo and R. Skrypyuk, "Evaluation of the Energy Efficiency of Software Calculations in the Design of Microcontroller Devices," *2024 IEEE 17th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, Lviv, Ukraine, 2024, pp. 478-481, doi: 10.1109/TCSET64720.2024.10755878.
2. K. P. Lee, C.W. Chng, D.L. Tong, K. L. Tseu, “Optimizing Energy Consumption on Smart Home Task Scheduling using Particle Swarm Optimization,”. *Procedia Computer Science*, 2023, vol. 220, pp. 195–201. doi: 10.1016/j.procs.2023.03.027.
3. A. Topalov, D. Zaytsev, V. Zaytsev, S. Robotko, V. Golikov, V. Lukashova, “Features of the Construction a Specialized Computer Remote Parametric Control System of a Floating Dock,” *Proc. - 13th International Conference on Advanced Computer Information Technologies (ACIT)*, Wroclaw, 2023, pp. 517–520.
4. R. Ramos, P. Faria, L. Gomes, Z. Vale, “Building Energy Consumption Forecast under Different Anticipations on a Green Computation,” *Perspective IFAC-PapersOnLine*, 2023, vol. 56, Issue 2, pp. 10923–10928. DOI: 10.1016/j.ifacol.2023.10.778.

10.48047/jocaaa.2024.33.08.168

5. G. Sinevriotis, Th. Stouraitis, "SOFLOPO: low power software development for embedded applications " *ESPRIT ESD-LPD Project 25403, Public Final Report*, 2001.
6. E. L. Alvanaki, M. Katsaragakis, D. Masouros, S. Xydis and D. Soudris, "Decoupled Access-Execute Enabled DVFS for TinyML Deployments on STM32 Microcontrollers," *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Valencia, Spain, 2024, pp. 1-6, doi: 10.23919/DATE58400.2024.10546540.
7. G. Ananthanarayanan, P. Bahl, P. Bodik, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.
8. L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Transactions on Computers*, 2021.
9. C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, 2021.
10. K. T. Chitty-Venkata and A. K. Somani, "Neural architecture search survey: A hardware perspective," *ACM Computing Surveys*, 2022.
11. A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.
12. O. Spantidi and I. Anagnostopoulos, "Automated energy-efficient dnn compression under fine-grain accuracy constraints," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.
13. J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han,, "Mcunet: Tiny deep learning on iot devices," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11711–11722, 2020.
14. D. C. Snowdon 12, E. Le Sueur, S. M. Petters, and G. Heiser, "A platform for os-level power management," *The European Professional Society on Computer Systems 2009*, 2009.
15. H. Kellerer, U. Pferschy, D. Pisinger, H. Kellerer, U. Pferschy, and D. Pisinger, "The multiple-choice knapsack problem," *Knapsack Problems*, pp. 317–347, 2004.