AURORA-15: Design of a Compact 15-Bit Instruction Set Architecture for Low-Power Embedded Processors

VENKATESWARA REDDY KUNDURU
Department of Electronics and Communication Engineering,
M.B.T.S. Government Polytechnic,
Guntur 522005, Andhra Pradesh,
INDIA

ORCID iD: 0000-0002-9567-1063

Abstract: - Compact instruction set architectures are essential for low-power embedded processors and educational CPU design, where hardware simplicity and code density are critical. Traditional RISC architectures such as MIPS and ARM employ wide instruction formats and large register files, which increase decoding complexity, silicon area, and power consumption. This paper presents AURORA-15, a fixed-length 15-bit instruction set architecture designed to achieve reduced hardware overhead while maintaining functional adequacy for embedded applications. The proposed ISA adopts a 28-bit datapath, an accumulator-influenced execution model, and a compact register file of eight registers, including dedicated zero, stack pointer, and link registers. Only two instruction formats are used to simplify decoding and control logic. PC-relative branching and base–offset addressing eliminate the need for complex absolute control hardware, while a partially reserved opcode space enables future extensibility. Functional validation using representative programs confirms correct arithmetic, memory, and control-flow operations. A comparative analysis with a MIPS-style architecture shows improved code density, reduced register file size, and lower decoder complexity. Owing to its minimal design and scalability, AURORA-15 is well suited for FPGA-based soft processors, educational platforms, and resource-constrained embedded systems.

Key-Words: - Instruction Set Architecture (ISA), Embedded Processors, Code Density, Accumulator-Based Architecture, Low-Power CPU Design, Compact Instruction Encoding, FPGA Soft-Core Processor

Received: March 11, 2026. Revised: April 13, 2026. Accepted: May 15, 2026. Published: July 16, 2026.

1 Introduction

The increasing deployment of embedded controllers in domains such as industrial automation, wearable devices, and Internet of Things (IoT) systems has intensified the need for processors that are both energy-efficient and hardware-economical. In these applications, the instruction set architecture (ISA) directly influences datapath complexity, control unit design, memory requirements, and overall silicon utilization. While modern RISC architectures such as MIPS and ARM are optimized for performance and compiler efficiency, their wide instruction formats, multiple addressing modes, and large register files lead to increased decoder complexity and higher implementation cost. Such features are often unnecessary for compact embedded systems and educational processor platforms, where simplicity, predictability, and ease of hardware realization are of primary importance.

Several approaches have been explored to improve code density and reduce architectural overhead, including compressed instruction sets and accumulator-based execution models. Accumulator-oriented designs reduce the number of explicit operands, enabling shorter instruction encodings and simpler ALU input selection logic. Similarly, limiting the number of instruction formats and addressing modes can significantly reduce control signal generation complexity and improve timing predictability.

In this work, a novel compact ISA named AURORA-15 is presented to address these design objectives. The proposed architecture employs a fixed 15-bit instruction format, a 28-bit datapath, and a reduced register file consisting of eight registers. An accumulator-influenced execution model is adopted to minimize operand encoding, while PC-relative branching and base–offset memory access eliminate the need for complex absolute addressing

hardware. The ISA is defined using only two instruction formats, enabling a streamlined decode stage and reduced control logic. Additionally, a partially reserved opcode map is incorporated to support future instruction extensions without modifying existing binary encodings.

The proposed ISA is functionally validated through representative programs demonstrating arithmetic computation, memory operations, and control flow. A comparative architectural analysis indicates that AURORA-15 achieves improved code density, reduced register file size, and simplified decoding logic when compared to a conventional MIPS-style ISA.

The primary contributions of this paper are:

1. The design of a compact fixed-length 15-bit ISA tailored for embedded and educational processors,
2. The integration of an accumulator-based execution model with a minimal register set,
3. A two-format instruction encoding strategy that simplifies control unit implementation, and
4. An analytical comparison highlighting reductions in hardware complexity and instruction memory requirements.

2 Background and Related Works

Instruction set architecture design has evolved significantly from accumulator-based processors to modern register–register RISC architectures. Classical designs such as MIPS emphasize a large uniform register file, fixed 32-bit instruction formats, and multiple instruction types to achieve high performance and compiler optimization efficiency [1]. While this approach simplifies pipelining and improves throughput, it increases instruction memory requirements and decoding complexity, which may not be suitable for resource-constrained embedded systems.

To address code size limitations, several compact ISA variants have been proposed. ARM Thumb and other compressed instruction set reduce instruction width while maintaining compatibility with their base architectures [2]. These designs improve code density but still rely on relatively complex register files and multiple instruction formats, resulting in nontrivial control logic and hardware overhead. Similarly, microcontroller-oriented ISAs often reduce register count and support simplified addressing modes; however, many retain variable-length instructions or extensive opcode maps, which complicate decoding and timing predictability.

Accumulator-based architectures represent an alternative approach focused on minimizing operand fields and simplifying datapath design. Early processors and several educational CPU models adopted accumulator-oriented execution to reduce hardware requirements and instruction encoding size [3]. Although such designs limit instruction-level parallelism, they provide advantages in terms of reduced control complexity, smaller register files, and straightforward ALU interfacing, making them suitable for compact and low-power implementations.

Despite these developments, there remains a gap for a minimal fixed-length ISA that combines a reduced register file, accumulator-influenced execution, limited instruction formats, and a scalable opcode allocation strategy within a compact encoding scheme. The proposed AURORA-15 architecture addresses this gap by introducing a 15-bit fixed instruction format, an eight-register organization, PC-relative control flow, and a partially reserved opcode map, enabling both hardware simplicity and future extensibility.

3 AURORA-15 ISA Design

This section presents the architectural design of the proposed AURORA-15 instruction set architecture, which is developed to achieve a balance between functional capability and hardware simplicity for embedded and educational processor implementations. The ISA is defined with a fixed-length 15-bit instruction format, a 28-bit datapath, and a compact register file consisting of eight registers. An accumulator-influenced execution model is adopted to reduce operand encoding and simplify arithmetic and logical operations. Furthermore, the architecture employs only two instruction formats, limited addressing modes, and PC-relative control flow to minimize decoder and control unit complexity. The opcode space is intentionally underutilized to allow future extensibility without modifying the existing encoding scheme. The following subsections describe the design goals, register organization, instruction formats, instruction set, and memory addressing model of AURORA-15 in detail.

3.1 Design Goals and Architectural Overview

The primary objective of the AURORA-15 instruction set architecture is to provide a compact and hardware-efficient processing platform suitable for low-power embedded systems, FPGA-based soft processors, and educational computing environments. Unlike conventional RISC architectures that emphasize high instruction throughput through wide instruction formats and large register organizations, the proposed architecture adopts a structurally simplified design

methodology focused on minimizing datapath complexity, reducing decoder overhead, and improving instruction compactness while preserving sufficient computational capability for general-purpose integer operations.

AURORA-15 employs a fixed-length 15-bit instruction format to reduce instruction memory utilization and simplify instruction fetch alignment. The compact encoding strategy decreases instruction bandwidth requirements and contributes to lower switching activity during instruction fetch and decode operations. To further reduce operand encoding overhead, the architecture adopts an accumulator-influenced execution model in which the accumulator register participates implicitly in arithmetic and logical operations. This approach eliminates the need for repeatedly encoding multiple source operands within each instruction and significantly simplifies ALU operand selection logic.

The architecture utilizes a 28-bit datapath and address space to maintain uniformity across the register file, arithmetic logic unit, memory interface, and control-flow operations. A reduced eight-register organization is selected to decrease register decoder complexity, interconnect overhead, and control circuitry while maintaining sufficient flexibility for embedded workloads. Dedicated functional registers, including a hardwired zero register, stack pointer, and link register, are incorporated to support efficient arithmetic execution, memory access, and subroutine control without increasing instruction-field complexity.

To minimize control-unit overhead and improve deterministic execution behavior, the ISA defines only two instruction formats covering arithmetic, memory, immediate, and control-flow operations. This limited-format organization substantially reduces combinational decoding complexity and simplifies control signal generation compared to conventional multi-format architectures. In addition, the architecture exclusively employs PC-relative branching and base-offset memory addressing to avoid complex absolute addressing hardware and improve position-independent execution capability.

Another important design consideration is architectural scalability. Instead of fully utilizing the available opcode space, a portion of the encoding map is intentionally reserved to support future instruction-set expansion without modifying existing binary compatibility. This enables the architecture to accommodate future enhancements such as pipelined execution support, specialized arithmetic instructions, or lightweight acceleration features while preserving the simplicity of the current implementation.

Collectively, these architectural decisions enable AURORA-15 to achieve a balanced trade-off between hardware simplicity, compact instruction encoding, deterministic execution behavior, and implementation flexibility, making the proposed ISA well suited for compact embedded computing platforms and instructional processor development.

3.2 Register File Organization

The AURORA-15 architecture employs a compact register organization consisting of eight 28-bit registers aligned with the processor datapath width to maintain uniform data handling across arithmetic, logical, memory, and control-flow operations. The reduced register count is intentionally selected to minimize hardware resources associated with decoder circuitry, read/write port control, register interconnects, and operand selection logic. Compared with conventional 32-register RISC architectures, the proposed organization significantly reduces register addressing overhead and contributes to lower datapath complexity and switching activity.

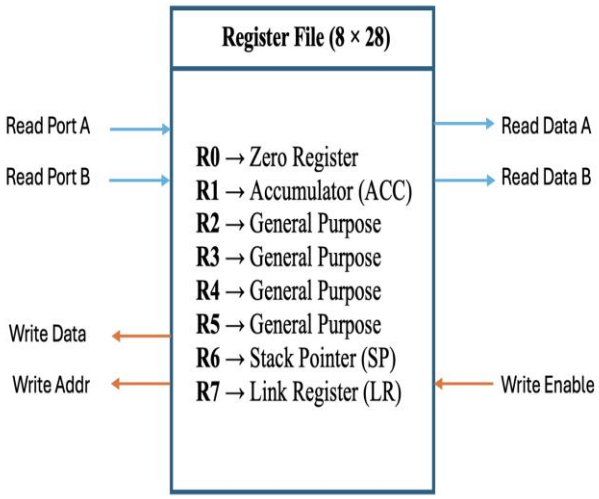

Fig. 1: AURORA-15 Register File Organization

The architectural reduction in register organization can be expressed as:

$$(32-8)/38 \times 100=75\%$$

This reduction decreases register decoder complexity and simplifies operand routing within the datapath, which is particularly beneficial for FPGA-oriented implementations and compact embedded processors.

As illustrated in Figure 1, each register is assigned a dedicated functional role to improve execution efficiency while preserving compact instruction encoding. Register R0 is implemented as a hardwired zero register that continuously provides a constant zero value without requiring immediate operand

generation or additional control logic. Register R1 functions as the accumulator (ACC) and serves as an implicit operand for arithmetic and logical operations. The accumulator-based organization eliminates repeated encoding of one source operand, thereby reducing instruction-field utilization and simplifying ALU input multiplexing.

Registers R2–R5 are used as general-purpose registers for temporary data storage and intermediate computations during program execution. Register R6 operates as the stack pointer (SP) and maintains the top-of-stack address for memory-oriented operations, while Register R7 serves as the link register (LR) for storing return addresses during subroutine execution and branch-link operations. These dedicated functional assignments improve execution organization without increasing opcode or addressing complexity.

The register file supports two simultaneous read ports and one write port, enabling concurrent operand fetch and single-cycle result write-back during instruction execution. This port organization supports deterministic datapath operation while maintaining a low-complexity hardware structure suitable for single-cycle processor implementation. The simplified register organization also reduces register interconnect congestion and contributes to predictable timing behavior within the datapath.

3.3 Instruction Encoding and Formats

AURORA-15 adopts a fixed-length 15-bit instruction encoding strategy to achieve compact program representation and simplified instruction decoding. Each instruction is stored within a 16-bit aligned memory location with one unused alignment bit, enabling uniform instruction fetch and deterministic program counter progression. Unlike variable-length instruction architectures that require complex decoding stages and alignment hardware, the fixed-length organization of AURORA-15 simplifies instruction parsing, reduces decode latency, and improves timing predictability within the processor datapath.

The compact instruction organization of AURORA-15 reduces instruction storage requirements relative to conventional 32-bit RISC architectures. The reduction in instruction width can be expressed as:

$$\frac{(32-15)}{32} \times 100 = 53.125\%$$

This reduction directly decreases instruction memory utilization, fetch bandwidth, and switching activity during instruction fetch and decode operations, making the architecture suitable for compact embedded implementations.

To minimize decoder complexity while preserving functional flexibility, the ISA employs only two instruction formats, namely Format-A and Format-B. The limited number of formats simplifies opcode interpretation and control signal generation, reducing combinational logic overhead in the control unit.

Format-A is primarily used for arithmetic and logical operations. As shown in Figure 2, the instruction consists of a 4-bit opcode field, a 3-bit destination register field, a 3-bit source register field, and a 5-bit function field that specifies the required ALU operation. The accumulator register participates implicitly in arithmetic execution, thereby reducing explicit operand encoding requirements and simplifying ALU operand selection logic.

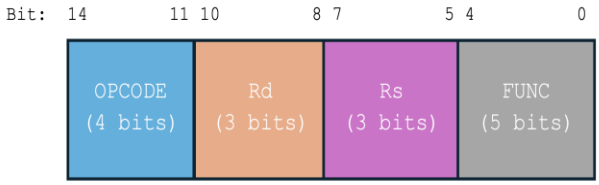

Fig. 2: AURORA-15 Format-A instruction bit-field structure for arithmetic and logical operations

Format-B is used for immediate operations, memory access instructions, and control-flow execution. The format contains a 4-bit opcode field, a 3-bit register field, and an 8-bit immediate or offset field, as illustrated in Figure 3. During execution, the immediate value is sign-extended to 28 bits to maintain datapath uniformity across arithmetic, addressing, and branch computations. This format supports immediate loading, base-offset addressing, and PC-relative branching while preserving compact instruction representation.

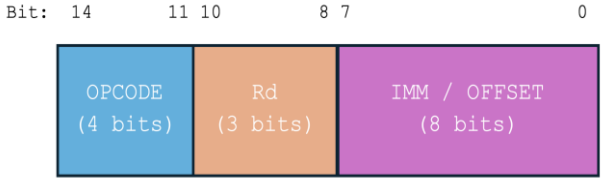

Fig. 3: AURORA-15 Format-B instruction bit-field structure for immediate, memory, and control-flow operations

The use of only two compact instruction formats significantly reduces instruction decoding complexity compared with conventional multi-format architectures. In addition, the reduced operand-field organization improves encoding efficiency and enables streamlined control-unit implementation without sacrificing the functional requirements

necessary for embedded and educational processor applications.

3.4 Instruction Set Architecture

The AURORA-15 instruction set architecture is designed using a compact and function-oriented instruction organization to minimize decoder complexity while maintaining sufficient computational capability for embedded applications. Instead of supporting a large number of specialized instructions and multiple execution classes, the ISA adopts a reduced instruction set philosophy centered on arithmetic computation, memory access, control-flow execution, and basic system control. This minimal instruction organization decreases opcode allocation overhead and simplifies control signal generation within the single-cycle datapath.

The instruction set is organized into four primary functional categories, namely arithmetic and logical operations, data movement and memory access, control-flow instructions, and system operations. The overall instruction classification of AURORA-15 is summarized in Table 1.

Table 1. AURORA-15 Instruction Set Classification

Category	Instructions	Description
Arithmetic & Logical	ADD, SUB, AND, OR, SHL	ACC-based ALU operations
Data Movement & Memory	LI, LD, ST	Immediate and base–offset access
Control Flow	JMP, JZ, JNZ	PC-relative branching
System	NOP, HALT	Program control

Arithmetic and logical instructions operate using an accumulator-influenced execution model in which the accumulator register implicitly participates as one of the ALU operands. This approach reduces operand-field requirements and enables compact instruction encoding without increasing datapath complexity. The arithmetic instruction group supports addition, subtraction, logical conjunction, logical disjunction, and shift operations required for general embedded integer processing tasks.

Data movement and memory-access instructions support immediate loading and aligned memory transfer operations. Load and store instructions employ base-offset addressing to simplify effective-address generation while maintaining compatibility with the 28-bit datapath organization. The immediate load instruction enables compact constant generation without requiring additional instruction formats or extended operand fields.

Control-flow operations are implemented using PC-relative branching to eliminate complex absolute branch hardware and improve position-independent execution capability. Conditional branch instructions utilize ALU status evaluation for branch decision generation, while unconditional branching supports deterministic program sequencing and subroutine-oriented execution flow.

System-level instructions provide essential execution control without introducing unnecessary architectural overhead. The NOP instruction enables timing alignment and controlled execution sequencing, whereas the HALT instruction terminates processor execution during validation and embedded control applications.

The reduced instruction organization of AURORA-15 improves hardware efficiency by minimizing decoder circuitry, simplifying control signal generation, and reducing opcode-space fragmentation. At the same time, the selected instruction set preserves sufficient flexibility for arithmetic computation, memory manipulation, and control-flow execution required in compact embedded processing environments.

3.5 Addressing Modes and Memory Model

AURORA-15 adopts a simplified memory organization and reduced addressing mechanism to minimize control-unit complexity and ensure deterministic execution behavior within the single-cycle datapath. The memory system is organized as a byte-addressable architecture, while all load and store operations are aligned to the 28-bit datapath width to maintain uniformity across arithmetic, logical, and memory-access operations. The simplified memory organization reduces address-generation overhead and eliminates the requirement for complex alignment correction hardware commonly observed in wider multi-format architectures.

The ISA supports three primary addressing modes, namely register direct addressing, immediate addressing, and base-offset addressing. Register direct addressing is primarily used for arithmetic and logical instructions, allowing operands to be accessed directly from the register file without additional memory-reference cycles. This organization minimizes operand-fetch latency and supports efficient single-cycle ALU execution.

Immediate addressing is implemented using the 8-bit immediate field available in Format-B instructions. During execution, the immediate operand is sign-extended to 28 bits before being forwarded to the datapath. The use of compact immediate encoding reduces instruction width while preserving compatibility with the 28-bit execution framework.

This addressing mode is primarily used for constant loading, arithmetic initialization, and offset generation during address computation.

Base-offset addressing is employed for load and store instructions to simplify memory-access operations. In this approach, the effective address is generated by adding a sign-extended offset to the contents of a base register using the arithmetic logic unit. The use of ALU-assisted address computation avoids the need for dedicated address-generation hardware and reduces datapath complexity. Furthermore, restricting memory access to aligned base-offset operations improves timing predictability and simplifies memory-interface control.

Control-flow execution in AURORA-15 is implemented exclusively using PC-relative branching. Instead of utilizing absolute branch targets that require large address fields and complex branch hardware, the branch destination is computed relative to the current program counter value. The branch target calculation is expressed as:

$$PC_{target} = PC + 2 + (offset \times 2)$$

where the offset field represents a signed 8-bit displacement value. Since the program counter stores byte addresses and each instruction occupies a 16-bit aligned memory location, the program counter increments by two after every instruction fetch. The signed branch displacement therefore supports both forward and backward branching within a compact address range while maintaining instruction alignment throughout execution.

The branch offset range is determined by the signed 8-bit immediate field:

$$-128 \leq offset \leq +127$$

Accordingly, the effective branch reach becomes:

$$-256 \text{ bytes} \leq \text{Branch Range} \leq +254 \text{ bytes}$$

This PC-relative organization significantly reduces branch-hardware complexity and improves position-independent execution capability compared with architectures relying on absolute control-flow addressing. The overall addressing and memory organization of AURORA-15 therefore achieves reduced hardware overhead, compact instruction encoding, and deterministic execution behavior suitable for embedded and FPGA-oriented processor implementations.

4 Processor Datapath Architecture

The AURORA-15 processor is implemented using a compact single-cycle datapath architecture designed to achieve deterministic instruction execution with reduced hardware overhead. The architecture integrates the program counter, instruction memory, register file, arithmetic logic unit, sign-extension unit, data memory interface, and control logic within a unified 28-bit execution framework. By combining compact instruction encoding with simplified datapath organization, the processor minimizes combinational control complexity while maintaining efficient support for arithmetic computation, memory access, and control-flow execution. The datapath organization is optimized for FPGA-oriented realization and educational processor implementation, where predictable timing behavior and reduced resource utilization are of primary importance.

4.1 Single-Cycle Datapath Architecture

The AURORA-15 processor is implemented using a compact single-cycle datapath architecture designed to achieve deterministic instruction execution with minimal hardware overhead. In the proposed organization, instruction fetch, decode, operand access, arithmetic execution, memory access, and result write-back are completed within a single clock cycle. This execution model eliminates the need for complex multi-stage sequencing and simplifies processor control while maintaining predictable timing behavior suitable for embedded and FPGA-based processor implementations.

The overall datapath organization of the proposed architecture is illustrated in Figure 4.

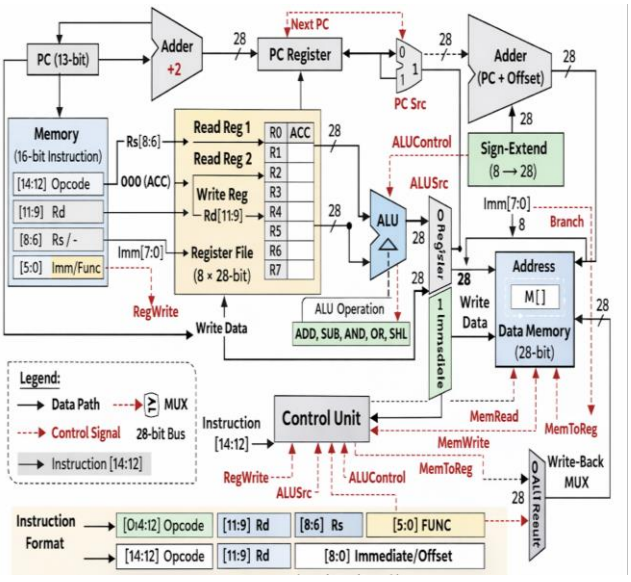

Fig. 4: Processor Datapath Architecture of a URORA-15

The datapath integrates the program counter, instruction memory, register file, arithmetic logic unit (ALU), sign-extension unit, data memory interface, and write-back selection logic within a unified 28-bit execution framework. The program counter stores byte-addressable instruction locations and supplies the address input to instruction memory during

instruction fetch. Since each instruction occupies a 16-bit aligned storage location, the PC is incremented by two through a dedicated address adder after every instruction fetch, ensuring consistent instruction alignment and simplified control flow.

The fetched instruction is decoded to extract opcode, register, function, and immediate fields required for datapath operation. Operand access is performed through the compact eight-register organization, which provides two simultaneous read ports and one write port for efficient single-cycle execution. The accumulator register is implicitly routed as one of the ALU operands during arithmetic and logical operations, thereby reducing operand-selection hardware and simplifying multiplexing logic within the execution stage.

The arithmetic logic unit performs integer arithmetic, logical computation, effective-address generation, and branch-condition evaluation. For immediate and memory-access instructions, the 8-bit immediate or offset field is sign-extended to 28 bits before being forwarded to the ALU. This sign-extension mechanism maintains datapath uniformity across arithmetic, addressing, and branch computations while avoiding additional conversion hardware.

Load and store instructions employ base-offset addressing in which the ALU computes the effective memory address using a base register and sign-extended offset value. The memory interface supports aligned 28-bit data transfer between the register file and data memory. During write-back operation, a multiplexer selects either ALU-generated data or memory-read data before updating the destination register.

Control-flow execution is implemented using PC-relative branching. The branch target address is generated by adding the sign-extended offset to the incremented program counter value, while branch-selection logic determines the next instruction address based on branch conditions. This organization eliminates complex absolute branch hardware and enables compact position-independent execution.

The simplified single-cycle datapath architecture of AURORA-15 therefore achieves reduced interconnect complexity, compact control organization, deterministic instruction execution, and efficient hardware realization while preserving the functional capability required for embedded processing applications.

4.2 Control Unit and Decoder Design

The AURORA-15 processor employs a hardwired control organization to achieve compact control generation and deterministic instruction execution within the single-cycle datapath. Unlike microprogrammed architectures that require multiple control states and sequencing memory, the proposed processor directly generates datapath control signals from instruction opcode and function fields using combinational decoding logic. This approach significantly reduces control latency and hardware overhead while improving timing predictability for embedded processing applications.
The control unit receives the opcode, register fields, and function bits extracted during instruction decoding and generates the required control signals for register access, ALU operation selection, memory interface control, branch evaluation, and write-back routing. Since the proposed ISA employs only two instruction formats, the decoder organization remains compact and requires fewer combinational decoding stages compared with conventional multi-format RISC architectures.

The control organization of AURORA-15 is summarized in Table 2.

Table 2. Decoder Operation and Control Function Mapping

Opcode Category	**Instruction Type**	**Decoder Action**	**ALU Control Requirement**
Arithmetic / Logical	ADD, SUB, AND, OR, SHL	Enable register access and ALU execution	Determined by function field
Immediate Operations	LI	Enable immediate routing and register write-back	Addition
Memory Access	LD, ST	Enable address generation and memory interface	Addition
Control Flow	JMP, JZ, JNZ	Enable branch evaluation and PC selection	Comparison / subtraction
System Operations	NOP, HALT	Disable datapath modification	No ALU operation

Instruction decoding is performed using a simplified opcode-driven decoding structure in which

each opcode activates a dedicated control path corresponding to the required instruction category. Arithmetic and logical instructions additionally utilize the function field of Format-A instructions to determine the specific ALU operation to be executed. This hierarchical decoding mechanism separates instruction-class identification from ALU-function selection, thereby reducing overall control complexity and improving modularity within the control architecture.

For arithmetic and logical instructions, the opcode identifies the arithmetic instruction class while the function field determines the exact ALU operation. The ALU control logic interprets the function bits to generate operation-specific control signals for addition, subtraction, logical conjunction, logical disjunction, and shift execution. This organization avoids excessive opcode-space utilization and preserves encoding flexibility for future ISA expansion.

Memory-access instructions utilize the ALU to compute effective addresses using base-offset addressing. During load operations, the control unit enables memory-read functionality and activates write-back routing toward the destination register. For store instructions, memory-write control is enabled while register write-back remains disabled. Similarly, branch instructions activate branch-evaluation logic and next-PC selection circuitry without modifying the register file.

The simplified hardwired control architecture reduces combinational decoder depth, minimizes control-signal propagation delay, and improves deterministic processor behavior. Furthermore, the reduced instruction-set organization and limited addressing mechanisms contribute to lower hardware complexity and improved implementation efficiency for FPGA-based and resource-constrained embedded processor environments.

4.3 Control Unit and Decoder Design

The operation of the AURORA-15 datapath is coordinated through a compact set of control signals generated by the hardwired control unit during instruction decoding. These control signals regulate register access, ALU operand selection, memory transactions, branch evaluation, and result write-back within the single-cycle execution framework. The reduced instruction-format organization and simplified addressing mechanisms of AURORA-15 significantly decrease control-generation complexity when compared with conventional multi-format RISC architectures.

The primary control signals used in the proposed architecture include RegWrite, ALUSrc, MemRead, MemWrite, Branch, and ALUOp. Each signal activates a specific datapath operation depending on the decoded instruction category. Since arithmetic, memory, and control-flow instructions share a common execution framework, the control unit can generate the required datapath configuration using a compact combinational control structure.

The RegWrite signal enables updating of the destination register during arithmetic, logical, immediate, and load instructions. Instructions that do not modify the register file, such as store and branch operations, disable this signal to avoid unintended register updates. The ALUSrc signal selects either register-based operands or sign-extended immediate values as ALU inputs depending on the instruction format and addressing mode.

Memory-access operations are controlled using MemRead and MemWrite signals. During load instructions, MemRead activates the data-memory output path and enables memory data transfer toward the write-back stage. Conversely, MemWrite enables aligned memory updates during store operations while disabling register write-back functionality.

Control-flow instructions utilize the Branch signal to activate branch-target computation and next-PC selection logic. Conditional branch instructions additionally use ALU-generated status information to determine whether sequential execution or branch redirection should occur. The ALUOp control field determines the arithmetic or logical operation to be executed by the ALU, including addition, subtraction, logical conjunction, logical disjunction, and shift operations.

The control-signal configuration for representative instruction classes is summarized in Table 3.

Table 3. Control Signal Configuration for AURORA-15 Instructions

Instruction Type	RegWrite	ALUSrc	MemRead	MemWrite	Branch	ALUOp
Arithmetic / Logical	1	0	0	0	0	FUNC
Immediate Load (LI)	1	1	0	0	0	ADD
Load (LD)	1	1	1	0	0	ADD
Store (ST)	0	1	0	1	0	ADD
Conditional	0	0	0	0	1	SUB

Branch (JZ/JNZ)						
Unconditional Jump (JMP)	0	X	0	0	1	X
NOP / HALT	0	X	0	0	0	X

The simplified control-signal organization reduces combinational decoder depth and minimizes datapath control overhead while preserving deterministic execution behavior across all instruction categories. In addition, the compact control structure contributes to reduced switching activity, lower interconnect complexity, and improved implementation efficiency for FPGA-based and low-power embedded processor applications.

5 Results & Discussion

This section presents the functional validation and architectural evaluation of the proposed AURORA-15 ISA. The processor behavior is verified using representative instruction sequences to confirm correct arithmetic operations, memory access, and control-flow execution. In addition, an analytical comparison with a conventional MIPS-style architecture is provided to assess code density, register file size, and decoder complexity. These evaluations demonstrate the effectiveness of the proposed design in achieving reduced hardware overhead while maintaining functional correctness for embedded applications.

5.1 Functional Verification

The functional correctness of the proposed AURORA-15 processor was verified using Logisim Evolution through step-by-step single-cycle execution analysis. The verification process evaluated arithmetic operations, register updates, ALU functionality, program-counter progression, instruction decoding, and control-flow execution within the implemented datapath architecture. Since the processor follows a deterministic single-cycle organization, each instruction completes execution within one clock cycle, enabling direct observation of datapath transitions and control-signal behavior during simulation.

Initially, the program counter was reset to the starting instruction location while all registers were initialized to zero. Immediate load instructions were then executed to transfer constant values into the general-purpose registers. During execution, the program counter incremented by two after every instruction fetch, confirming correct instruction alignment and sequential control flow within the instruction memory space. The register file simultaneously demonstrated correct operand fetch and write-back operation through dual-read and single-write datapath functionality.

Arithmetic instruction verification was performed using accumulator-based addition operations. The ALU successfully executed arithmetic computations by implicitly utilizing the accumulator register as one of the source operands. Simulation results confirmed correct ALU output generation, accumulator updates, and register synchronization during execution. The observed execution sequence verified that arithmetic operations were correctly decoded and routed through the datapath without control conflicts or timing inconsistencies.

Branch and control-flow validation were also performed during simulation. Conditional branch instructions correctly evaluated ALU-generated status information and selected the appropriate next-PC path through the branch-selection logic. The branch mechanism successfully redirected program execution relative to the current program counter value, validating the correctness of the PC-relative control-flow organization implemented in the processor.

The step-by-step execution behavior of the proposed processor is illustrated in Figure 6.

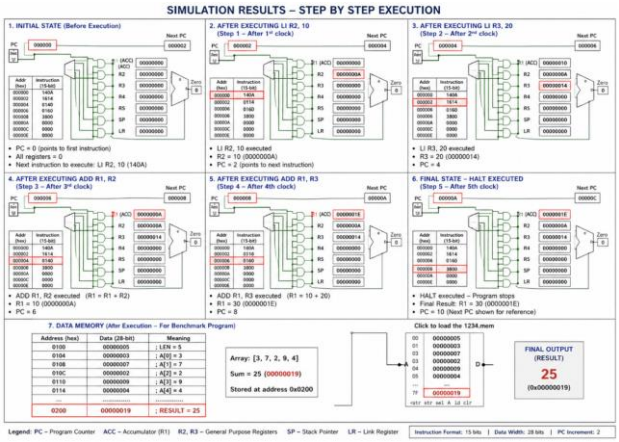

Fig. 6: Step-by-Step Functional Verification of a URORA-15 in Logisim Evolution

The major instruction-level execution results obtained during functional verification are summarized in Table 4.

Table 4. Functional Verification Summary of AURORA-15

Step	Executed Instruction	Register / ALU Result	PC Value
1	LI R2, 10	R2 = 10	0x0002
2	LI R3, 20	R3 = 20	0x0004
3	ADD R1, R2	ACC = 10	0x0006
4	ADD R1, R3	ACC = 30	0x0008
5	HALT	Execution terminated	0x000A

The simulation results confirm that the proposed datapath, register organization, ALU operation, branch mechanism, and write-back logic operate correctly within the single-cycle execution framework. The successful execution of arithmetic computation, register transfer, and control-flow instructions demonstrates the functional validity of the AURORA-15 architecture for compact embedded processor applications.

5.2 Benchmark Validation

To evaluate the operational correctness of the proposed architecture under practical execution conditions, a benchmark validation was performed using an array summation program implemented in the AURORA-15 instruction set. The benchmark was designed to verify arithmetic computation, memory access, register updates, loop execution, conditional branching, and accumulator-based processing within the single-cycle datapath environment. The program repeatedly accessed array elements from memory, accumulated the intermediate results using the arithmetic logic unit, and stored the final output at a predefined memory location.

During initialization, the data memory was populated with an array of integer values together with the corresponding array length parameter. The benchmark program loaded the base memory address and loop parameters into the register file before beginning iterative execution. The accumulator register was initialized to zero and subsequently updated during every loop iteration as array elements were fetched from memory and added to the accumulated result.

The benchmark execution process involved repeated load, add, increment, and branch operations. Load instructions transferred array elements from memory into the register file using base-offset addressing, while arithmetic instructions updated the accumulator with the current partial sum. The loop counter and address pointer were updated sequentially during each iteration to access the next memory location. This iterative execution validated the coordinated operation of the register file, ALU, memory interface, and write-back logic within the proposed datapath architecture.

Conditional branch instructions were used to control loop termination. During every iteration, the branch logic evaluated the loop condition using ALU-generated status information and redirected execution to the loop body until all array elements had been processed. The successful execution of conditional branching verified the correctness of the PC-relative branch mechanism and branch-target computation path implemented in the processor architecture.

The benchmark execution summary obtained during simulation is presented in Table 5.

Table 5. Benchmark Execution Summary for Array Summation Program

Execution Stage	Operation Performed	Result
Memory Initialization	Array values loaded into memory	Successful
Register Initialization	Base address and counters initialized	Successful
Memory Access	Sequential array elements fetched	Verified
Arithmetic Execution	Accumulator updated during each iteration	Verified
Loop Execution	Iterative branch-controlled execution	Verified
Branch Validation	Conditional branching performed correctly	Verified
Final Result Storage	Sum stored in destination memory location	Successful
Final Output	Computed array sum	25 (0x00000019)

To further validate instruction compatibility and execution flexibility, portions of a conventional MIPS-style benchmark sequence were manually adapted into the AURORA-15 instruction format and executed within the proposed datapath. Unsupported instruction formats were replaced using equivalent accumulator-oriented instruction sequences while preserving arithmetic and control-flow behavior. The successful execution of the converted benchmark demonstrated the capability of the proposed ISA to support general embedded computation using compact instruction encoding and simplified control organization.

The benchmark validation results confirm that the proposed processor correctly supports arithmetic execution, iterative loop control, memory-address generation, conditional branching, and accumulator-

based computation within the single-cycle execution framework. The successful execution of practical benchmark operations further demonstrates the functional reliability and architectural consistency of the AURORA-15 processor for compact embedded processing applications.

5.3 Hardware Complexity Analysis

The proposed AURORA-15 architecture is designed to reduce datapath complexity, decoder overhead, and instruction-processing cost through compact instruction encoding, reduced register organization, simplified addressing mechanisms, and deterministic single-cycle execution. Unlike conventional 32-bit RISC processors that employ wide instruction formats and large register files, the proposed architecture minimizes hardware resources while preserving sufficient computational capability for embedded and educational processing applications.

One of the primary architectural optimizations achieved by AURORA-15 is the reduction in instruction width. Since the proposed processor employs a fixed-length 15-bit instruction format instead of a conventional 32-bit organization, the reduction in instruction size can be expressed as:

$$\frac{32-15}{32}\times 100=53.125\%$$

The reduced instruction width decreases instruction-memory utilization, fetch bandwidth, decoder-input complexity, and switching activity during instruction execution. Furthermore, the use of only two instruction formats minimizes combinational decoder logic and simplifies control-signal generation compared with architectures employing multiple instruction classes and variable decoding paths.

The proposed processor also employs a compact eight-register organization instead of the 32-register structure commonly used in conventional RISC architectures. The reduction in register count is expressed as:

$$\frac{32-8}{32}\times 100=75\%$$

This reduced register organization lowers register-decoder complexity, operand-routing overhead, multiplexer requirements, and interconnect congestion within the datapath. In addition, the accumulator-based execution model reduces explicit operand encoding requirements and simplifies ALU operand-selection hardware during arithmetic execution.

The overall register-storage requirement of the proposed architecture is substantially smaller than that of a conventional MIPS-style processor. For a conventional 32-register 32-bit architecture, the total register-storage requirement becomes:

$$S_{MIPS}=32\times 32=1024 \text{ bits}$$

For the proposed AURORA-15 architecture, the corresponding register-storage requirement is:

$$S_{AURORA}=8\times 28=224 \text{ bits}$$

The reduced storage organization contributes directly to lower silicon utilization and decreased switching capacitance within the register file subsystem.

The comparative hardware characteristics of AURORA-15 and conventional MIPS-style architecture are summarized in Table 6.

Table 6. Hardware Complexity Comparison Between MIPS and AURORA-15

Metric	**MIPS (Typical)**	**AURORA-15**
Instruction width	32 bits	15 bits
Register count	32	8
Instruction formats	3	2
Branch mechanism	Multiple modes	PC-relative only
Decoder complexity	High	Low

In addition to hardware simplification, the proposed architecture also improves theoretical dynamic power efficiency by reducing switching activity within the instruction-fetch, decoder, and datapath subsystems. The dynamic power behavior of CMOS-based digital systems is generally represented as:

$$P_{dyn}=\alpha CV2f$$

where α represents switching activity, C denotes effective switching capacitance, V is the operating voltage, and f is the operating frequency.

The compact instruction organization of AURORA-15 reduces instruction-fetch transitions and decoder switching activity, thereby lowering effective capacitance within the control path. Similarly, the smaller register organization decreases operand-routing complexity and register-interconnect activity during execution. The use of only two instruction formats further simplifies combinational decoding hardware and reduces unnecessary control transitions. In addition, PC-relative branching and simplified addressing mechanisms eliminate complex address-generation hardware and contribute to lower datapath switching activity during control-flow execution.

The overall architectural organization of AURORA-15 therefore achieves reduced hardware overhead, simplified decoder implementation, lower instruction-fetch bandwidth, decreased register interconnect complexity, and improved theoretical

power efficiency when compared with conventional wide-instruction embedded processor architectures.

5.4 Code Density and ISA Comparison

Code density plays an important role in embedded processor architectures because instruction-storage requirements directly affect memory utilization, instruction-fetch bandwidth, and execution efficiency. The proposed AURORA-15 architecture improves code compactness through reduced instruction width, simplified operand encoding, compact register addressing, and PC-relative control-flow organization. These architectural optimizations collectively reduce program-memory footprint while maintaining sufficient functionality for arithmetic computation, memory access, and control-flow execution.

The fixed-length 15-bit instruction organization significantly decreases instruction-storage requirements when compared with conventional 32-bit RISC architectures. Since every instruction occupies less than half the storage size of a standard 32-bit instruction, the proposed processor reduces instruction-fetch bandwidth and memory-access activity during execution. This reduction is particularly beneficial for embedded systems and FPGA-oriented soft processors where memory resources and fetch energy are constrained.

The accumulator-based execution model further improves encoding efficiency by implicitly utilizing the accumulator register during arithmetic and logical operations. Unlike conventional register–register architectures that repeatedly encode multiple source operands within every instruction, AURORA-15 reduces operand-field utilization by eliminating one explicitly encoded arithmetic operand. This compact operand organization enables simplified instruction representation while reducing decoder complexity and datapath routing overhead.

Code compactness is additionally improved through the reduced eight-register organization employed by the architecture. Since fewer register-specifier bits are required within each instruction, the available encoding space can be utilized more efficiently for immediate values, opcode allocation, and branch displacement representation. The use of only two instruction formats also simplifies instruction organization and eliminates unnecessary format-specific encoding overhead.

Control-flow instructions in AURORA-15 utilize PC-relative branching instead of absolute address representation. This organization avoids large branch-target fields and enables compact branch encoding using sign-extended relative offsets. Consequently, control-flow instructions require smaller instruction overhead while simultaneously reducing branch-generation complexity.

A comparative analysis of instruction-storage requirements for representative operations is presented in Table 7.

Table 7. Code Density Comparison for Representative Operations

Operation	**MIPS Instruction Size**	**AURORA-15 Instruction Size**
Load immediate	32 bits	15 bits
Register addition	32 bits	15 bits
Memory load/store	32 bits	15 bits
Conditional branch	32 bits	15 bits

The comparison demonstrates that AURORA-15 consistently achieves lower instruction-storage requirements across arithmetic, memory-access, and control-flow operations. The reduced instruction width, compact operand organization, and simplified branching mechanism collectively improve code density and reduce instruction-fetch overhead, making the proposed architecture well suited for compact embedded processing environments.

6 Conclusion

This paper presented AURORA-15, a compact 15-bit fixed-length instruction set architecture designed for low-power embedded processors and FPGA-based soft-core implementations. The proposed architecture integrates a 28-bit datapath, accumulator-assisted execution model, compact eight-register organization, and simplified control structure to reduce datapath complexity, decoder overhead, and instruction-storage requirements while maintaining functional support for arithmetic, memory, and control-flow operations.

The architecture employs only two instruction formats together with PC-relative branching and base-offset memory addressing to achieve compact instruction encoding and deterministic single-cycle execution. Analytical evaluation demonstrated a 53.125% reduction in instruction width and a 75% reduction in register organization compared with conventional 32-bit RISC architectures, resulting in lower instruction-fetch bandwidth, reduced decoder complexity, and improved theoretical power efficiency.

Functional validation using Logisim Evolution confirmed correct execution of arithmetic operations, register updates, ALU functionality, memory-access operations, program-counter progression, and branch execution within the proposed datapath framework. Benchmark validation using array summation and conditional branching further verified iterative execution, memory interfacing, and control-flow correctness under practical execution scenarios.

The overall results demonstrate that AURORA-15 achieves compact instruction organization, simplified hardware realization, and efficient embedded execution capability, making the architecture suitable for educational processors, FPGA-oriented systems, and resource-constrained embedded computing applications. Future work will focus on pipelined implementation, FPGA synthesis, and hardware-level area, timing, and power characterization.

References:

[1] D. A. Patterson and J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, 5th ed. San Francisco, CA, USA: Morgan Kaufmann, 2014.

[2] D. A. Patterson and J. L. Hennessy, Computer Architecture: A Quantitative Approach, 6th ed. San Francisco, CA, USA: Morgan Kaufmann, 2019.

[3] MIPS Technologies, MIPS32® Architecture for Programmers Volume I: Introduction to the MIPS32® Architecture, 2009.

[4] ARM Ltd., ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition. Cambridge, U.K.: ARM Ltd., 2012.

[5] S. Furber, ARM System-on-Chip Architecture, 2nd ed. Reading, MA, USA: Addison-Wesley, 2000.

[6] K. Asanović et al., "The landscape of parallel computing research: A view from Berkeley," EECS Dept., Univ. California, Berkeley, Tech. Rep. UCB/EECS-2006-183, 2006.

[7] J. L. Hennessy and N. P. Jouppi, "Computer technology and architecture: An evolving interaction," Computer, vol. 24, no. 9, pp. 18–29, Sep. 1991.

[8] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The RISC-V instruction set manual, volume I: User-level ISA," EECS Dept., Univ. California, Berkeley, 2014.

[9] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," in Proc. Design, Automation and Test in Europe (DATE), 2001, pp. 642–649.

[10] J. E. Smith and G. S. Sohi, "The microarchitecture of superscalar processors," Proc. IEEE, vol. 83, no. 12, pp. 1609–1624, Dec. 1995.

[11] M. Morris Mano and M. D. Ciletti, Digital Design, 5th ed. Upper Saddle River, NJ, USA: Pearson, 2013.

[12] A. S. Tanenbaum and T. Austin, Structured Computer Organization, 6th ed. Boston, MA, USA: Pearson, 2012.

[13] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," Commun. ACM, vol. 62, no. 2, pp. 48–60, Feb. 2019.

[14] D. Seal, ARM Architecture Reference Manual, 2nd ed. Reading, MA, USA: Addison-Wesley, 2001.

[15] ARM Ltd., "Thumb-2 technology overview," White Paper, 2003.

[16] R. E. Bryant and D. R. O'Hallaron, Computer Systems: A Programmer's Perspective, 3rd ed. Boston, MA, USA: Pearson, 2016.

[17] J. L. Hennessy, "The role of instruction-set architectures in computer architecture," Computer, vol. 15, no. 9, pp. 15–24, Sep. 1982.

[18] M. Hill and M. Marty, "Amdahl's law in the multicore era," Computer, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[19] P. Mishra and N. Dutt, Processor Description Languages: Applications and Methodologies. San Francisco, CA, USA: Morgan Kaufmann, 2008.

[20] S. Hauck and A. DeHon, Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. San Francisco, CA, USA: Morgan Kaufmann, 2007.

[21] J. Shen and M. Lipasti, Modern Processor Design: Fundamentals of Superscalar Processors. New York, NY, USA: McGraw-Hill, 2005.

[22] A. Waterman and K. Asanović, "The RISC-V instruction set manual, volume II: Privileged architecture," 2019.

[23] N. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed. Boston, MA, USA: Pearson, 2011.

[24] J. Bhasker and R. Chadha, Static Timing Analysis for Nanometer Designs. New York, NY, USA: Springer, 2009.

[25] D. Flynn and W. Luk, "Computer system design: System-on-chip," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 9, no. 1, pp. 5–15, Feb. 2001.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

Venkateswara Reddy Kunduru solely contributed to the conceptualization of the research problem, architectural design of the proposed AURORA-15 instruction set architecture, definition of instruction formats, register organization, opcode allocation, and addressing modes, datapath modeling and functional validation, analytical comparison with conventional ISAs, interpretation of results, and preparation of the manuscript.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The author has no conflicts of interest to declare that are relevant to the content of this article.