STM JOURNALS

Journal of
Electronic Design Technology

ISSN: 2229-6980 (Online)
ISSN: 2321-4228 (Print)
Volume 17, Issue 2, 2026
May–August
DOI (Journal): 10.37591/JoEDT

https://journals.stmjournals.com/joedt

Review JoEDT

Design and Implementation of a 4-Bit Breadboard CPU Using Digital Integrated Circuits

Aditya Chauhan[1,*], Shrushti Amin[2], Himanshu Prajapati[3], Purvang Dalal[4]

Abstract

This paper presents the design and hardware realization of a simple 4-bit CPU built using discrete TTL integrated circuits on a breadboard. The CPU employs a 4-bit common data bus with tri-state buffers for device arbitration, three 4-bit registers (Register A, Register B, and an Output Register) implemented with 74LS175 quad D flip-flops, and a 4-bit ALU for addition/subtraction using a 74LS83 adder and XOR gates. Manual control signals (load, enable, add/sub, etc.) and DIP switch inputs allow step-by-step operation under a hand-driven clock pulse. The system was first verified in simulation and then tested on a physical proto-type. Experiments confirmed correct execution of register load, register-to-register transfer, add, and subtract operations across multiple test cases, with results validated against expected values on the LED output array. Beyond the specific hardware result, the project is framed as a self-contained pedagogical case study: every design decision, from the choice of a shared tri-state bus to the reuse of a single adder for both addition and subtraction, is made deliberately visible and traceable to a single hand-clocked step, so that a learner can connect abstract register-transfer-level concepts directly to measurable electrical behaviour. The reported test cases, spanning ordinary operation, zero operands, and deliberate overflow/underflow conditions, are intended to serve as a compact, reproducible verification checklist for similar low-cost, breadboard-scale processor builds, and the resulting datapath is explicitly positioned as a reusable foundation for the automated, control-unit-driven extension discussed later in the paper.

Keywords: 4-bit CPU, breadboard implementation, Tri-state buffers, ALU, TTL circuits digital design.

INTRODUCTION

Building a CPU from basic digital ICs provides an educational bridge between theoretical concepts and practical designs. Simple TTL-based CPUs have been used as learning tools to illustrate processor architectures and data flows [1, 2]. As Wabiszewski notes, constructing a computer on a breadboard helps explain how computer hardware functions and makes abstract concepts concrete [1]. This project is a smaller-scale adaptation of the breadboard-CPU concept popularized by hobbyist and educational builds in this space, scoped down to a 4-bit datapath that performs register-to-register and ALU (add/subtract) operations under a manually applied clock pulse, with automated (free-running) clocking left for future work. This project aims to design, simulate, and implement a functional 4-bit CPU using 74xx-series TTL ICs and validate its operations in hardware.

Unlike a microprocessor-based system, where the internal register transfers, bus activity, and control decisions are hidden inside a single packaged die, a discrete-IC CPU exposes each of these operations to direct measurement and observation. A learner can probe the data bus with an oscilloscope or simply watch an LED, physically trace a load or enable a signal from the switch to the chip pin, and correlate a single hand-applied clock pulse with a single, isolated change in the machine state. This

***Author for Correspondence**
Aditya Chauhan
E-mail: aadityabrc@gmail.com

[1,2]Student, Department of Electronics and Communication, Dharmsinh Desai University, Nadiad, Gujarat, India
[3,4]Faculty Guide, Department of Electronics and Communication, Dharmsinh Desai University, Nadiad, Gujarat, India

Received Date: July 15, 2026
Accepted Date: July 20, 2026
Published Date: July 27, 2026

Citation: Aditya Chauhan, Shrushti Amin, Himanshu Prajapati, Purvang Dalal. Design and Implementation of a 4-Bit Breadboard CPU Using Digital Integrated Circuits. Journal of Electronic Design Technology. 2026; 17(2): 30–42p.

transparency is difficult to reproduce with a packaged microcontroller or an FPGA soft core, where the fetch-execute cycle proceeds automatically and at a speed unsuited for manual inspection. The 4-bit scope chosen for this project keeps the chip count, wiring complexity, and cost low enough for a single undergraduate to design, wire, and debug within a typical project timeline, while still being large enough to require genuine bus arbitration, register sequencing, and two's-complement arithmetic – the same core ingredients found in any larger processor.

The pedagogical value of exposing processor internals in this way continues to be actively studied in the computer-architecture education literature rather than being a settled or purely historical concern. In a recent large-scale classroom study, it was reported that having undergraduate teams design, simulate, and then verify their own small processors in a hardware description language measurably increases student motivation and depth of understanding relative to purely lecture- or simulator-based instruction, and hands-on, staged design-build-verify workflows were identified as a key driver of that improvement [3]. This finding is consistent with the discrete TTL approach followed here, in which the same design is first captured and exercised in a simulation and only afterward committed to physical hardware, giving the builder an analogous staged verification experience at a much smaller and more affordable scale. Simultaneously, the broader trend in undergraduate computer-architecture teaching has moved toward open, reconfigurable platforms: survey the pedagogical strengths and weaknesses of existing open-source RISC-V processor implementations and argue that a composable, well-documented reference design substantially lowers the barrier for instructors who want students to modify and extend a working processor rather than merely study one in the abstract [4]. The present work adopts the same underlying philosophy of an inspectable, modular, and easily extended design but applies it at the level of physical TTL hardware rather than synthesizable RTL, so that the first artifact a learner interacts with is a set of real voltages on a real bus rather than a waveform in a simulator window.

Taken together, these recent studies suggest that discrete hardware and HDL/FPGA-based teaching approaches are not competing alternatives but complementary stages of the same learning progression: a small, hand-clocked, breadboard-resident CPU, such as the one presented here, makes the lowest-level electrical behavior of a processor directly observable, while composable HDL platforms of the kind surveyed by McDougall et al. later let the same student explore scale, pipelining, and instruction-set design once the fundamental register-ALU-bus relationship is already understood at the hardware level [3, 4]. This project is positioned explicitly at the first of these stages, with the intention that the design and verification methodology described in the following sections can serve as a starting point for this kind of staged, multi-platform computer-architecture curriculum.

LITERATURE REVIEW

The design of educational-scale processors using discrete digital logic has long been an effective pedagogical approach for understanding the fundamentals of computer architecture. Early instructional computing systems emphasized the visibility of data path operations and control sequencing, allowing learners to directly observe how arithmetic, storage, and data transfer operations form the basis of processor execution [1, 2]. Breadboard-based CPUs, in particular, provide a tangible bridge between abstract architectural models and real hardware implementations.

Fundamental processor organization principles originate from classical computer architecture research, where the separation between architecture and implementation enables a systematic exploration of performance and design trade-offs. Simplified hardware structures can improve the efficiency and clarity of processor operations, motivate reduced-complexity processor designs, and emphasize streamlined data paths and control mechanisms [5]. This perspective was further developed in the original VLSI RISC work, which showed that a small, regular instruction set and data path could be implemented with substantially fewer hardware resources while remaining easy to reason and verify [6]. These ideas remain central even in minimal educational CPUs, where simplicity enhances understandability and reliability. They directly motivate the decision in this work to keep the datapath manually sequenced rather than adding control-unit complexity before the basic register/ALU/bus operation is fully validated.

Modern architectural theory further formalizes the processor design around the quantitative evaluation of performance, cost, and implementation of trade-offs. Hennessy and Patterson described processor systems as structured compositions of data paths, control logic, and memory interactions governed by measurable design metrics, such as execution time and resource efficiency [7]. Stallings similarly frames the CPU as an interconnection of registers, an arithmetic-logic unit, and a control unit linked by internal buses, a view that maps directly onto the register-ALU-bus organization adopted in this study [8]. Although contemporary processors operate at massive scales, the same architectural abstractions register transfer operations, arithmetic units, and shared buses are preserved in small-scale instructional CPUs, such as the one presented here. At the circuit level, digital integrated circuit design emphasizes robustness, timing accuracy, and modular composition. Sequential elements, such as registers and combinational datapaths, form the backbone of synchronous digital systems, with their performance governed by propagation delay, power consumption, and interconnect behavior [9]. Standard digital electronics references describe how 74-series TTL devices, including quad D-type flip-flop registers and octal tri-state buffers, implement these sequential and bus-interfacing functions [10]. TTL-based implementations, while technologically older, remain ideal educational platforms because their behavior closely reflects fundamental digital logic principles without the abstraction layers introduced by modern VLSI automation. This view is reinforced by classic digital design texts such as Mano and Ciletti, who present register-transfer-level design, combinational ALU construction, and bus-structured datapaths as the standard conceptual vocabulary through which any processor, whether discrete or integrated, should be understood and analyzed [11].

Previous small-scale CPU implementations commonly employed shared data buses with tri-state buffering to prevent contention while enabling modular expansion [12]. Such architecture illustrates core processor concepts, including register transfers, ALU-based computation, and controlled data movement. Disciplined enable-signal sequencing is essential whenever multiple sources share a common bus, regardless of whether the bus is realized in discrete TTL or in a synthesizable hardware description language [13]. Bus arbitration correctness is not merely a matter of pedagogical convenience; at the industrial level, tri-state bus contention arising from imperfectly synchronized enable signals has been the subject of dedicated circuit patents describing timing schemes that guarantee a driver has fully released the bus (entered the high-impedance state) before another driver is permitted to assert it [14]. The single-enable at a time discipline adopted in the present design follows the same underlying principle in a much simpler, manually sequenced form, and reviews of high-performance on-chip bus arbitration schemes similarly stress that the choice of arbitration policy directly trades off throughput against implementation complexity, a trade-off that is trivially resolved here by restricting the design to one bus master at a time [15]. An alternative and increasingly common pedagogical route to the same learning outcomes is to implement a small CPU in an HDL, such as VHDL, and target an FPGA; this approach offers faster iteration and easier debugging at the cost of hiding some of the low-level electrical behavior (propagation delay, loading, bus contention) that a discrete TTL build makes directly observable [12]. The present design deliberately retains the discrete TTL and hand-clocked approach for this reason, even though it limits the operating speed and scale compared to clocked or HDL-based implementations.

A closely related strand of work has directly examined how discrete logic processors build function as teaching tools in formal engineering curricula. Hall described an undergraduate digital design laboratory sequence in which students first build a small state machine on a protoboard using discrete TTL logic ICs and only afterward re-implement and verify the same design in VHDL on an FPGA, explicitly to let learners compare gate-level and HDL-level views of the same processor behavior [16]. This staged TTL-to-HDL progression parallels the trade-off discussed above between physical transparency and design flexibility and supports treating a discrete-TTL build, such as the present one, as a pedagogically valuable first stage rather than a design dead end. In a related classroom-oriented study, a digital electronics teaching methodology that deliberately combines breadboard practice with discrete 74-series TTL chips alongside Multisim simulation and FPGA-based verification was proposed, so that students are not limited to either purely historical discrete parts or purely modern EDA-based abstractions [17].

Beyond curricular studies, dedicated small-scale computer kits and minimal processor architectures built from standard TTL parts have long served as concrete demonstrators of register-transfer-level (RTL) operations. A model computer system and laboratory kit were built from standard small-scale integration TTL chips on a breadboard, in which a shared bus, an instruction cycle generator, and associated control logic were used to make each step of program execution directly observable to a learner [18]. This type of fully visible, breadboard-resident control and bus structure is conceptually close to the manually sequenced load/enable control scheme adopted in the present CPU. At the other end of the complexity spectrum, a minimal stack-oriented processor (the PISC) built from only about twenty-two standard TTL chips shows that a very small gate count is sufficient to realize a working, fully static processor whose clock can be run slowly or single-stepped by hand for classroom observation [19]. Taken together with the tri-state bus-arbitration techniques discussed above, this body of work reinforces the view that discrete TTL remains a viable and instructive medium for realizing small processors, and it is this tradition on which the present 4-bit CPU builds. Complementing these hardware-focused efforts, dedicated breadboard-circuit simulation software, such as the Java Bread-Board (JBB) tool, has also been developed specifically to allow students to lay out and simulate TTL-chip-level digital circuits before committing to physical construction, reflecting the same simulate-then-build workflow used in the present project [19]. More recently, open textbooks and course projects such as Nisan and Schocken's "Elements of Computing Systems" have popularized the practice of building a complete computer – from NAND gates up through an assembler and simple operating system as a semester-long teaching exercise, and hobbyist video series such as Eater's breadboard-computer build have brought the same discrete-TTL, hand-clocked construction style to a very wide, non-academic audience, underscoring the enduring appeal of physically visible processor construction as a learning tool [20, 21].

Recently, the same design-build-verify philosophy has been re-examined from two complementary perspectives. Through a structured classroom study, it was found that undergraduate teams who designed a processor, implemented a desktop simulator for it, and then verified the design in an HDL report had significantly higher motivation and self-assessed understanding than teams following a purely lecture-based curriculum, directly supporting the pedagogical premise that staged, hands-on processor construction improves learning outcomes [3]. From the opposite end of the abstraction spectrum, the ecosystem of open-source RISC-V processor implementations is surveyed against a set of pedagogical criteria, and it is found that most existing designs are not well suited to classroom use without substantial adaptation, motivating a composable reference implementation intended specifically for teaching [4]. Neither of these recent efforts targets discrete TTL construction directly, but both reinforce the same underlying premise motivating the present work: that a design a student can inspect, modify, and verify end-to-end, whether in silicon-adjacent RTL or in physical logic chips on a breadboard, produces a qualitatively different and more durable understanding of processor architecture than a black-box treatment of the same concepts.

Therefore, existing work establishes four key foundations: (1) simplified architectures improve conceptual clarity; (2) processor functionality emerges from coordinated datapath and control design; (3) discrete digital circuits remain valuable for experimental validation of architectural concepts; and (4) the choice between discrete TTL and HDL/FPGA implementation is a pedagogical trade-off between physical transparency and design flexibility. Building upon these principles, the present study implements a functional 4-bit CPU using manually clocked TTL integrated circuits to experimentally validate the processor register-transfer and ALU operation at the hardware level.

CPU ARCHITECTURE AND DESIGN

Figure 1 shows the high-level architecture of CPU. A shared 4-bit data bus connects all the modules. The CPU has three main registers: Register A, Register B, and the Output Register, each 4 bits wide. These registers were implemented using 74LS175 quad D-type flip-flops. Each 74LS175 contains four edge-triggered D flip-flops with a common clock and an asynchronous reset [10]. In our design, the master reset (MR) inputs are held inactive (high) during normal operation. When a load control (LDA, LDB, or OUT) is asserted, the next manually applied clock pulse captures the current bus value in the corresponding register.

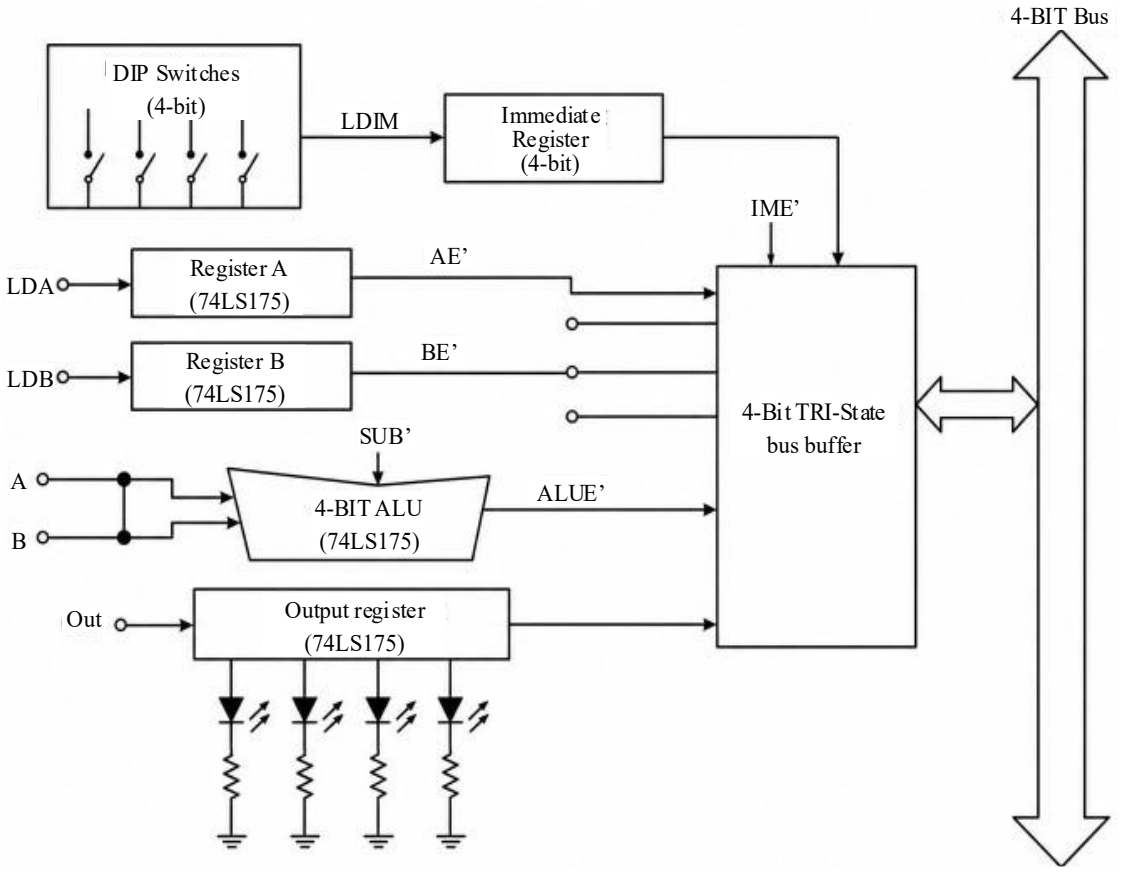

Figure 1. Block diagram of the 4-bit CPU: 4-bit registers (Reg A, Reg B, Output), ALU (Add/Sub), Immediate input, and control signals.

Each register interfaces with the bus through a tri-state buffer. We used 74LS244 octal bus transceiver chips, each containing two groups of four non-inverting buffers with active-low output-enable inputs [10]. When an output-enable (OE) input is active (low), the corresponding buffer drives the bus lines; when the OE is inactive (high), the buffer outputs are high-impedance. For example, when AE is asserted, the 4-bit output of Register A is driven onto the bus; otherwise, Register A is disconnected. This arrangement allows any device (Register A, Register B, ALU, or Immediate input) to drive the bus at a time, preventing conflicts [12]. By default, pull-up resistors hold the bus lines high when no device drives them.

Beyond the individual chip functions, the overall architecture reflects a deliberate separation between storage elements, computation, and interconnects, mirroring the classical register-ALU-bus organization discussed in Section "*Literature Review*". Every module that can source data onto the bus (Register A, Register B, the ALU, and the Immediate register) is buffered through its own dedicated tri-state stage, so that at the schematic level, the bus itself has no permanent driver and simply acts as a passive, shared signal path whose logic value at any instant is determined entirely by whichever single enable line is currently asserted. This one-driver-at-a-time discipline is the central invariant that the control signal sequencing described in Section "*Immediate Input and Control Signals*" is designed to preserve, and it is the same invariant that underlies bus arbitration in far larger processors and system-on-chip buses.

Register Modules

Registers A, B, and the Output Register each store a 4-bit value. Each register uses a 74LS175 integrated circuit (IC). In each chip, the four D flip-flops share a clock (CP) and a common asynchronous clear (MR) signal [10]. We keep MR inactive (high) so that the registers retain their state until actively cleared at startup. To load the data, the appropriate load line (LDA, LDB, or OUT) is

driven high, and a clock pulse is applied manually. On the rising edge of this manual pulse, the 4-bit bus value present at the inputs is latched into the register. For example, asserting LDA and pulsing the clock will store the bus value in Register A.

The Output Register's Q output drives the LEDs for visualization. Whenever the Output Register is loaded (OUT asserted on a clock pulse), the new 4-bit result appears immediately on the LED array (with appropriate resistors). This provides a real-time view of the CPU output. (Alternatively, the immediate register and its LED outputs can be treated similarly for debugging.)

Because all four register chips (Register A, Register B, the Output Register, and the Immediate register) are electrically identical 74LS175 devices, the design is highly modular. The same load-line-plus-clock-pulse discipline applies uniformly to every register in the system, and no register requires any special-case wiring beyond its bus-side tri-state buffer. This uniformity simplified both the schematic capture and the physical wiring, because a single verified register subcircuit could be replicated three additional times with only the control-line labeling changed, reducing the chance of a wiring error being introduced independently in each copy.

Arithmetic Logic Unit (ALU)

The ALU performs 4-bit addition and subtraction operations. We used a 74LS83 4-bit binary adder IC. Its inputs are the four bits from Register A and the four bits from Register B. The 74LS83 produces a 4-bit sum and carry-out bit (C0). To support subtraction, each bit of Register B is first passed through a 74LS86 (XOR gate) that inverts it when the subtraction control line (SUB') is low. In our design, SUB'=1 selects subtraction (inverting B), and SUB'=0 selects addition. When SUB'=1, we also manually set the carry-in to 1 (by wiring the carry-in pin high) to perform a two's complement subtraction. Thus, with SUB'=1 and carry-in=1, the ALU computes A + (NOT B) + 1 = A – B. The carry-out (from the 4-bit addition) can serve as a carry/borrow indicator. After the ALU computes a result, the 4-bit sum is available to place on the bus by asserting ALUE (enabling the ALU output buffer).

The XOR-based B-invert stage is a standard technique for sharing a single adder between addition and subtraction, as it avoids duplicating the 74LS83 for a separate subtractor and instead reuses the same ripple-carry hardware for both operations, with the operation selected purely by the logic level applied to SUB' and the adder's carry-in pin. One consequence of this shared-adder approach is that the carry-out bit changes meaning depending on the selected operation: during addition, it indicates an unsigned overflow out of the 4-bit range, whereas during subtraction (in two's-complement form), it complement effectively indicates a borrow. This dual interpretation of C0 is noted here because it directly informs the edge cases selected for the observation table in Section "Implementation and Verification".

Immediate Input and Control Signals

A 4-bit dual in-line package (DIP) switch array provides immediate data input. These switches feed the fourth 74LS175 register (immediate register). When the load-immediate signal (LDIM) is asserted and clock pulses are generated, the DIP switch values are latched into this register. Later, if the IME signal is enabled (low), the contents of the immediate register are driven onto the bus, allowing constant values to be used in calculations.

Table 1 summarizes the main control signals and their corresponding functions. The load signals (LDA, LDB, LDIM, and OUT) cause the corresponding register to capture the bus value in the next clock pulse. The enable signals (AE, BE, IME, and ALUE) gate each source onto the bus (active low). The SUB' selects addition or subtraction in the ALU. By sequencing these signals manually (and providing a clock pulse by hand at each load), operations such as loading values and executing A + B or A–B can be performed.

All the control lines in Table 1 are driven by simple manual toggle switches or push-buttons rather than by any timing or sequencing logic, which means that the correctness of a given operation depends entirely on the operator asserting the right signal at the right moment and then pulsing the clock. While this places the full burden of sequencing on the user, it also has the pedagogical benefit of making every control decision explicit: nothing happens inside the CPU that was not directly and consciously triggered by a switch toggle, which is precisely the property that later motivates the control unit and instruction decoder extensions discussed in "*Conclusion*".

PROTEUS SIMULATION

Before any physical wiring was attempted, the complete CPU schematic was captured and simulated using the Proteus Design Suite (Figure 2). The simulation reproduces every IC used in the hardware build, including the 74LS175 registers, 74LS244 tri-state buffers, 74LS83 adder, and 74LS86 XOR gates, interconnected exactly as in the physical prototype, with virtual DIP switches, push-buttons, and LED indicators standing in for the corresponding front-panel hardware. This allowed every control signal sequence (register load, bus enable, ALU add/subtract selection, and output latch) to be exercised and checked against the expected logic levels at each node before any chip was placed on the breadboard, considerably reducing the amount of trial-and-error debugging required at the hardware stage. Figure 3 shows the full Proteus schematic capture of the CPU, including the labelled control-signal switches on the right-hand side and the 4-bit LED output banks for Register A, Register B, the Immediate register, and the final output register.

Discrepancies between the simulated and intended behaviors, for instance, an incorrectly wired enable line briefly driving two sources onto the bus simultaneously, were caught and corrected at this stage, because Proteus flags such contention conditions directly in the simulated logic trace. The design was carried over to the physical breadboard described in Section "*Proteus Simulation*" only after each test case in Table 2 was reproduced correctly in the simulation.

IMPLEMENTATION AND VERIFICATION

Hardware Setup

The CPU design was first validated using a digital simulator. The final hardware prototype used 74LS series ICs mounted on solderless breadboards. A regulated +5V supply powers all the chips. Figure 2 shows the prototype that was assembled. The ICs include multiple 74LS175s (for registers), 74LS83 (ALU), 74LS86 (XOR gates), and 74LS244 (tri-state buffers). The DIP switches and light-emitting diodes (LEDs) were mounted on the front panel. Manual toggle switches and a push-button drive the control inputs (LDA, LDB, etc.) and generate the clock pulse applied to the registers; no free-running clock generator was used in the current version.

Table 1. Key CPU control signals.

Signal	Description
LDA	Load bus data into Register A (on clock edge).
LDB	Load bus data into Register B (on clock edge).
LDIM	Load DIP switch data into Immediate register (on clock).
OUT	Load bus data into Output Register (on clock, displays on LEDs).
AE	Enable Register A output onto bus (active low).
BE	Enable Register B output onto bus (active low).
IME	Enable Immediate register output onto bus (active low).
ALUE	Enable ALU output onto bus (active low).
SUB'	ALU mode: 0 = A+B, 1 = A–B (with carry-in=1).

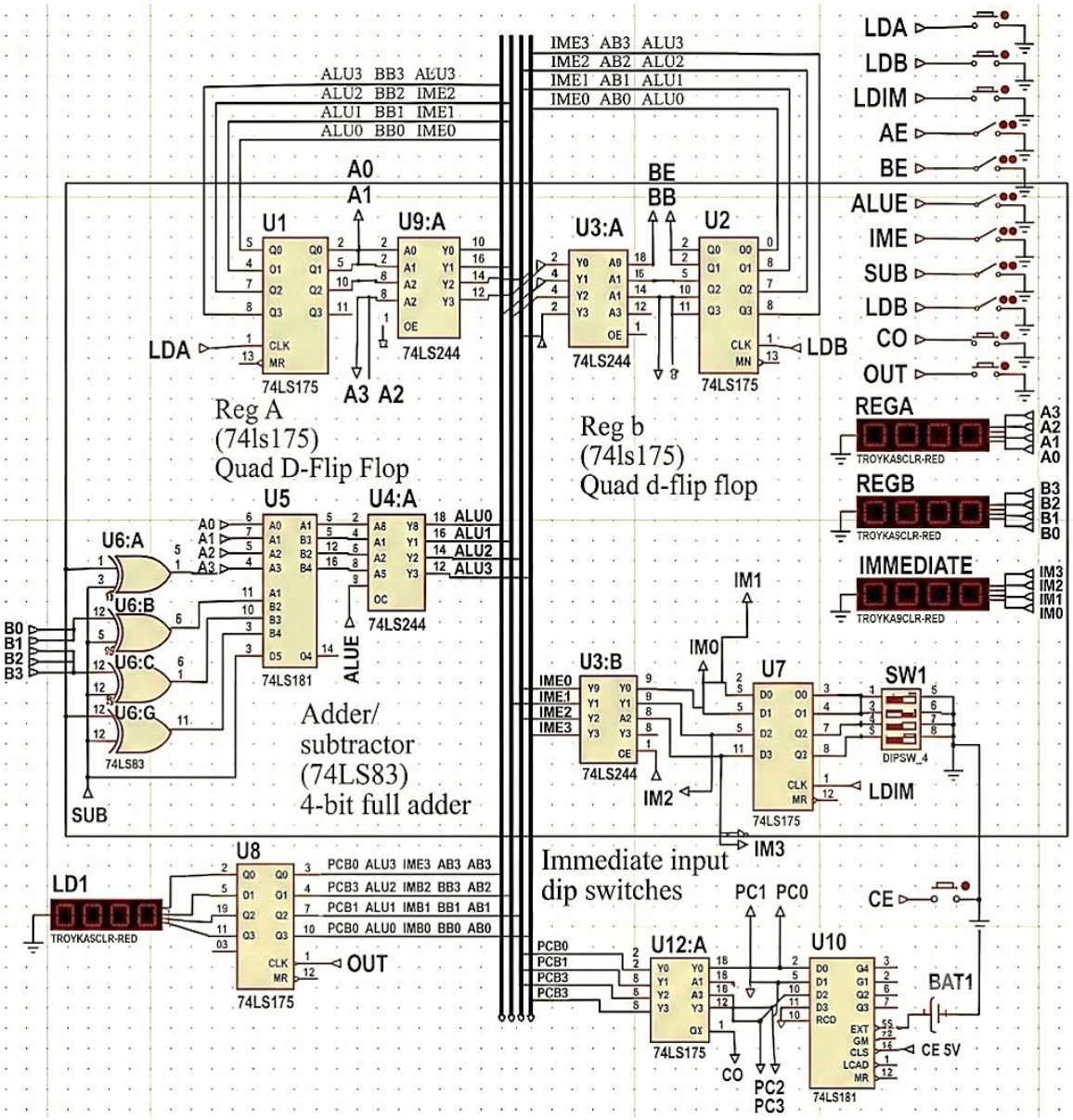

Figure 2. Proteus design suite schematic capture of the 4-bit CPU, showing the register, ALU, tri-state buffer, and control-switch layout used for pre-hardware simulation and verification.

Wiring is organized to minimize interference; power rails run along the board, and the data bus lines are arranged in parallel. Each register and ALU output is connected through a 74LS244 buffer to a common bus strip. Pull-up resistors ensure that the bus lines default to logic-1 when no driver is active. We strictly enable only one buffer at a time (as recommended for shared-bus tri-state systems [12] to prevent the bus contention. With careful wiring and the use of tri-state isolation, the system proved to be stable and reliable despite the dense layout.

Beyond the electrical layout, a considerable share of the implementation effort was devoted to the physical organization of the breadboard itself. Color-coded jumper wires were used consistently, with one color for bus lines, another for control signals, and a third for power and ground, so that a wiring fault could be traced visually rather than by continuity testing every connection individually. Chips belonging to the same functional module (for example, a register together with its associated tri-state buffer) were placed adjacent to one another on the board to keep the corresponding wiring runs short, which also had the practical benefit of reducing the parasitic capacitance and crosstalk discussed in Section 5.3.

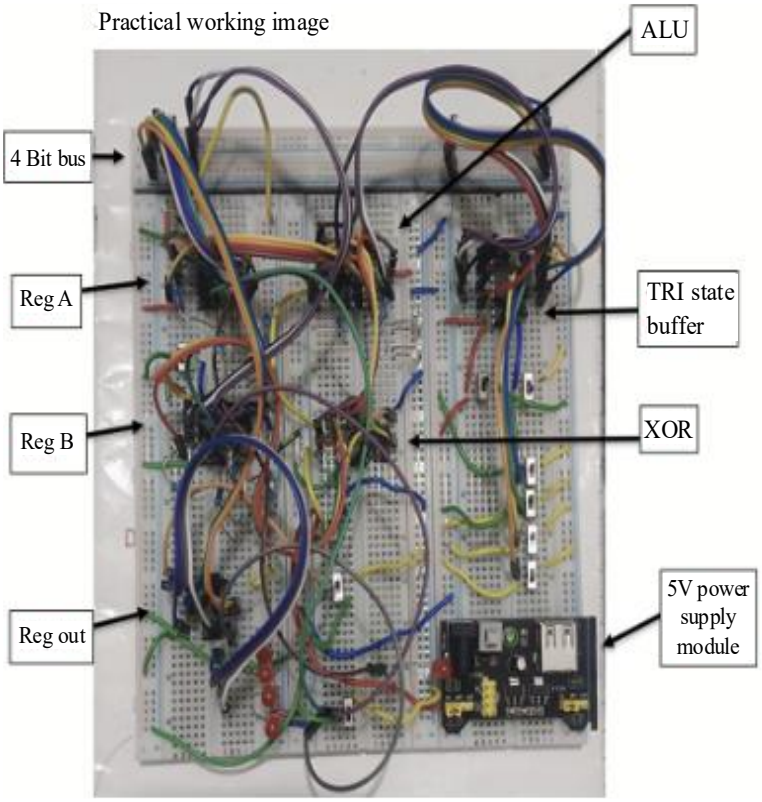

Figure 3. Breadboard prototype of the 4-bit CPU. Registers, ALU, and buffers are wired on the board. The front panel has DIP switches (right side) for immediate input and LEDs (bottom) for output.

Testing Methodology

Each test case was performed as a manual sequence: the DIP switches were set to the desired immediate value, the corresponding load signal was asserted, and a single hand-applied clock pulse was used to latch the value into the target register. This was repeated for Register A and Register B, after which the ALU mode was selected via SUB', the ALU output was enabled onto the bus (ALUE), and the result was latched into the Output Register (OUT) on a further manual clock pulse. The 4-bit value displayed on the LED array was then compared by hand against the expected binary result computed offline for that combination of inputs and operations. This procedure was repeated for the load, register-to-register, addition, and subtraction cases, which are summarized in Table 2.

To reduce the chance of operator error influencing the results, each test case in Table 2 was independently repeated at least twice on separate occasions, with the DIP switches and control toggles reset to a known default state between the runs. Only test cases that produced consistent LED outputs across repeated trials were recorded as passing. This repetition also served as an informal reliability check on the hardware itself: any intermittent contact fault on the breadboard, a common failure mode in solderless prototyping, would be expected to surface as an inconsistent result across repeated trials of the same test case.

RESULTS AND DISCUSSION

The breadboard CPU successfully performed all the intended operations. Table 2 summarizes the representative test cases covering register loading, addition, and subtraction, including edge cases such as a zero operand and an out-of-range (overflow) sum. In every case, the LED output matched the expected 4-bit result.

Rows 1–5 cover addition, including the all-zero case and a deliberate overflow (15+1), where the 4-bit sum wraps to 0 with the carry-out flag set, consistent with the unsigned 4-bit wraparound behavior.

Rows 6–9 show the subtraction in two's-complement form, including a case where the result borrows below zero (row 7) and a case of equal operands yielding zero (row 8). Across all tested cases, the hardware output matched the expected results, validating the correctness of the register, bus arbitration, and ALU logic under manual clocking.

It should be noted that this prototype is a deliberately reduced-scope adaptation of the general breadboard-CPU concept popularized in the hobbyist/educational space (e.g., 8-bit hand-clocked breadboard computers), scaled down to 4 bits, and currently limited to register-transfer and ALU operations only, without an automated control unit, program counter, or instruction memory. As such, it is not positioned as an improvement over or a direct comparison with more complete prior builds, but rather as a smaller, self-contained validation of the same core architectural principles.

Hardware Output Demonstration

To further illustrate the correct hardware operation, two representative photographs of the running prototype are included below. Figure 4 captures the board midway through an additional operation, with Register A and Register B loaded and the ALU output visibly enabled on the bus prior to being latched into the Output Register. Figure 5 shows a case selected specifically to demonstrate the carry-out behavior of the ALU together with the corresponding value latched into the Output Register, illustrating how the carry/borrow indicator and the 4-bit result are presented simultaneously on the front-panel LEDs.

In both figures, the LED patterns were cross-checked against the expected binary values computed offline (as summarized in Table 2), and in every photographed case, the observed pattern matched the expected result, providing direct visual confirmation of the numerical results reported in Table 2.

Design Limitations

The proposed CPU currently relies entirely on a manually applied clock pulse rather than a free-running clock generator, which limits its operational speed, repeatability, and ability to run any automated sequence of instructions. It also lacks instruction memory, control units, and pipelined execution. Breadboard wiring additionally introduces parasitic capacitance and noise, which would constrain the maximum clock frequency once automated clocking is introduced. These limitations motivate the future integration of a clock generator, program counter, and a hardware control unit.

Table 2. Observation table: sample test cases and results (binary, 4-bit).

S.N.	Reg A	Reg B	Op	Expected	Output	Status
1	0011	0101	ADD	1000	1000	Pass
2	0110	0011	ADD	1001	1001	Pass
3	1001	0110	ADD	1111	1111	Pass
4	1111	0001	ADD	0000*	0000*	Pass
5	0000	0000	ADD	0000	0000	Pass
6	0101	0011	SUB	0010	0010	Pass
7	0011	0101	SUB	1110	1110	Pass
8	1000	1000	SUB	0000	0000	Pass
9	1010	0101	SUB	0101	0101	Pass
10	DIP 1100 → A	–	LOAD	1100	1100	Pass

Note: *Decimal equivalents follow standard 4-bit unsigned binary (e.g., 0011 = 3, 1111 = 15). Row 4 (marked *) is a deliberate overflow case: 15 + 1 = 16, which wraps to 0000 with carry-out C = 1. Row 7 is a two's-complement subtraction (3 – 5), computed as A + NOT(B) + 1 = 0011 + 1010 + 1 = 1110 (i.e., −2 mod 16 = 14). Row 10 is a register-load test: a value set on the DIP switches is latched into the Immediate register and then transferred via the bus into Register A; the Output Register is used to display the loaded value for verification.*

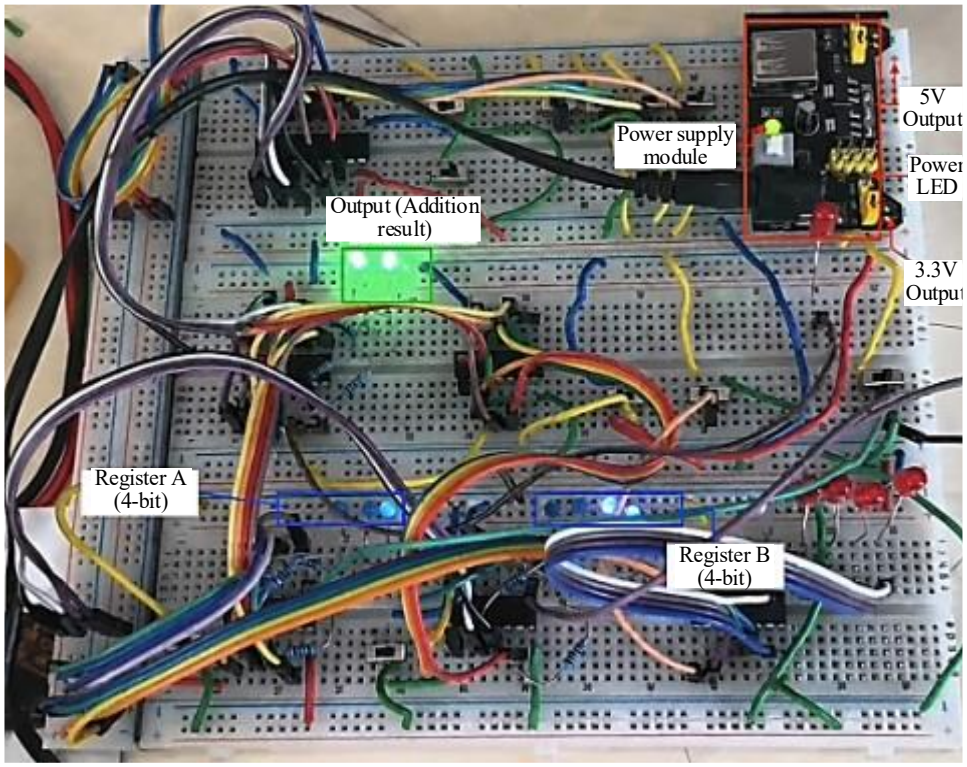

Figure 4. Hardware output during an addition operation: Register A and Register B hold the operands, and the ALU sum is visible on the bus/output LEDs prior to latching.

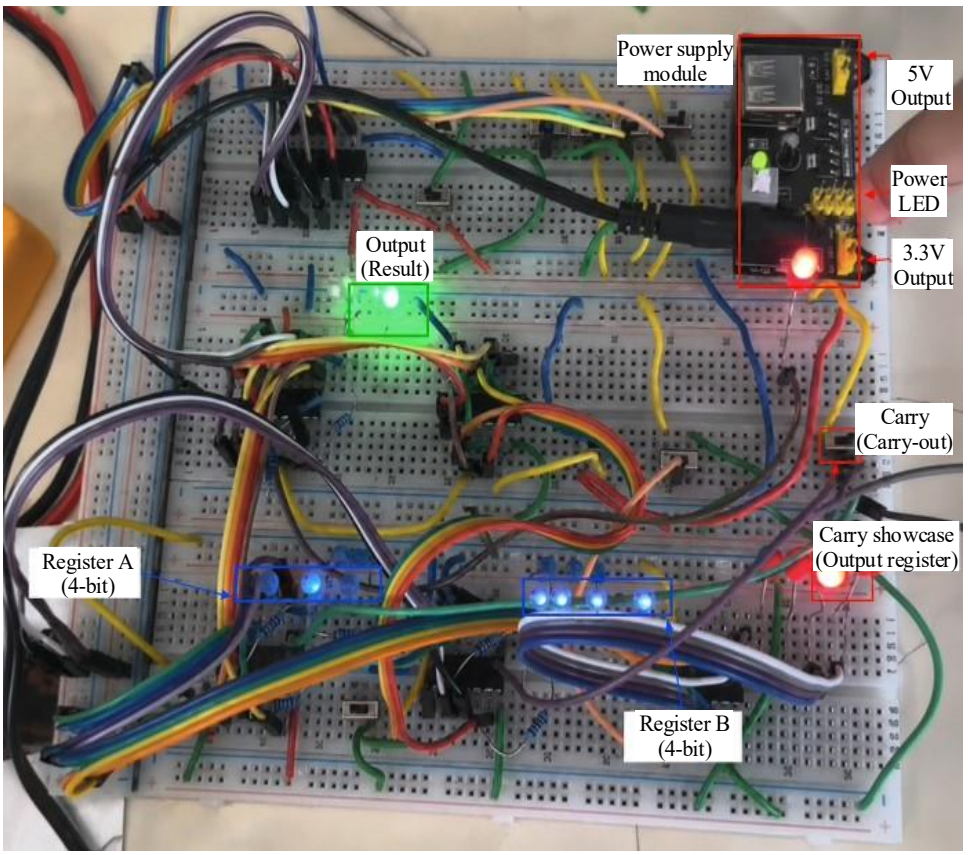

Figure 5. Hardware output showing the ALU carry-out indicator together with the corresponding 4-bit result latched into the Output Register, demonstrating correct carry/overflow signaling.

A further limitation, not immediately visible from the test results in Table 2, concerns the scalability of the manual testing methodology itself. Because every test case requires a human operator to set switches, assert control lines in the correct order, and read the LED output, the number of distinct operand/operation combinations that can realistically be exercised in a reasonable amount of time is small relative to the full input space of a 4-bit ALU (256 possible A/B combinations for each of the two operations). The nine arithmetic test cases and one load test case in Table 2 were therefore chosen specifically to cover representative and edge-case behavior (zero operands, equal operands, and overflow/underflow) rather than exhaustively verifying every possible input combination, and this exhaustive verification would become both feasible and desirable once an automated clock and simple test sequencer are introduced.

CONCLUSION

A 4-bit CPU was successfully built on a breadboard using discrete TTL logic. The design demonstrates core CPU component registers, an ALU, and a shared data bus working together under manual, hand-clocked control. The use of LEDs and switches made the operation transparent and educational in nature. The hardware results, validated across multiple loads, add, and subtract test cases, confirm that basic register-transfer and ALU functions can be implemented and verified with simple ICs. Pre-hardware validation in Proteus, followed by photographic confirmation of representative addition and carry-out cases on the physical board, together provide a consistent chain of evidence from schematic simulation to hand-clocked hardware operation, supporting the correctness of the design described in this study. This work provides a concrete platform for understanding the processor architecture and can be extended to automated clocking and more advanced CPU features in future research.

Several extensions of this prototype are possible. The most immediate extension is the replacement of the manual clock pulse with a free-running or step/run clock generator circuit, enabling a timed, automated operation instead of hand-triggered pulses. In addition, adding a program counter and ROM would allow automated instruction fetching and execution. An instruction decoder and control logic can interpret opcodes, enabling a sequence of operations without manual intervention. The introduction of flag registers (e.g., carry and zero) and branching permits conditional operations. The data path can be widened to 8 or 16 bits by using additional registers and ALU chips. Finally, a stable clock generator and state machine can control the instruction timing, transforming the system from a manually operated register/ALU demonstrator into an autonomous CPU.

In the long term, the authors intend to use this 4-bit design as a steppingstone toward the 8-bit extension outlined above, reusing the same register, ALU, and bus-arbitration subcircuits validated here while adding the control unit and program memory components identified as the principal limitations of the current design. Because every module in the present design was verified independently, first in the Proteus simulation and then on hardware, the authors expect that a substantial portion of the existing register, ALU, and tri-state buffering circuitry can be reused directly in this extended design, with the bulk of the additional engineering effort concentrated in the new control unit and instruction-sequencing logic rather than in re-deriving the already-validated datapath.

Acknowledgment

This work was carried out by Aditya Chauhan as part of an undergraduate project under the guidance of Prof. Himanshu Prajapati and Prof. Purvang Dalal, Department of Electronics and Communication, Dharmsinh Desai University, India.

REFERENCES

1. Rafiquzzaman M. Fundamentals of Digital Logic and Microcomputer Design. John Wiley & Sons; 2005 Jun 6.
2. Stallings W. Computer Organization and Architecture: Designing for Performance. Pearson Education India; 2016.

3. Sklar B. Digital Communications: Fundamentals and Applications. Pearson; 2021 Jan 27.
4. Hennessy JL, Patterson DA. Computer Architecture: A Quantitative Approach. Elsevier; 2011 Oct 7.
5. Tamir Y, Tremblay M. High-performance fault-tolerant VLSI systems using micro rollback. IEEE Trans Comput. 1990;39(4):548-54.
6. Yunten T, inventor. Yunten model computer system and lab kit for education. United States patent US 8,480,398. 2013 Jul 9.
7. Wang G. Bridging the gap between textbook and real applications: a teaching methodology in digital electronics education. Comput Appl Eng Educ. 2011;19(2):268-79.
8. Rabaey JM, Chandrakasan A, Nikolic B. Digital Integrated Circuits. Englewood Cliffs (NJ): Prentice Hall; 2002 Dec.
9. Null L, Lobur J. The Essentials of Computer Organization and Architecture. Jones & Bartlett Publishers; 2014 Feb 17.
10. Leach N. Digital Design: Principles and Practices. Michael D. Ciletti, M. Morris Mano. Quizlet; 2021.
11. Mano MM, Ciletti MD. Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog. 6th ed. [place unknown]: [publisher unknown]; 2018.
12. Wolford BJ, Hui CS, Venkatramani K, inventors; International Business Machines Corp, Motorola Inc, assignees. Tri-state bus contention circuit preventing false switching caused by poor synchronization. United States patent US 6,134,620. 2000 Oct 17.
13. Rinku R, Dahiya P. Advance high performance bus arbitration techniques (AHB): a state-of-the-art review. IOSR J VLSI Signal Process. 2017;7:51-6.
14. Nisan N, Schocken S. The Elements of Computing Systems: Building a Modern Computer from First Principles. MIT Press; 2021 Jun 15.
15. Nair VS, Nair AS. Exploring abstractions of a general-purpose computer by developing a mini breadboard computer. In: 2018 4th International Conference for Convergence in Technology (I2CT); 2018 Oct 27. p. 1-4.
16. Newman KE, Hamblen JO, Hall TS. An introductory digital design course using a low-cost autonomous robot. IEEE Trans Educ. 2002;45(3):289-96.
17. Rodriguez BJ. A minimal TTL processor for architecture exploration. In: Proceedings of the 1994 ACM Symposium on Applied Computing; 1994 Apr 6. p. 338-40. ACM.
18. Bailey C, Freeman MJ. A Java bread-board simulator: digital circuit simulation with an open-source toolset. IADIS Int J Comput Sci Inf Syst. 2010;55(1):13-25.
19. Doğan M, Öztoprak K, Tolun MR. Teaching computer architecture by designing and simulating processors from their bits and bytes. PeerJ Comput Sci. 2024;10:e1818.
20. Patterson DA. Reduced instruction set computers. Commun ACM. 1985;28(1):8-21.
21. Kumamaru N. Students experiment to construct a 4-bit CPU. In: 2022 IEEE International Conference on Mechatronics and Automation (ICMA); 2022 Aug 7. p. 152-6.

