Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

An Interactive Toolkit for Designing and Exploring Custom CPU Architectures

Ghulam Mustafa
Department of Computer Science, BIIT, Rawalpindi, Pakistan: Saeed Iqbal

Abdul Rehman
Department of Computer Science, BIIT, Rawalpindi, Pakistan: Saeed Iqbal

Sohaib Bin Waheed
Department of Computer Science, BIIT, Rawalpindi, Pakistan: Saeed Iqbal

Samiullah Istihaq
Department of Computer Science, BIIT, Rawalpindi, Pakistan: Saeed Iqbal

Saeed Iqbal (Corresponding author)
Department of Computer Science, BIIT, Rawalpindi, Pakistan: Saeed Iqbal
saeed@biit.edu.pk

ABSTRACT

Computer architecture is very difficult for users to understand due to its abstract nature. We present a CPU Architecture Toolkit, a software application that allows users to design their own CPU, specifying memory size, bus size, registers, addressing modes, flag registers, and instructions. The most distinctive feature of the toolkit is the level of control it provides, allow- ing users to define the behavior of instructions. This enables users to specify and write their own micro-operations and observe how the instruction executes and behaves. To the best of our knowledge, no existing toolkit or simulator provides this level of control and flexibility to the user when customizing CPU designs and defining micro-operations. This paper describes the system design, implementation, and educational value of the toolkit as an innovative aid for both teaching and research in the field of computer architecture.

Keywords— Computer Architecture, CPU Design, Instruction Set Architecture, Micro-operations, Educational Toolkit

1. Introduction

Computer architecture is a subject that is fundamental for students in the field of computer science. However, it is considered to be one of the most difficult subjects due to its complexity. The reason for its difficulty lies in architectural concepts, such as the

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

execution of instructions, the flow of data, control logic, and hardware-level interactions. Even though it is possible to understand these concepts in theory, it is still very challenging for students to properly observe the execution of instructions and their effects on memory and registers. This makes it difficult for learners to develop a proper understanding through normal teaching approaches, such as lectures.

Many simulators and educational tools have been developed to address the challenges faced when learning and teaching computer architecture (Burch, 2011; Larus, 2010, 2011; Vollmar and Sanderson, 2014). These existing tools allow users to observe the execution of instructions and perform experiments with predefined instruction sets. While these tools are undoubtedly valuable, they are limited and only provide predefined architectures and instruction behaviors. As a result, users can observe how a microprocessor works but they do not have the ability to design their own architectures or define custom behavior for each instruction.

Understanding computer architecture is very difficult just by observation. Experimentation with architectural components and instructions is necessary for learning. students and researchers would benefit significantly if they were able to modify and design their own pro- cessors and have custom instruction sets. Observing the changes caused by making these mod- ifications can greatly increase learning potential. Unfortunately, most existing simulators do not provide the flexibility and control for users to design their own architectures or to define the internal behavior of instructions.

In this paper, we are presenting a Computer Architecture Toolkit that is designed to over- come limitations mentioned above. The purpose of the toolkit is to provide users with control when designing their own architectures and instruction behaviors. The toolkit allows users to set architectural components like, memory size, bus size, stack size, general-purpose registers, special-purpose registers, flag registers, addressing modes, and instruction sets. Its most dis- tinctive feature is the ability for users to define the behavior of instructions by writing their own micro-operations in the form of java code and observing how step-by-step execution of instructions will affect the memory, registers, and flag registers. This level of customization enables deeper insight into the relationship between architectural design choices and processor behavior.

The major contributions of this manuscript are given below:

- The introduction of a flexible environment that allows users to design and customize architectures according to their learning or research needs and requirements.
- A mechanism for defining and executing instructions that have user-specified micro- operations, enabling detailed exploration of instruction behavior.
- To demonstrate the educational value of the toolkit as an effective aid for teaching and learning computer architecture concepts, as well as its potential applicability in research- oriented experimentation.

The remainder of this paper is organized as follows. Related work and existing computer architecture simulators are discussed in section 2. The overall system design and architecture of the proposed toolkit are described in section 3. The implementation details and key features of the toolkit are presented in section 4. The educational value

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

and potential applications of the toolkit are discussed in section 5. Finally, Section 6 concludes the paper and outlines directions for future work.

2. Related Work

A large number of toolkits and simulators have been developed that help user when teaching and learning concepts of computer architecture. These tools provide the ability to experiment and visualize the results of executed instructions. This helps users understand how instructions are executed by processors and how hardware sources are managed. Most of these existing simulators provide predefined architectures and offer limited control in terms of customization.

One simulator that is widely used is MARS (MIPS Assembler and Runtime Simulator) (Vollmar and Sanderson, 2014). It allows users to write and execute assembly programs that are written in MIPS while observing changes in memory and registers. Although MARS is useful and effective for teaching and learning basic instruction execution behavior. It relies on a fixed set of instructions along with a pre-defined architecture and offers very little and limited flexibility to customize.

Similarly, SPIM and QtSPIM also provide environments that simulate MIPS processors (Larus, 2010, 2011). These tools are not designed with the intention of exploring architectural concepts and processes. They can only be used for learning assembly languages. These tools also provide the ability to execute instructions in a step by step manner and track the changes caused by the instructions, but do not permit the users to alter the architecture itself or to define their own micro-operations for each instruction.

Another very popular tool used for learning is Logisim (Burch, 2011). It focuses on digital logic and basic CPU construction using logic gates like AND, OR, and NOT. Logisim allows its users to build simple processors visually. This makes is useful for users to understand the datapaths and control logic. The limitation is that it does not support high-level instruction set customization or to define custom micro-operations in a flexible and reusable manner.

CPUlator is another online simulator that provides the feature of multiple architectures like ARM and MIPS (Wong, 2018). It provides the users with a web based interface that is used to run programs written in assembly languages along with an interface for observing the state of the CPU. While CPUlator is capable of supporting multiple architectures, the limitation is still present. The architectures are fixed and pre-defined. There is also no proper way for users to modify architectural parameters or instruction behavior.

More advanced simulators such as GEMS and Simics are designed for research and perfor- mance evaluation (Intel, 2019; Multifacet Project, 2005). Although these tools provide highly detailed and configurable simulation environments, they also require a significant amount of expertise making it not suitable for beginners and undergraduate students. The complexity and steep learning curve of these tools make them impractical for learning computer architectural concepts.

In contrast to these existing tools, the proposed Computer Architecture Toolkit focuses in user-driven architectural design and instruction definition. Instead of limiting user to pre- defined architectures and instruction sets. The toolkit enables the users to specify

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

architectural parameter and define instruction behavior themselves through user-written micro-operations. This bridges the gap between existing educational simulators and research-level tools by pro- viding flexibility and also maintaining usability.

3. Methodology

In this section the methodology and system design of the proposed computer architecture toolkit will be discussed. The objective of the toolkit is to provide a flexible and interactive environ- ment where users have the ability and control to design custom CPU architectures and define behavior of instructions at a micro-operation level. The main focus of the methodology is on modular design, extensibility, and transparency of the execution of instructions so that both educational and experimental use cases can be supported.

3.1 System Overview

The Computer Architecture Toolkit is implemented as a software-based CPU simulator that contains the fundamental components of a processor. The system allows users to define the specifications of their architectures which can include memory size, bus size, stack size, num- ber and names of general purpose registers, addressing modes and instructions. The micro- operation of each instruction will be provided by the user in the form of Java code. Once an architecture is configured, users can write assembly code and observe how the step by step execution of instructions affect the internal state of the CPU which includes memory, stack, registers, and flag registers.

The toolkit is implemented using a modular approach, in this approach all of the major architectural components are treated as independent modules. This approach allows the user to be able to modify one component without affecting the other parts of the system. The overall flow of the toolkit consists of architecture configuration, definition of instructions, and their micro-operations, execution of the assembly program, and visualization of results through memory, registers, and flags.

subsectionArchitectural Configuration Module

The module for configuring architecture specifications provides users with the ability to design and customize the structural components of a CPU. This module allows users to specify parameters such as:

- The size of main memory
- The sizes of address and data bus
- Number and type of general-purpose registers
- Number and type of instructions
- Special-purpose registers such as program counter and stack pointer
- Flag registers for condition codes
- Supported addressing modes such as direct, indirect, and indexed addressing mode

These mentioned parameters define the basis of the architecture according to which the execution of instructions takes place. By providing the modification ability to the users, the toolkit enables experimentation with different architectural specifications and helps users to understand how such choices can influence the behavior of processor.

3.2 Instruction Set Design

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

The toolkit also allows users to define custom instruction sets instead of forcing them to use predefined ones. Each instruction is associated with a mnemonic, operation code, execution behavior, and number of operands, as well as addressing destination and source operands. This design enables the users to experiment with different instruction formats and explore how dif- ferent instruction designs can impact the flow of execution.

Unlike traditional simulators that restrict users to fixed and pre-defined instruction sets, the proposed toolkit treats instructions as configurable entities. This flexibility allows users to cre- ate simple instructions for the purpose of learning and teaching as well as complex instructions for experimentation and research.

3.3 Micro-Operation Definition

A feature of the proposed toolkit is that it provides the ability for a user to define his own micro-operation against every instruction. The micro-operations are provided by the user in the form of Java code. These micro-operations show how each instruction interacts with the architecture's components at the time of execution such as registers, memory, and flag registers.

Micro-operations are executed sequentially. This allows the users to observe the state of the CPU during execution of the assembly program. The approach makes the internal working of the processor transparent and helps users to get a better understanding of how high-level instructions are broken down into lower-level operations.

By allowing users to write and modify micro-operations themselves, the toolkit enables the users to get a detailed exploration of an instruction's behavior and also supports experimenta- tion through custom execution logic.

3.4 Execution Engine

The execution engine is responsible for validating assembly code, interpreting instructions, creating instruction formats, and executing the micro-operation of each instruction. To execute the micro-operation, a Java compiler is used as the micro-operations are provided as Java code by the user. The execution engine then uses the received results to update the architectural components that include memory, registers, stack and flags along with maintaining consistency across system state.

The execution engine of the toolkit supports both continuous and step-by-step execution of instructions. In step-by-step mode, the users are able to control the flow of execution through buttons that can either execute the next instruction or undo the execution of an instruction and change the state of architectural components. This feature is particularly useful for educational purposes, as it allows learners to track the changes caused by execution of each instruction in detail.

3.5 Visualization and Feedback

The visualization of architectural components provides feedback on the state of the CPU as the instructions are being executed. The contents of memory, registers, and flag registers, are displayed and the executed instruction is highlighted on the same screen providing more clarity to the learners.

These visual results and changes help in learning by enabling users to co-relate architectural behavior with the instruction's outcome after execution. The visualization module plays a very crucial role by helping users to develop a deep understanding of

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

the performed operations.

4 Implementation

The implementation of the proposed Computer Architecture Toolkit is described in this section of the paper. The system is divided is three different parts front-end presentation and user interface of the application, back-end control logic, and micro-operation execution through a Java compiler (Oracle, 2026). The modularity, maintainability, and extensibility are enhanced through this design as the suitable technologies are employed where they are most effective.

4.1 Overall System Architecture

The toolkit is divided into three main layers: a front-end presentation layer, a back-end control layer, and a micro-operation execution layer. Each layer its own responsibilities and the layers communicate with each other through structured interfaces.

The front-end of the toolkit is to provide the user interface and visualization, the back-end maintains the internal state of the simulated CPU and controls the validation and step by step execution of assembly code, The execution engine is responsible for compiling and executing micro-operations provided by the user against each instruction using a Java compiler. The benefit of this separation is that modifications in one layer do not cause problems in other layers. This allows users to change or update the micro-operation or even the core specifications of the architecture like memory size, stack size, bus size etc. This type of flexibility supports scalability and future enhancements.

4.2 Front-end Implementation

The front-end is implemented using Flutter (Google, 2026). It provides an environment that is interactive, responsive, and capable of rendering on platforms like Android devices, desk- tops, iOS devices, or even in the form of Web applications. In this layer, Users can configure specifications of the architecture, define instruction sets, write assembly programs, and observe results of execution in real time.

The front-end displays architectural components such as, memory, registers, stack, and the flag registers. It also highlights the instruction that is currently being executed when performing step by step execution. Users have the ability to load pre-written assembly programs, perform step by step forward and back execution of the instructions. The state of the CPU is captured after execution of each instruction and sent to the back-end so that it can it stored for evalua- tion. Flutter was selected because of its ability to create consistent interfaces across multiple platforms while also maintaining high responsiveness.

4.3 Back-end Implementation

The back-end development is done in the.NET Framework with the C Sharp language (Mi- crosoft, 2026). It acts as the core control unit of the toolkit, maintaining the internal state of memory, registers, stack, and flag registers. The back-end is responsible for the validation, parsing, decoding of instructions, execution of micro-operation against each instruction.

There are two modes that support execution, step by step and continuous. Additionally, the back-end also creates execution state of each instruction and enables users to undo

Online ISSN 3007-3197
Print ISSN 3007-3189

and rollback operations, this ensures state consistency during experimentation and debugging. Communica- tion with front-end is handled through structured interfaces. This allows the current state of the CPU to be relayed efficiently for real-time visualization.

4.4 System Architecture and Component Interaction

The communication between layers follows a structured procedure. The front-end sends a request to the back-end for actions such as architecture configuration, instruction definition, and instruction execution. The back-end receives the request, identifies the associated micro- operations, and invokes the Java-based execution engine, passing the current CPU state includ- ing of memory, registers, stack and flags.

The micro-operations engine compiles and executes the user-defined operations dynami- cally, and returns results to the back-end. The back-end updates the system state accordingly and sends the updated state back to the front-end for visualization, This procedure is summa- rized as:

Front-end (Flutter) → Back-end (.NET C Sharp) → Micro-Operation Engine (Java) → Back-end → Front-end

The design makes sure that modifications in micro-operations or the front-end components do not require any changes to the back-end, improving modularity and maintainability.

4.5 Micro-Operation Execution Workflow

A key contribution of this toolkit is the support for user-defined, runtime compiled micro- operations. Each instruction is associated with a micro-operation provided in form of Java code that is stored in the database against that instruction. At runtime, the back-end dynamically compiles these instructions into executable bytecode.

During the execution process, the back-end passes the micro-operation along with parame- ters provided by the user to the Java compiler. The Java compiler validates the micro-operation, executes the Java code, and returns the results to the back-end. The back-end uses these results to update the state of memory, registers, stack, and flag registers. The current updated CPU state is then transferred to the front-end for visualization.

This work flow provides flexibility to the users,. It allows them to experiment with custom instruction behavior without altering the core back-end logic. The separation of the execution layer ensures the safety of the system as well as maintainability.

4.6 State Management and Consistency

Controlled updates of the CPU's state are used to maintain consistency of the architecture. All of the modifications are done on temporary copies of the registers, memory, stack, and flags before being saved in the database. The benefit of using this approach is that the data can be prevented from corruption and users can rollback during stepwise execution.

The back-end maintains records of all the previously executed statements to enable undo and redo operations. The layers are synchronized in such manner that the front-end always reflects the true internal state of the CPU. This help to maintain accuracy during continuous or step-by-step execution.

4.7 Error Handling and Validation

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

Validations are performed when a user creates an architecture to ensure valid parameters are being set. Architecture definitions are checked for correctness, including register count and memory bounds. Instruction formats are validated for syntax and operand correctness, and the micro-operations are validated and compiled though Java compiler at runtime. Any compilation errors returned by the Java compiler are returned to the front-end to be displayed.

In-case of any runtime errors, such as invalid memory access, or division by zero are cap- tured but the back-end and the state of the CPU is reverted back to the last stable version. The errors are displayed on the front-end with proper description to help in debugging process. This approach helps to ensure both experimental safety as well as system reliability.

5 Results and Discussion

This section of the paper discusses the behavior and outcomes of the proposed Computer Ar- chitecture Toolkit based on the observations that were made on current implementation and testing. As the system is still being developed, the results that are presented here are supposed to reflect the functionality that has been defined, implemented, and validated at the current state of the project.

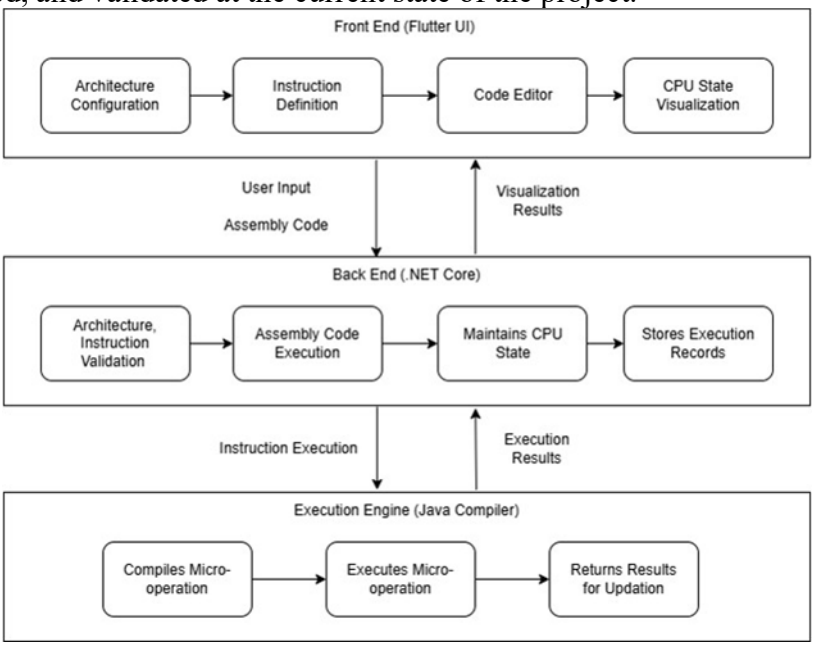

Figure 1: System Architecture of the Computer Architecture Toolkit. The front-end interacts with the back-end, which invokes the Micro-Operation Engine to execute user-defined instruc- tions.

5.1 Architecture Configuration Results

The proposed Toolkit at the moment successfully supports user-defined architecture configura- tions. Users are able to set architecture specifications such as the size of memory, bus, stack, and the number of registers and instructions along with other

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

specifications through the front- end interface. These definitions are stored in the database. This allows architectures to be reused and modified without complete redefinition.

5.2 Instruction Set Example

Table 1 provides an example of how instructions are defined in the system, reflecting the fields from the 'Instruction' model.

Table 1: Example Instructions Defined in the Toolkit

Mnemonics	Opcode (3-bit)	Addressing MID	#Operands	Destination	Micro-operation
ADD	000	1	2	R1	R1 = R1 + R2
SUB	001	3	2	R1	R1 = R1 - R2
LOAD	010	1	1	R1	R1 = Mem[Addr]
STORE	011	2	1	Mem[Addr]	Mem[Addr] = R1
NOP	100	1	0	-	-

During testing phase, the back-end validates all parameters of an architecture to ensure cor- rectness, which include bounds checking for instructions and registers count. In case of invalid configurations, a descriptive message is displayed. This prevents the creation of inconsistent or unstable architectures. The system enforces structural correctness at architectural level as it is essential for reliable instruction execution and educational use.

The ability to configure architectural parameters dynamically highlights the flexibility of the toolkit and differentiates it from other existing architectures that come with fixed instruction sets and architecture specifications.

5.3 Instruction Execution and Micro-Operation Results

The system allows users to define instructions with custom specifications and associates a user- defined micro-operation for each instruction. These micro-operations are provided in Java and are stored in the database against their respective instructions. At runtime, the back-end compiles these micro-operations dynamically with the help of a Java compiler.

Through testing it was confirmed that valid micro-operations are compiled successfully and are executed against the current state of the CPU. The results produced by the execution engine are returned to the back-end. These results then update memory, registers, stack, and flag registers. This demonstrates that the toolkit supports flexible instruction behavior without requiring changes to the core back-end logic.

The usage of compiled micro-operations, instead of interpreted logic, provides a realistic execution model. This allows users to experiment with low-level instruction behavior in a controlled environment.

5.4 Step-by-Step Execution and State Management

The Toolkit supports both continuous and step-by-step execution modes. In step-by-step mode, the state of the CPU is updated and captured after the execution of each instruction. This includes the values of registers, memory, stack, and flag registers.

An execution history is maintained by the back-end. It enables users to move forward

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 4 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

and backward during the execution of instructions. As an instruction is executed, a record is created and stored in the database. The rollback/step-back feature was verified during testing and the records in the database were used to ensure that the previous state could be restored without any corruption or loss of information.

This type of state management procedure is useful for debugging and educational analysis, as it allows users to observe how each instruction affects the CPU state after execution.

5.5 Error Handling and System Robustness

The system also includes validation and error handling mechanisms. Any type of invalid archi- tecture definitions, incorrect instruction mnemonics or operation codes, and micro-operations are detected during either validation or compilation. When these errors occur, the system re- turns descriptive error messages to the front-end to display. Oftentimes suggestions are also returned to help, and guide the users about invalid inputs.

Runtime errors such as invalid memory access or illegal operations are also captured by the back-end and returned in form of responses to be displayed. In case such errors occur, the state of the CPU is reverted back to the last stable version to ensure consistency and preventing sys- tem crashes. This behavior was observed during testing and confirms that the toolkit prioritizes experimental safety and reliability.

5.6 Educational Value and Observations

The proposed toolkit is still under development, but even with its current functionality, the importance and benefits for educational purposes are visible. The flexibility it provides for cre- ating architectures with custom specifications, instruction definitions, and the ability it provides for users to observe the change in the state of the CPU after the execution of each instruction makes it very suitable for learning computer architecture concepts which are often hard to grasp through traditional learning methods.

The separation of architecture configuration, instruction definition, and micro-operation execution allows learners to focus on individual aspects of the processor's design. This keeps users from being constrained by simulators that have fixed architecture designs. These observa- tions indicate that the toolkit can serve as an effective learning and experimentation tool when completed.

6 Conclusion and Future Work

This paper presented the design and partial implementation of a computer architecture toolkit that provides flexibility and enables users to define custom CPU architectures, instruction sets, and micro-operations for each individual instruction. The system uses a modular, multi-layer approach that enhances maintainability, extensibility, and provides experimental safety.

While the core back-end logic, execution engine, and architectural configuration modeling have been implemented and validated, several main features remain under development. Fu- ture work includes full implementation of the front-end developed through Flutter, support for interrupts and special-purpose registers. Additional advancements may include advanced vi- sualization, pre-defined architecture

Online ISSN 3007-3197
Print ISSN 3007-3189

templates, animation for execution of an instruction, and pre-written assembly programs for pre-defined architectures.

Once completed, the toolkit aims to provide a comprehensive educational and experimental platform for studying computer architecture concepts by directly interacting with and influenc- ing core components of a processor, as well as providing custom micro-operations.

References

Burch, C. (2011). *Logisim: Educational digital logic simulator* [Original version developed up to 2011]. https://www.cburch.com/logisim/

Google. (2026). *Flutter documentation*. https://docs.flutter.dev/

Intel. (2019). *Simics full-system simulator* [Full-system, multi-architecture simulator; commer- cial product]. https://developer.intel.com/simics-simulator

Larus, J. R. (2010). *QtSPIM: A mips simulator with gui* [QtSPIM is the actively maintained GUI version of SPIM]. https://sourceforge.net/projects/spimsimulator/

Larus, J. R. (2011). *SPIM: Mips simulator* [Legacy MIPS R2000/R3000 simulator]. https :
//spimsimulator.sourceforge.net/

Microsoft. (2026). *.net documentation*. https://learn.microsoft.com/dotnet/

Multifacet Project. (2005). *GEMS: General execution-driven multiprocessor simulator* [Part of the Multifacet Project at University of Wisconsin]. https://www.cs.wisc.edu/gems/

Oracle. (2026). *Java se documentation*. https://docs.oracle.com/javase/

Vollmar, K., & Sanderson, P. (2014). *MARS: Mips assembler and runtime simulator* [Last release: version 4.5, August 2014]. https://dpetersanderson.github.io/MARS/

Wong, H. (2018). *CPUlator: Web-based cpu simulator* [Supports MIPS, Nios II, ARMv7, and RISC-V simulation in browser]. https://cpulator.01xz.net/