

# UNR Sim: A Simulated Computer for Computer Engineering Education

Andrew M. Olson     Dwight D. Egbert     Frederick C. Harris, Jr.

Computer Science and Engineering  
University of Nevada, Reno  
Reno, NV, USA  
egbert@cse.unr.edu

## Abstract

UNR Sim is a decimal based Von Neumann architecture computing device simulator. Chronologically precedent simulators did not include factors we considered necessary or desirable, each in and of itself. Actualized by these inadvertent omissions, we decided to integrate all the desirable behaviors instantiated in these simulators while either neutralizing or negating their undesirable aspects. This design inspiration resulted in a robust computing device simulator suitable for pedagogical functionality at the introductory level with extensions available for more advanced students to continue their exploration of fundamental Von Neumann architecture computing systems.

**Keywords:** Computer Architecture, Interactive Teaching Tools, Simulation.

## 1 Introduction

“The very idea of a computer science program that did not provide students with an insight into the computer would be strange in a university that purports to educate students rather than to merely train them.”[8]

Electrical engineering students are taught the fundamentals of semiconductor behavior in order to understand how integrated circuits function. In a similar manner, computer engineering students need to understand the fundamentals of computing machinery behavior in order to understand how computers function. Concurrently and in conjunction with the understanding of fundamentals, is the preference for education over training. This implies a need for such *ab initio* internalization of the manner in which computing devices function.

The fundamental facet addressed by UNR Sim is the simplest form of the Von Neumann cycle, as shown below in Figure 1. Each element in this sequence is demonstrated through illumination of the conductive path connecting the two components in each step, *in nomine*, fetch, increment and execute.

From our beginning, the discussion initiates with an examination of historical and paradigmatic pedagogical computer simulations in Section 2. Following our discussion of previous implementations of pedagogical educational paradigms, we have a design discussion of UNR Sim in Section 3. Following this, we consider the details of the implementation of UNR Sim in Section 4. The paper concludes with a discussion of conclusions and future development work in Section 5.

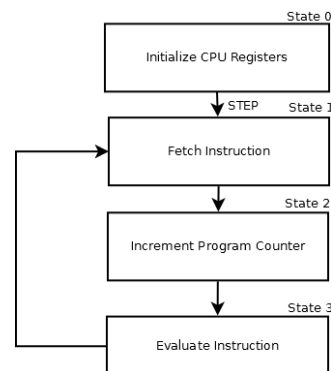


Figure 1: The von Neumann Cycle.

## 2 Background

Four distinct phyla make up the the taxonomy of computer simulators preceding UNR Sim. The most significant of these, as it is the precursor and most correlative to UNR Sim, is Simulated Computer II. The most primitive level of computer simulation, or in our context, most basic taxonomic phylum, consists of gate

and logic level simulators, in this document represented by Logisim. On a similar conceptual level, focusing on the Von Neumann cycle instead of simple hardware elements, is the Little Man Computer. Occupying the contrasting position, of representing primarily the computing hardware conceptualizations using hexadecimal notation, is Easy 4, and its succeeding upgrade, Easy 8.

The primary consequence of these distinctly variant approaches is a paucity of integration across these aspects, most decidedly the three points of decimal notation, focus on data movement in accordance with the Von Neumann cycle and finally, and not trivially, the ability to run the simulator on a wide range of modern computers without secondary emulation. UNR Sim addresses these issues both individually and *in toto*. Regarding the first, UNR Sim primarily functions in decimal mode, analogous to Simulated Computer II and the Little Man Computer, although unlike those, it is possible to display and input data in hexadecimal mode as well to show the relationship between the two. The Von Neumann cycle is displayed in UNR Sim by highlighting the wire bus connection between the two data involved in the particular step of the Von Neumann cycle, unlike Simulated Computer II which implies a more directional movement of the data rather than the data occupying a connection specified by addressing. Lastly, although important, arguably least critical, UNR Sim can be run on all three major platforms, Windows, Linux and OS X without specialized knowledge of emulators of obsolete hardware.

## 2.1 Simulated Computer II

In 1982, Carousel Software released Simulated Computer II for the Atari 800 and Commodore 64.[6] During this chronological period, software development, particularly game development, was wildly divergent, with numerous small companies creating a broad range of offerings. So while Simulated Computer II did win the award of “Best Microcomputer Software of 1982” from Learning Magazine, amid so many software offerings on the market of all quality levels, it was not surprising to find it did not find wide spread adoption as evidenced by the rarity of copies of Simulated Computer II.[6]

Notwithstanding such factors, Simulated Computer II is notable for its elegant implementation of the concepts underpinning the Little Man Computer and demonstrating how those concepts apply in the most fundamental level to a non-specific Von Neumann architecture computing device as show in Figure 2.

Unfortunately, while Simulated Computer II is an

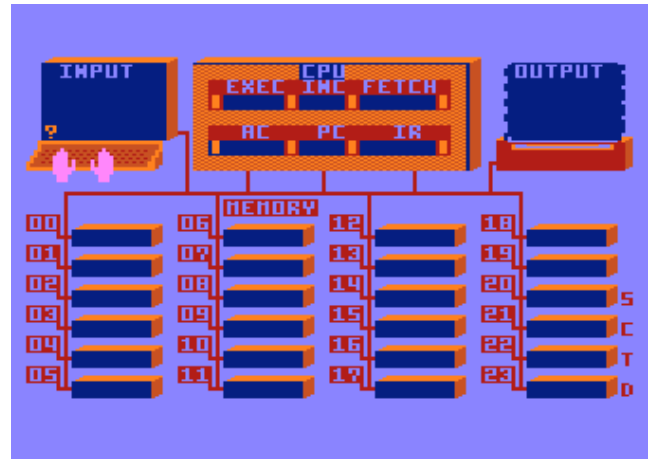


Figure 2: Simulated Computer II.[6]

excellent pedagogical tool, it is not as fully actualized as could be implemented today. The primary difficulty posed by Simulated Computer II is the fact it was written for what is now obsolete hardware, requiring functional knowledge of emulation software. However, as has already been shown, due to its obvious strengths, something similar to Simulated Computer II is certainly a desirable tool and so it is pedagogically valuable to create a computing device simulator with the positive qualities of Simulated Computer II without these negative factors.

## 2.2 Logisim

No discussion of simulation would be truly complete without examining at least one simulator of the most basic kind, *id est* a component level simulator like Logisim, shown in Figure 3. As stated by the creator of Logisim, Carl Burch, “Logisim can be used (and is used) to design and simulate entire CPUs for educational purposes.”[1] Acknowledging this as true, however, does little to ameliorate the fact such an endeavor can be an entire class, not merely a single session or module in a class focused on computational device design and architecture in general.

This extensive knowledge base required to apply Logisim to the pedagogical challenge of the fundamentals of computer architecture design, in particular the function of the Von Neumann cycle concomitant with simplified decimal interface, renders Logisim unsuitable for our purposes. In addition to this challenge, the fact that the input is all wires and signals places Logisim as perhaps a way to create a simulator at the conceptual level of UNR Sim, it is not inherently at that level by itself.

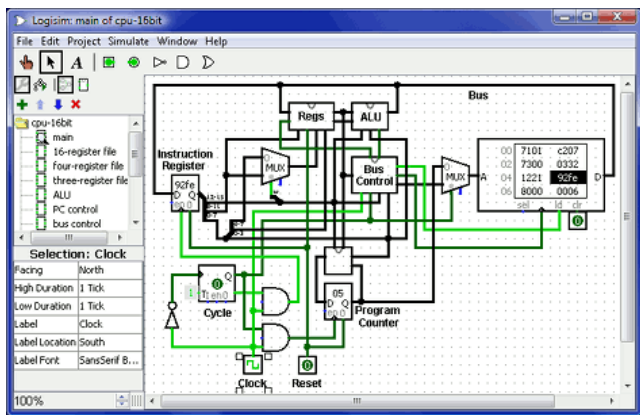


Figure 3: Logisim.[1]

## 2.3 Little Man Computer

The Little Man Computer, a conceptual computing device simulator was created in 1965 by Stuart Madnick of MIT.[10] The primary focus of the Little Man Computer is the fetch and execute cycle, ultimately an integral part of the basic Von Neumann cycle, albeit not even all the necessary components, as implemented by UNR Sim. Typical implementations of the Little Man Computer are instantiated in decimal, not hexadecimal, correlative with the minimalist ideal of this simulator. The Little Man Computer, shown in Figure 4 is the paradigmatic representation of the most minimal characteristics of the Von Neumann cycle.

In contrapositive effect, Little Man Computer is not a single implemented computing device simulator, but rather a conceptual exercise which has been instantiated in different ways creating inconsistencies in ultimate effectiveness for pedagogical application. Concurrently, Little Man Computer also obfuscates certain details inherent to the internal functioning of a typical central processing unit for the prototypical Von Neumann architecture computing device by means

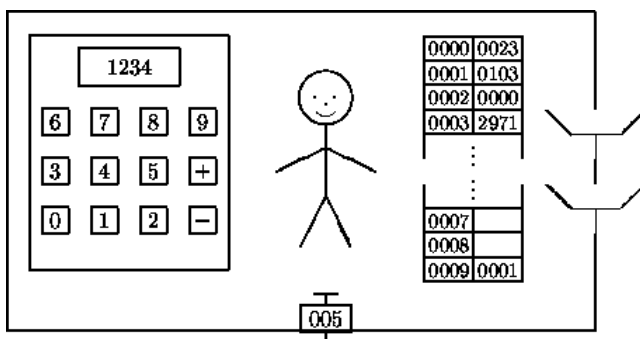


Figure 4: Little Man Computer.[10]

of oversimplification.

## 2.4 Easy4 Computer

In increasing order of complexity from Simulated Computer II and UNR Sim is Easy4 Computer, a Microsoft Windows based application included in *PC Architecture from Assembly Language to C*. [9] This application, as shown in Figure 5, is restricted to an accessible memory domain of sixteen locations due to its functional range of sixteen inputs for Easy 4. architecture.

Easy 4 and Easy 8 both use hexadecimal notation for all memory and functional behaviors. This is an additional complexity often seen to impede student assimilation of more critical concepts forming the fundamental basis of computational machine behavior.

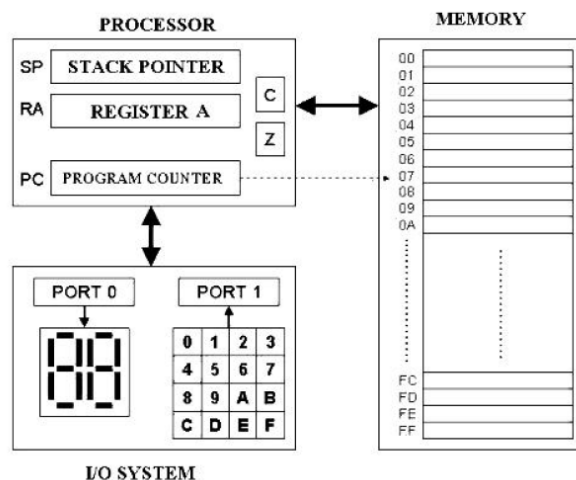


Figure 5: Fundamental EasyX Architecture.[9]

## 3 UNR Sim Components

The cardinal aspect of computer architecture to be taught by UNR Sim is the von Neumann cycle, with the use of decimal notation, connection highlighting and accessibility substantive to this pedagogical focus, as in Figure 6.

The first elements of UNR Sim were based on the behavior and appearance of Simulated Computer II, with its components from Little Man Computer and replicated later in Easy 4. Visual characteristics were updated for current student expected environmental experiences and also kept simple, to reflect the fundamental nature of the lessons under study. Immediately accessible controls and codes were left as similar to the original Simulated Computer II, however a few

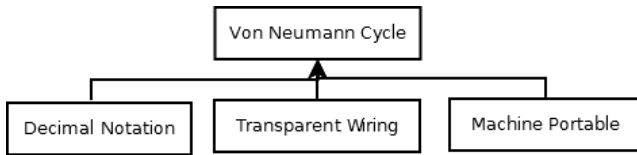


Figure 6: Fundamentals Contributing to the Von Neumann Cycle.

were discarded as excess to requirements and many were added although not included in the primary documentation. The animation of data movement was changed from the disingenuous slowly traveling light to a highlight of the entire connection, showing the nature of wire exclusivity. Lastly, a sound generator was adapted from Simulated Computer II to give students something more responsive to interact with.

### 3.1 Visual Characteristics

UNR Sim in its basic configuration is shown below in Figure 7. At the top left of the display is the screen. At the bottom left of the display is the CPU. At the top right of the display are the control buttons. At the bottom right of the display are the memory banks. The lines connecting the screen, CPU and memory are representative of the wire connecting these three simulated components of an entire computing device.

The first thing we removed was the teletype output, replacing it with a single screen for input and output as found on modern computing systems. Students often found the dual system confusing, as many had never experienced a teletype machine. The colors were chosen to accentuate the three primary interface means, *in nomine* the screen, the CPU and the memory. The control buttons were drawn in black so as to be the same as the wires, in that they are both representations of concepts more than exact representations of components in the machine.

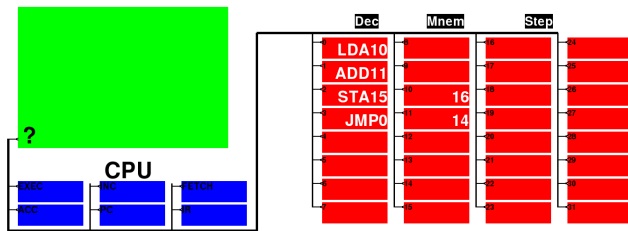


Figure 7: Fundamental Mode of UNR Sim.

### 3.2 Commands, Mnemonics and Errors

Arithmetic systems are rarely integrated fluidly into mathematical education to allow naive computer science and engineering students to intellectually assimilate material such as the binary and hexadecimal systems used in computer science and engineering without difficulty. This dual assimilation issue, that of the difficulty in comprehension of both the von Neumann cycle and hexadecimal notation concurrently, is avoided by the primary numerical system used in UNR Sim (as it had been in Simulated Computer II) is decimal. This is a continuation of the decimal arithmetical system used by the Little Man Computer.[10] In addition, the top left button in Figure 7 toggles the display and input between hexadecimal and decimal.

The commands selected for UNR Sim, as shown in Table 1, are very similar to those used by Simulated Computer II. The fundamental commands to load and run code are unchanged. However, the ability to reset the simulation such that it starts with a cleared CPU but the same memory values was added as well as the ability to clear the memory completely. We did, however, remove the command input to change between operation codes and mnemonics, replacing it with a button located above the memory.

Command	Command Effect
RUN	Starts Program Running at 00
RUNxx	Starts Program Running at xx
RNSxx	Starts Program Running at Speed xx
NEW	Clears All Memory
RST	Resets the CPU
LOAD	Starts Loading at Location 00
LOADxx	Starts Loading at Location xx

Table 1: UNR Sim Control Commands

All the mnemonics were continued from Simulated Computer II and many were added. However, the equivalent operation codes were maintained between the earlier program and the current program, with the added mnemonics added sequentially after the original mnemonics. A table of operation codes and mnemonics that are shared between UNR Sim and Simulated computer are shown in Table 2

Error messaging was updated due to the loss of the output teletype as well as to reflect the more typical behavior of modern computer systems to respond with error codes at the time of failed execution of a command. Table 3 displays the formally documented error codes.

OpCode	Mnemonic	Function
001	LDAxx	LoaD ACC with xx
002	STAxx	STore ACC in xx
003	ADDxx	ADD xx to ACC
009	JMPxx	JuMP to xx
010	SKZ	Skip Next if ACC = 0
011	SNZ	Skip Next if ACC != 0

Table 2: Example OpCodes and Mnemonics

Error Code	Error
OVF	Overflow
UDF	Undefined Location
INV	Invalid Instruction
DVZ	Divide by Zero
NAS	Not a Sound

Table 3: UNR Sim Error Messages

### 3.3 Wire and Bus Displays

The primary display mechanism used by UNR Sim is to highlight the entire wire and bus connection between the two points of interest for the particular stage of the Von Neumann cycle in particular. The three fundamental steps displayed are when the simulated computer fetches the instruction. The second step displayed is when the computer increments the program counter. The final step under examination is when the computer executes the function, see Figure 7. Each of these steps is shown as a change in color of the whole connection.

The practice of highlighting the entire wire and bus connection was implemented to echo the step function of the Von Neumann cycle. For each step of the cycle, one connection is highlighted.

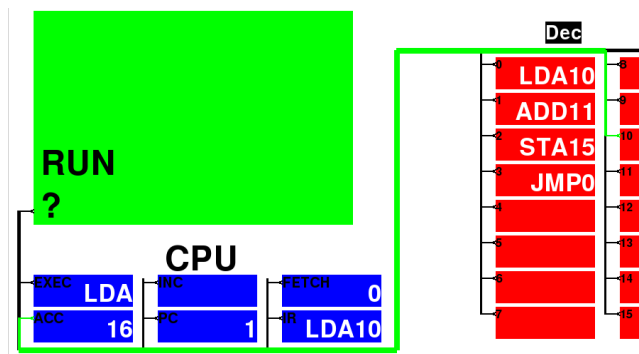


Figure 8: Bus Highlighting for **LoaD** Accumulator

### 3.4 Audio Generator

The original Simulated Computer II had a simple sound generator capable of sounding discrete tones a single time. UNR Sim generates a sine wave of a single frequency with programmable duration and volume. Unlike normal functions, when the tone is done playing, the simulator progresses to the next step in the program. This was done so that students could program the simulator to reproduce a melody, by setting the run time to maximum speed to minimize delay between tones.

The purpose of the audio generator is to give students an application upon which to apply their programming skills. In a similar manner to Simulated Computer II, the audio generator in UNR Sim can be used for programming assignments.

## 4 Implementation

The primary tool used to design UNR Sim was Python 2.7.9[5] and the Pygame 1.9.1[4] library. These were chosen because both Python and Pygame are transportable across all major operating systems, Python is the sixth most popular programming language by normalized rating,[3] and is normally packaged with the code in plain text so it is easily modifiable by end users.

The visual layout was organized by percent of full scale as opposed to on a per pixel basis. This way all of the individual components can be referenced by a static spatial orientation and location scheme. Using this methodology, every location on the screen is referred to on a scale from zero to one hundred. Pygame was used to display all the graphic and text elements in an independent window.

Commands and mnemonics are implemented with individual logic statements, to allow for ease of replacement or modification. No command or mnemonic is dependent on any other, to prevent the possibility of cascading errors causing a system wide failure. The commands were selected from the original Simulated Computer II commands. The basic mnemonics and opcodes were transferred from Simulated Computer II, however the advanced mnemonics were adapted from the Motorola 68000 opcodes.[2] Error codes were selected from Simulated Computer II and observation of common user errors.

Wire highlighting was accomplished by locating all nodes of the wire and bus combination used to carry the signal between the two points and then overwriting that wire and bus combination with another color. This remains highlighted until the next step of the Von

Neumann cycle is taken. This was done to explicate the nature of a wire connection, that being it is intact no matter where the signal is on its transmission between two points.

The tone for the sound generator is created from a discrete array representation of a sine wave at a specific frequency. These discrete values are then sent through the Pygame sound generator to the sound system on the device UNR Sim is currently run on.

All the keyboard handling was implemented in Pygame as well. In addition, mouse location and status was implemented in Pygame, particularly for the control buttons.

## 5 Conclusion and Future Work

### 5.1 Conclusion

It has long been a proverb that one is unable to construct a building descendant from the rafters. In the context of computers, this inherently implies a fundamental applicable knowledge and understanding of the foundation and seminal basis for computer functionality. For von Neumann architecture computing devices, this mechanism is the von Neumann cycle. UNR Sim explicates this cycle after excising all possible extraneous distracting complications, such as hexadecimal numeration and recondite user experience abstractions.

### 5.2 Future Work

Future work for the Simulated Computer 2015 project includes the addition of non-core features; *exempli gratia*, the implementation of a turtle graphics module to allow students to draw pictures or methods to save and load external code.

Other types of future work include the migration onto a more general purpose platform such as Unity,[7] giving access to smart phones, tablets, consoles, web platforms and other operating systems and hardware.

## References

- [1] Logisim. <http://www.cburch.com/logisim/index.html>. Accessed: 2015-04-03.
- [2] Motorola 68000 Family Programmer's Reference Manual. [http://www.freescale.com/files/archives/doc/ref\\_manual/M68000PRM.pdf](http://www.freescale.com/files/archives/doc/ref_manual/M68000PRM.pdf). Accessed: 2015-04-27.
- [3] Programming Language Popularity. <http://langpop.com/>. Accessed: 2015-04-27.
- [4] Pygame; Modules for Writing Games in Python. <http://pygame.org/news.html>. Accessed: 2015-04-28.
- [5] Python programming language. <https://www.python.org/>. Accessed: 2015-04-28.
- [6] Simulated Computer II atari mania. [http://www.atarimania.com/game-atari-400-800-xl-xe-simulated-computer-ii\\_4681.html](http://www.atarimania.com/game-atari-400-800-xl-xe-simulated-computer-ii_4681.html). Accessed: 2015-04-03.
- [7] Unity. <https://unity3d.com/>. Accessed: 2015-04-27.
- [8] Why Do We Teach Computer Architecture? <http://alanclements.org/teachingarchitecture.html>. Accessed: 2015-04-28.
- [9] D. Hergert and N. Thibeault. *PC Architecture from Assembly Language to C*. Prentice Hall, 1998.
- [10] W. Yurcik and H. Osborne. A crowd of little man computers: visual computer simulator teaching tools. In *Simulation Conference, 2001. Proceedings of the Winter*, volume 2, pages 1632–1639 vol.2, 2001.