6-22-2008

A Simulated Mano Machine An Novel Project For Computer Architecture Class

Vicky Fang
Cedarville University, vfang@cedarville.edu

Clinton E. Kohl
Cedarville University, kohlc@cedarville.edu

AC 2008-80: A SIMULATED MANO MACHINE--AN NOVEL PROJECT FOR COMPUTER ARCHITECTURE CLASS

Vicky Fang, Cedarville University
assistant professor

Clinton Kohl, Cedarville University
associate professor

A Simulated MANO Machine -- A Novel Project for Undergraduate Computer Architecture Class

Abstract:

Hands-on experience and visualization are both crucial to enhance undergraduate engineering education. This paper will describe a novel project that we feel meets both of these key elements for a first undergraduate computer architecture class. Instruction level simulation, though helpful, does not expose students to the hardware behavior or the internal instruction behavior. Likewise, FPGA simulation alone will not provide a good real-time visualization of the many digital signals which make up the microprocessor hardware. To avoid such drawbacks, we designed a project that requires each student to implement a 16-bit general-purpose computer on a real time digital logic simulator named Cedarlogic.

Students are given an instruction set specified in the textbook and a short assembly level test program. Students will: 1) build the entire computer hardware using the Cedarlogic simulator from fundamental logic gates; 2) write an assembler to translate the test program into binary code; 3) load the program into the memory of their computers; and 4) run the test program on their hardware. Cedarlogic is a unique real-time digital logic simulator designed by six of our senior engineering and computer science students for their capstone project over two successive years. In Cedarlogic, a logic high signal is shown in red, a logic low signal is shown in black, while high impedance is shown in green. As a result, when a project is working correctly students can actually watch all the internal signals within the computer "dancing" with the clock. Students can watch how the address buses change, how the data is latched, and how the ALU calculates... It is a real-time simulation, an experience which uncovers the mysterious veil of the computer. The students are excited to watch their computer executing the test program, clock cycle by clock cycle. It is truly an enlightening experience for the undergraduate computer architecture student.

Introduction

Computer Architecture is a fundamental course in every computer engineering curriculum. Two important goals of the computer architecture class are to give the students a good understanding of:

1. how digital hardware is used in the construction of a computer, and;
2. how each instruction propagates through the microprocessor.

These goals are especially important for the first exposure of the undergraduate student to computer architecture. Without a good understanding of these basics, all the student will receive will be some vague terminologies and theories. As a result, it will be hard for them to further develop and to receive advanced topics in computer architecture and apply them to the real world.

To fulfill the above goals, many schools have developed projects to give their students hands on practice in these areas. These projects have a variety of forms. One approach is to use computer instruction simulators. For instance, the SPIM simulator will read and execute assembly language programs written for MIPS machines; the emu86 will run x86 instructions. Projects developed by using these kinds of simulators will expose the student to the instruction level of the computer architecture. The student is able to watch data moving among registers and memory in the instruction cycle level within the simulated windows. The drawback is that the student is not exposed to the hardware behavior and inner-instruction operations.

Another recent approach is to implement a simple processor on a FPGA board. The student can design digital logic blocks and put them together to form a small microprocessor. The implemented microprocessor can be burned and tested onto an FPGA board. Indiana University, for example, requires students to implement a simple RISC processor on a XSA-100 FPGA board [1]. Texas A&M University also adopted the FPGA implementation in their microprocessor class to let the students experience the design process [2]. Although it is advantageous to get students involved with computer hardware design, there are some limitations to this kind of project. First, it requires students who are taking computer architecture to be proficient in VHDL or Verilog, which is the language commonly used in programming an FPGA board. Second, with the FPGA board, only signals that are mapped to the I/O pins can be observed on an oscilloscope or logic analyzer. Most FPGA boards also provide the ability to display information on a VGA Monitor. The problem is that even in a very simple RISC microprocessor there are many signals and buses. With a limited screen size on the scope or computer monitor, it is hard to observe the various signals simultaneously. Finally, since the student cannot observe very many signals at once, it is harder to debug their designs. As a result, FPGA implementation can give the student a good hands-on experience on the hardware and machine cycle level simulation, but it does not provide good visualization of the signals, buses and hardware elements, and visualization is very helpful for the student.

The authors of this paper have developed a novel project that fulfills both the need for hands on experience and visualization by using the Cedarlogic simulator [3]. We have received very positive feedback from the students who have completed this project and believe that it has tremendous educational value.

Course Overview

The Computer Architecture course at Cedarville University is a junior level offering with prerequisites of Digital Logic Design and Microprocessors. Since this represents a student's first computer architecture class, the primary goal is to give her a solid foundation in the basics. The student will then be well prepared for the Advanced Computer Architecture class that is offered in their senior year.

The textbook for computer architecture is "Computer System Architecture," by M. Morris Mano (Prentice Hall) [4].

Project Background

The first four chapters of the textbook introduce digital logic circuits that are commonly used in computer organizations. These chapters give the student the desire to know how to connect all the isolated digital pieces and make them function together.
Chapter five of the textbook leads the student through the entire design process of a small computer system, which includes instruction set selection, instruction format design, data path

design, instruction cycle analysis and control signals derivation. Our project is to actually implement this small computer system on the Cedarlogic digital logic simulator.

System Specification

The computer to be implemented is a 16-bit wide general-purpose computer. The system specifications are shown in Figure 1 below:

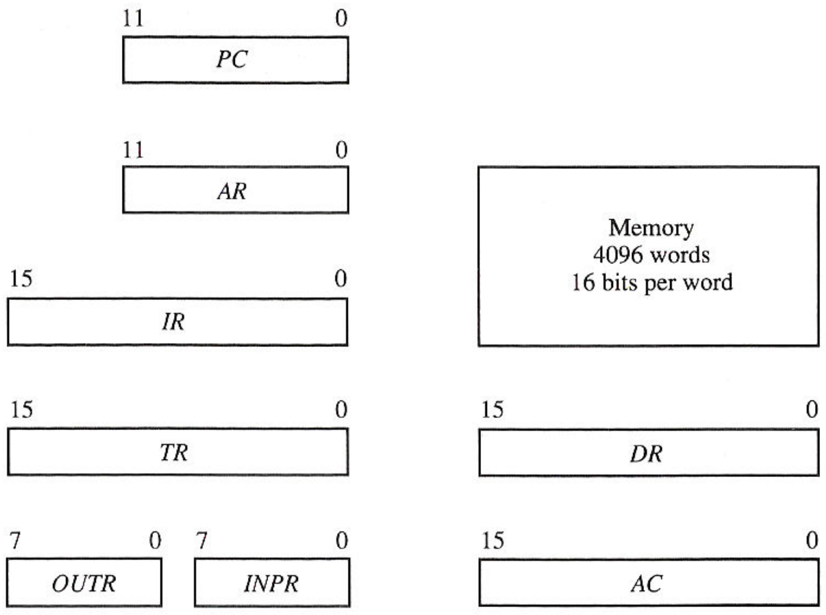


Figure 1. Mano Machine System Specifications with Width Marked

A 4k by 16 bits wide memory space will require a 12-bit wide address bus. This explains why the PC (program counter) register and AR (address register) are 12 bits wide. 16 bit wide registers are used for DR (data register), TR (temporary register), AC (Accumulator) and IR (instruction register). 8 bit wide registers are used for I/O operations (OUTR and INPR). A common bus will be used to connect them together as shown in Figure 2.

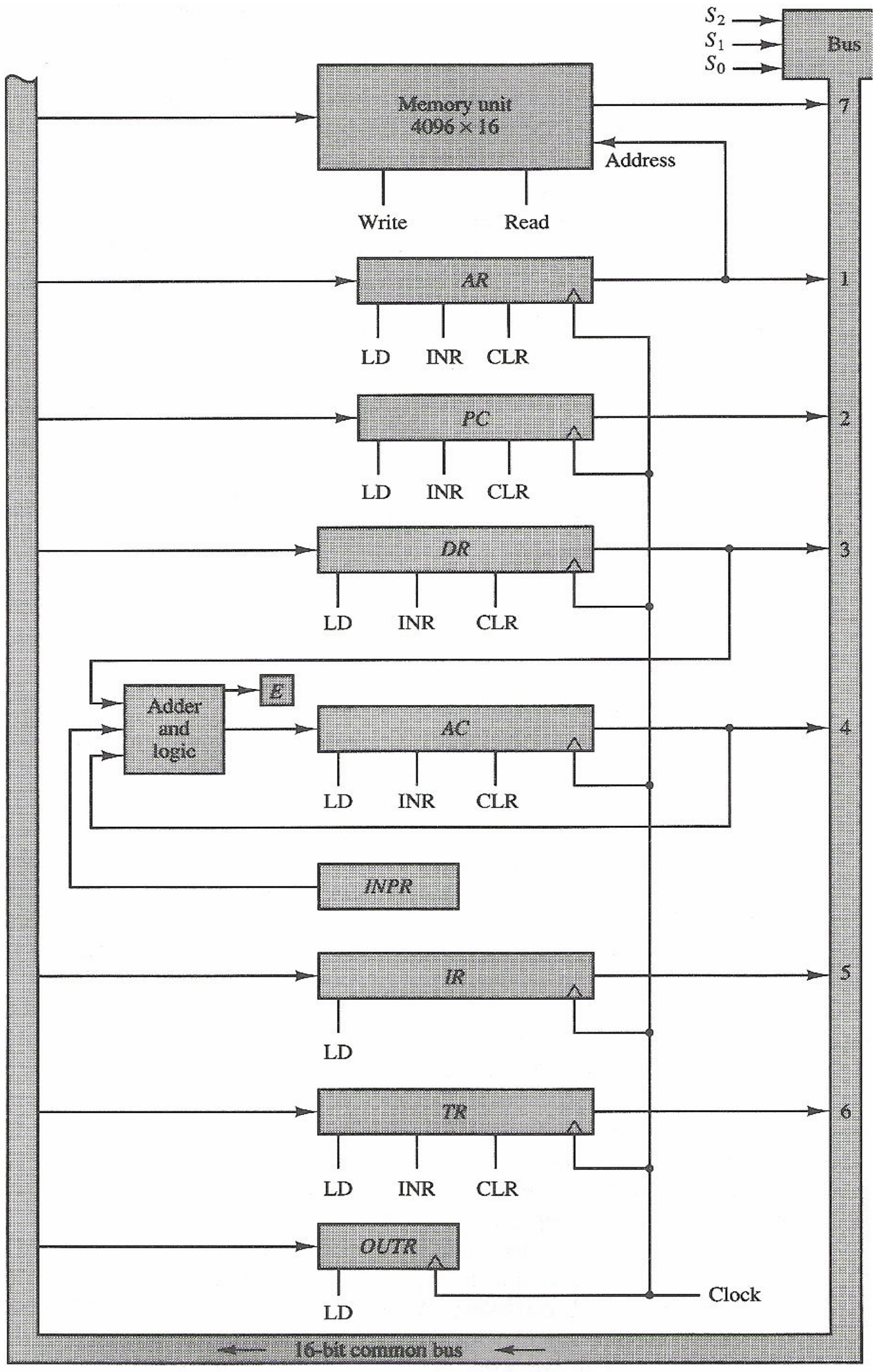


Figure 2. Data Path of Mano Machine

The instruction format is defined in Figure 3.

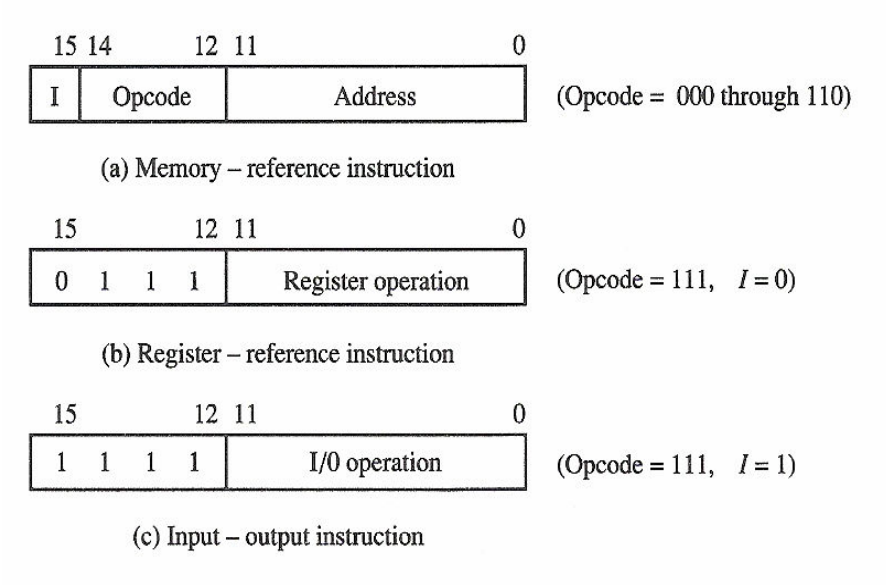


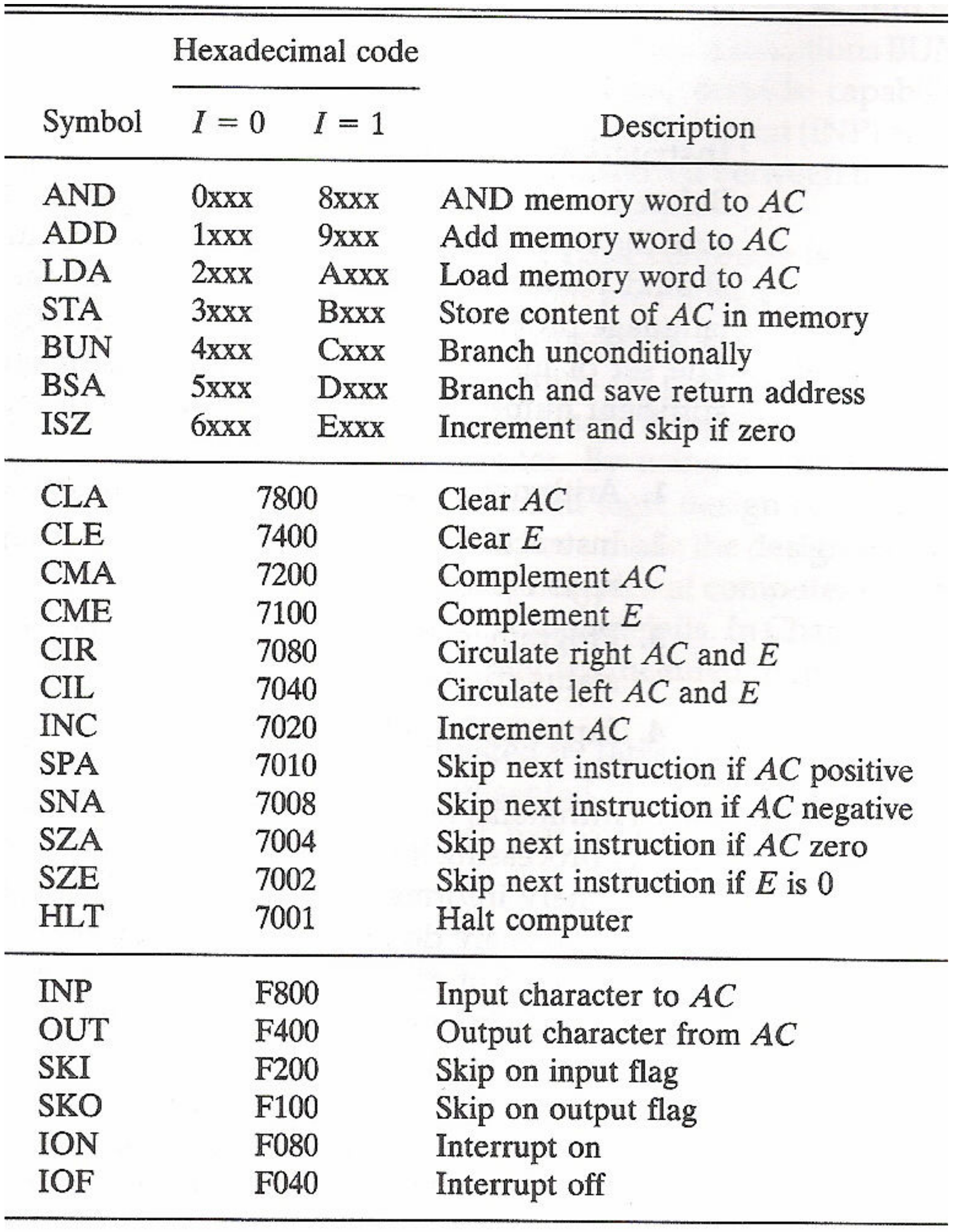
Figure 3. Mano Machine Instruction Set Format.

Bit 12-14 are used to indicate if the instruction is a memory reference instruction or not. If bit 12-14 are 111, the instruction is either a register-reference instruction or I/O instruction. Bit 15 is used to further distinguish each one. If bit 12-14 are not 111, the instruction is a memory-reference instruction. Bit 15 is used to indicate indirect or direct addressing mode for memory reference instructions. A decoder will be used to further decode which type of instruction it is.

Table 1 lists the entire instruction set that is to be implemented. The binary code of each instruction complying with the instruction format defined in Figure 3 is also listed.

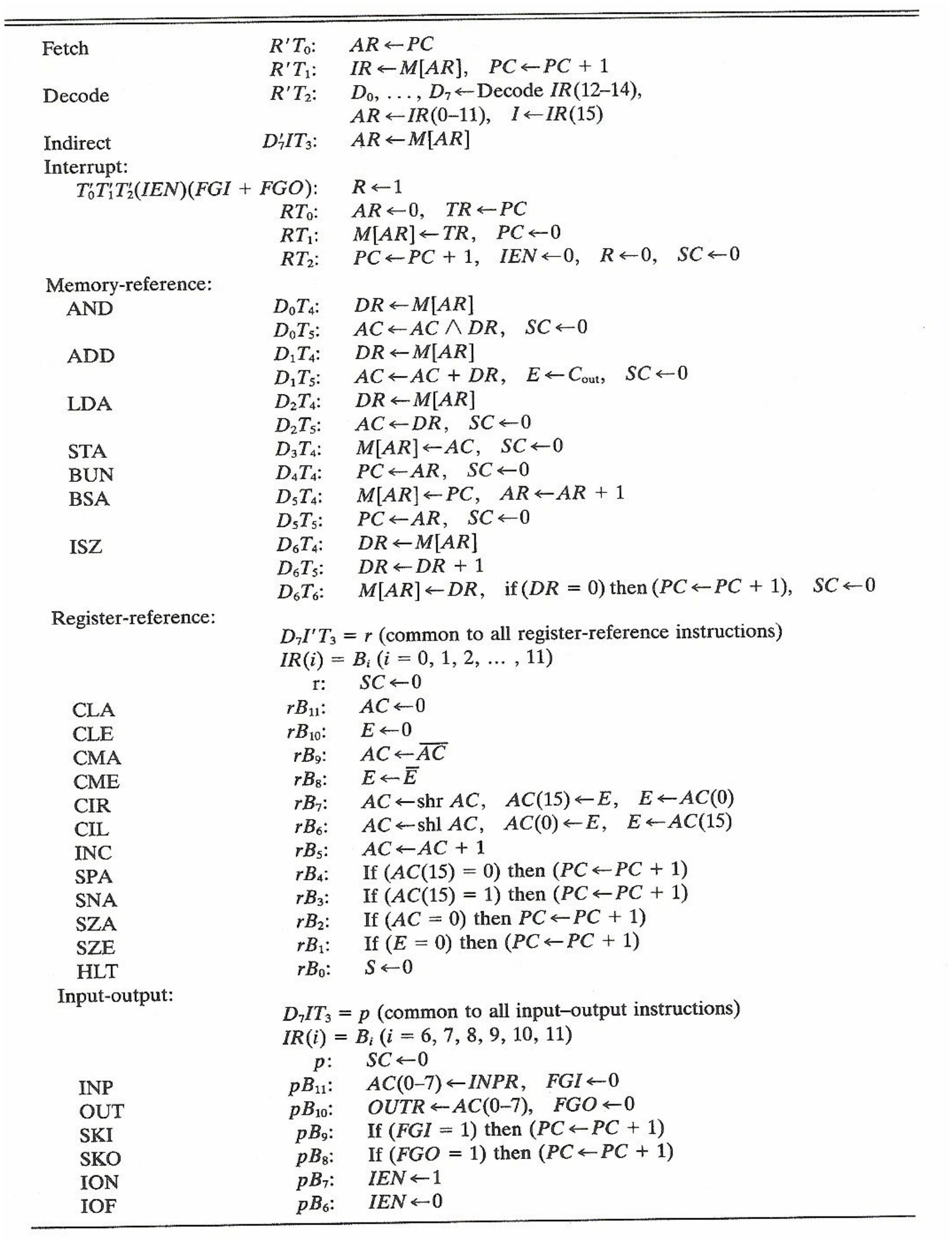
A complete list of micro-operations and timings for the interrupt cycle and all the instructions of the Mano machine are developed as shown in Table 2.

Table 1: Mano Machine Instruction Set



	Hexadecimal code		
Symbol	*I* = 0	*I* = 1	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Table 2: Computer Micro-Operations and Controls for the Mano Machine.



Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11), \quad I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:		
$T'_0T'_1T'_2(IEN)(FGI + FGO)$:		$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, \quad TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, \quad PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1, \quad IEN \leftarrow 0, \quad R \leftarrow 0, \quad SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, \quad SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, \quad E \leftarrow C_{\text{out}}, \quad SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, \quad SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC, \quad SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, \quad SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, \quad AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, \quad SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
Register-reference:		
	$D_7I'T_3 = r$ (common to all register-reference instructions)	
	$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$	
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E, \quad E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E, \quad E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Input-output:		
	$D_7IT_3 = p$ (common to all input–output instructions)	
	$IR(i) = B_i$ $(i = 6, 7, 8, 9, 10, 11)$	
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR, \quad FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7), \quad FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

In Table 2, R is the interrupt flip flop, T_i is the output from the sequence counter at different cycles. D_i is the output of the decoder which decodes the instruction function. IR(i) and B_i refer to the *i*th bit in the instruction register IR. Table 2 summarize how long each instruction takes, and what micro-operations are to be done in each clock cycle. It is not difficult to derive the control signals for the mano machine. Notice the following example:

To derive the load control signal of Data Register DR, we scan the whole table and identify all the places that have DR← (indicating DR is the recipient of a load operation). We find that DR will load data when $D_0 \cdot T_4$ is true, or $D_1 \cdot T_4$ is true, or $D_2 \cdot T_4$ is true, or $D_6 \cdot T_4$ is true. Thus, the control signal of load DR is $D_0 \cdot T_4 + D_1 \cdot T_4 + D_2 \cdot T_4 + D_6 \cdot T_4$. The control signal logic expressions can be derived by using the same method.

Project Assignment

The project requires that each student in the class implement this Mano Machine in Cedarlogic. To test the machine, a small assembly language program is used as shown in Table 3.

Another significant part of this project involves students writing an assembler for the Mano machine using their favorite high level programming language, usually C++ or Java. The assembler software accepts as input a text file of an assembly language program. It must then parse each line of text, and translate it into the corresponding machine code. The translated binary codes along with their memory addresses are output to a newly generated .txt file that will be loaded into the memory inside the Cedarlogic simulator. This requirement helps students come to a better understanding of the three levels of program representation since they will need to code with a high level language, as well as manipulate assembly language and binary code. The students also realize the importance of the instruction set architecture.

Once the binary code is ready, it is then loaded into the 4096×16 simulated memory chip in the simulated computer system and run (Cedarlogic provides a simulated memory chip which can be written by loading text format files). A successful computer will compute the right sum “023FH” in both the accumulator and the memory when the program halts the machine.

Table 3: Test Program for Basic Computer

Address	Contents		/This programs Adds 10 numbers
0000	**4100**		/jump to 100H where the test program stores
		ORG 100	
0100	**210B**	LDA ADS	/Load first address of operands
0101	**310C**	STA PTR	/Store in Pointer
0102	**210D**	LDA NBR	/Load minus 10
0103	**310E**	STA CTR	/store in counter
0104	**7800**	CLA	/Clear Accumulator
0105	**910C**	LOP, ADD PTR I	/Add an operand to AC Indirect
0106	**610C**	ISZ PTR	/Increment Pointer
0107	**610E**	ISZ CTR	/Increment Counter
0108	**4105**	BUN LOP	/Repeat Loop again
0109	**310F**	STA SUM	/Store Sum
010A	**7001**	HLT	/Halt
010B	**0150**	ADS, HEX 150	
010C	**0000**	PTR, HEX 0	
010D	**FFF6**	NBR, DEC -10	
010E	**0000**	CTR, HEX 0	
010F	**0000**	SUM, HEX 0	
		ORG 150	
0150	**0019**	DEC 25	/first # to add at address 150
0151	**0032**	DEC 50	
0152	**004B**	DEC 75	
0153	**0064**	DEC 100	
0154	**0019**	DEC 25	
0155	**0032**	DEC 50	
0156	**004B**	DEC 75	
0157	**0064**	DEC 100	
0158	**0019**	DEC 25	
0159	**0032**	DEC 50	/10th # to add at address 159
		END	/end of program The sum should be $575_{10} = 23F_{16}$

Cedarlogic Simulator

This project could not be undertaken without the Cedarlogic simulator. Cedarlogic is a windows-based digital logic simulator. It was the result of a senior design project of the computer science and engineering department of Cedarville University [5]. Cedarlogic functions in a similar way to the Diglog simulator [6], but it is Windows-based and provides a much better graphical user interface. Figure 4 shows a screen shot of a simple circuit within Cedarlogic:

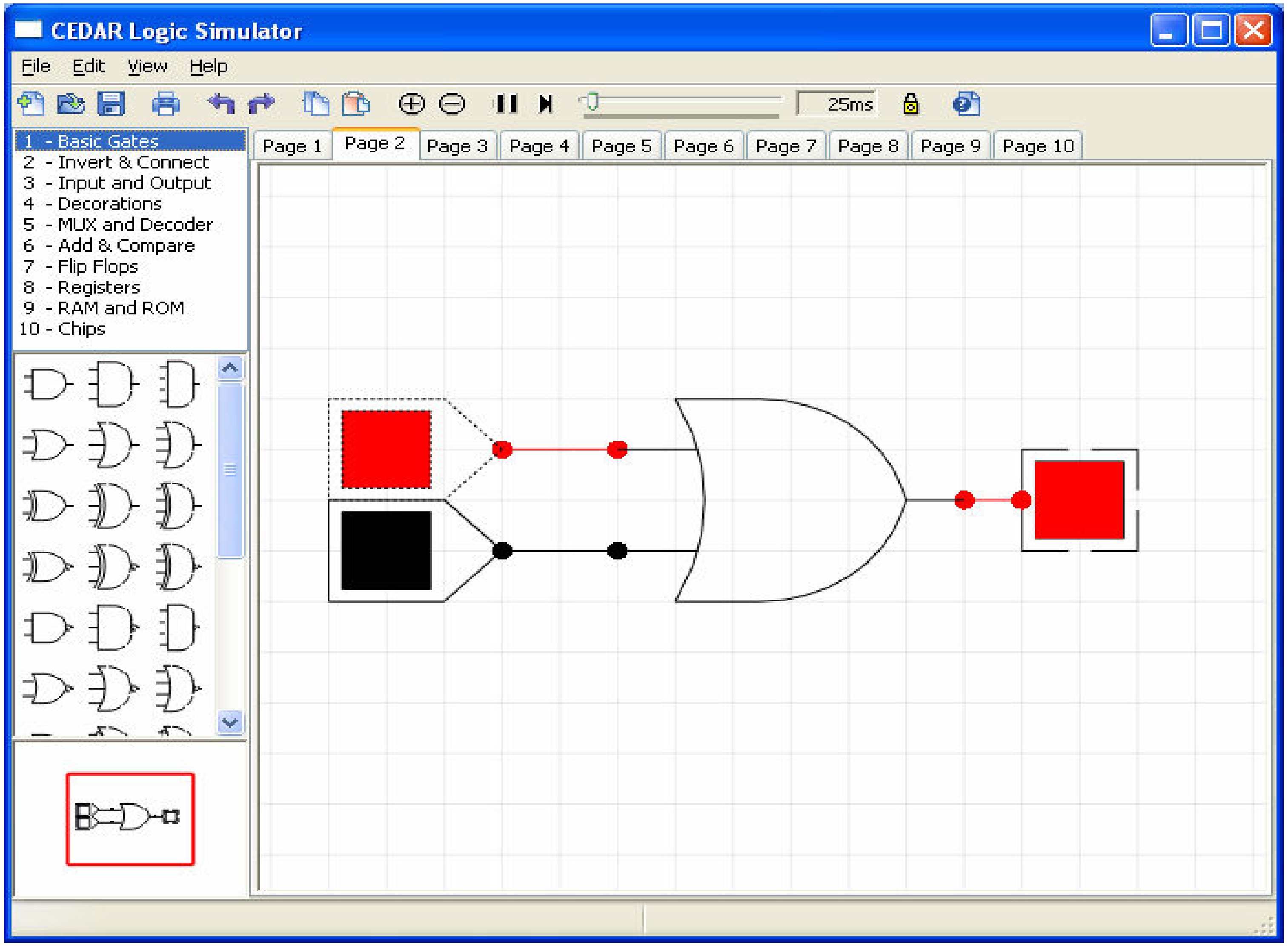


Figure 4. A Screen Shot of the Cedarlogic Simulator

Note the left hand menu that provides all the basic digital gates, clock, input dip switches, output LED's, keypad, and memory chips. In addition, the student can actually see the signals switching between high and low during this real-time simulation. Red indicates high, black indicates low, and green indicates high impedance. Besides an adjustable clock, registers and memory chips are also provided in Cedarlogic. They are shown in Figure 5.

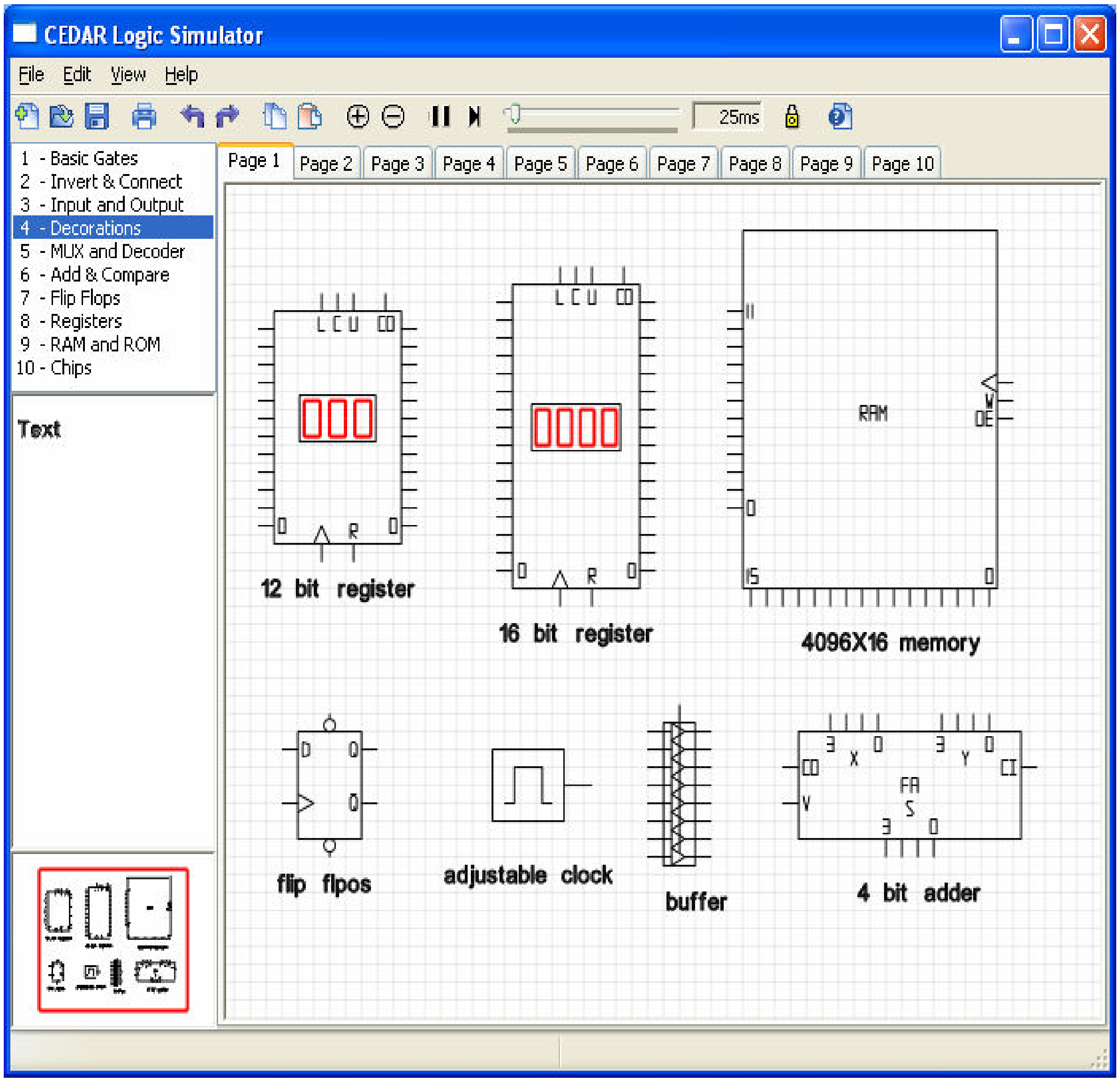


Figure 5. Digital Components Provided by Cedarlogic to Carry Out the Project.

Registers are available in 4, 8, 12, and16 bits wide. They can perform the functions of count (up or down), parallel load reset and hold. A memory chip can be loaded in the form of a txt file. If the student double clicks on a memory chip, a window will pop up requesting the input of a txt file. It has both write and read controls along with the synchronizing clock. Figure 5 shows some other basic digital components that are necessary to carry out the project, such as buffers, adders, and flip flops.

Project Implementation

Data path

The first step of this project is to require the student to complete the data path architecture in CedarLogic found in Figure 2, except arithmetic and logic operation units. A memory chip, registers and buffers are used. Figure 6 is one of the implementations of the data path with an accumulator.

From Figure 6, it can be seen that all the required registers are connected according to Figure 2. All the control signals for memory, buffers and registers are named properly. When the machine is reset, it will be noted that all the registers are cleared to zero.

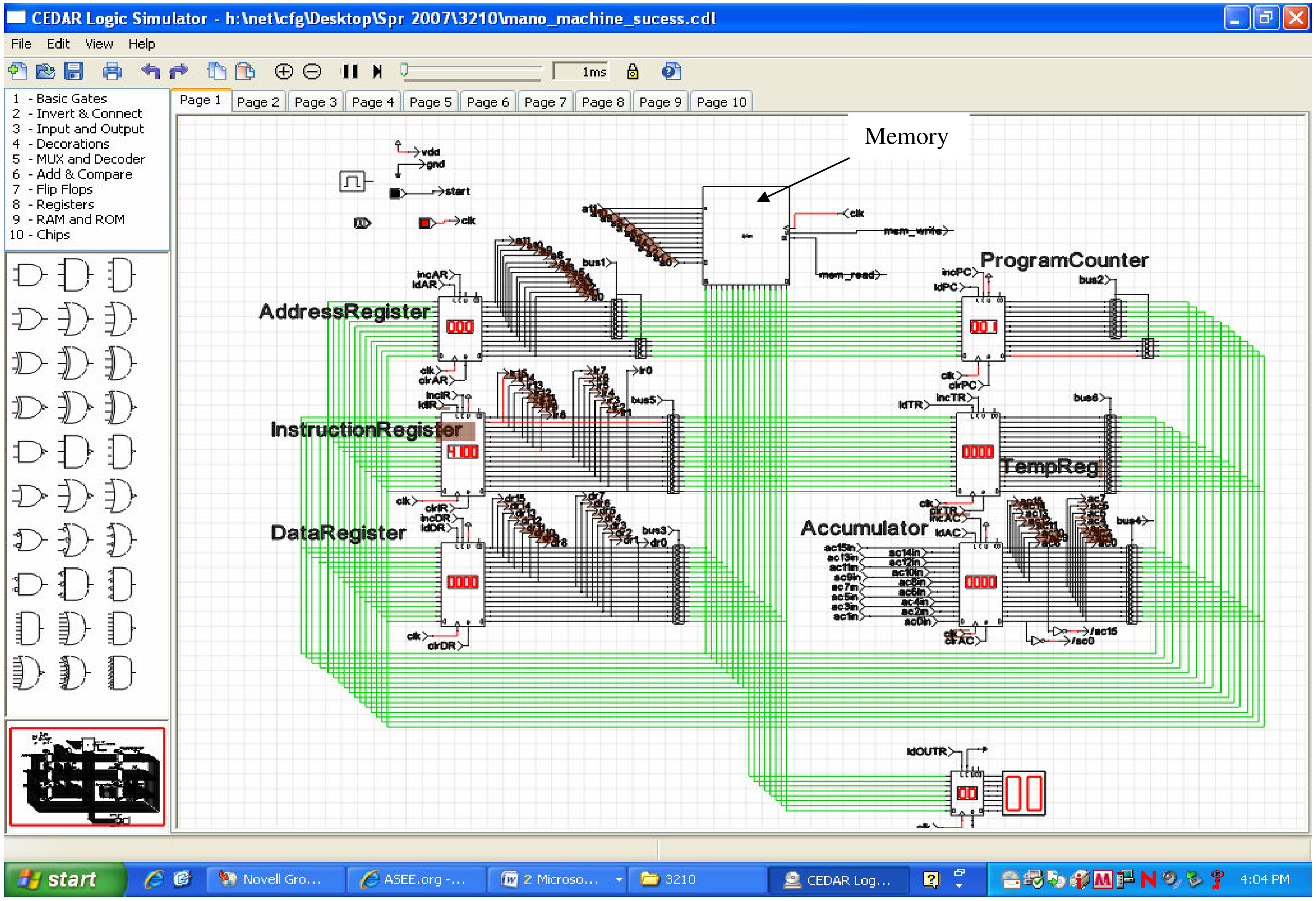


Figure 6. Mano Machine Data Path Implementation.

Arithmetic and Logic Unit

In order to keep the circuit easy to see and manage the arithmetic and logic operation units are implemented on a different page. Cedarlogic supports up to 10 pages. As long as they are named correctly, signals from different pages can interact with each other just like a single page simulation. The ALU can be implemented using a bit slice technique, which means a single bit of ALU is implemented and then 15 more copies can be made to form a 16 bit wide ALU with slight modifications. Figure 7 shows one of the implementations. On this page, the majority of the hardware is composed of 16 pieces that are almost identical. Figure 8 shows a closeup view of one bit of the ALU. In this example, a four-bit built-in adder in the Cedarlogic simulator is used as shown in Figure 9.

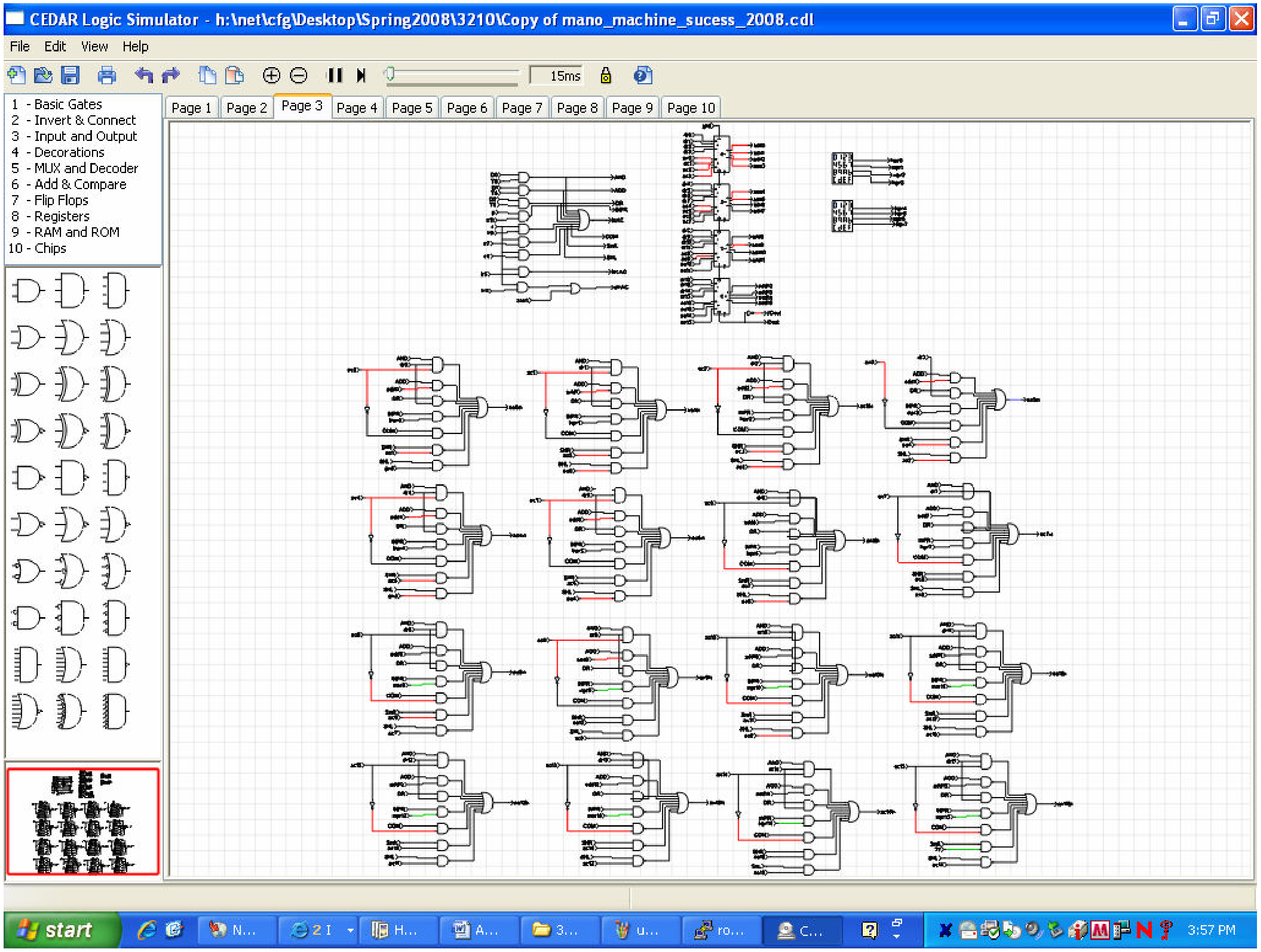


Figure 7. Overview of 16-bit ALU

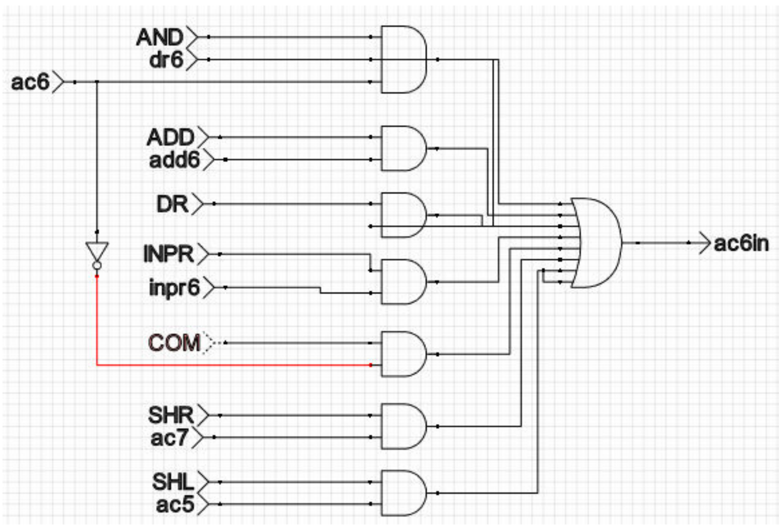


Figure 8. A Close-Up of a Single Bit of the ALU.

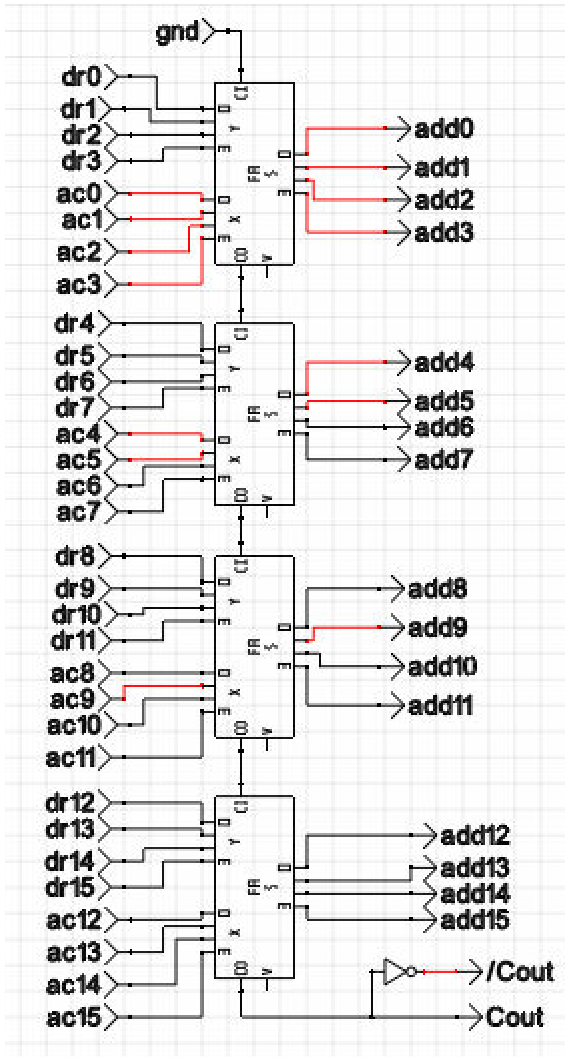


Figure 9. Four 4-bit Adders Used to Create the 16 Bits Addition Function Unit

As explained earlier, control signals can be derived from Table 2. Before implementing the control unit in Cedarlogic, students are asked to scan Table 2 to derive the Boolean expressions of all the control and timing signals on the registers, memory, bus buffers, instruction decoder, ALU, sequence counter, flip flops, etc. Once all the boolean expressions are ready, it is easy to convert them into a digital logic circuit. For instance, scan Table 2 about AR register, it can be summarize that: AR will be loaded only when the following conditions is true:

1. at T0 and not in the interrupt cycle (/R is true)
2. at T2 and /R is true
3. at T3 and indirect addressing mode (I=1) and it is memory reference instruction (D7=0)

The boolean expression confine ldAR control signal will be

ldAR=T0./R+T2./R+T3.I./D7

As a result, the control hardware of the ldAR is shown in Figure 10.

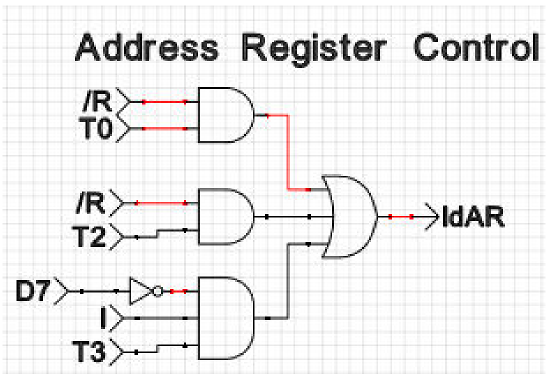


Figure 10. Control Signal Implementation Example – Load AR Control Signal.

Figure 11 shows the control unit with instruction decoder, sequence counter and all the control circuits related to memory, registers, buses, and resets. The entire control signal generated from the control unit will be propagated to the data path shown in Figure 6 to control the data flow and timing.

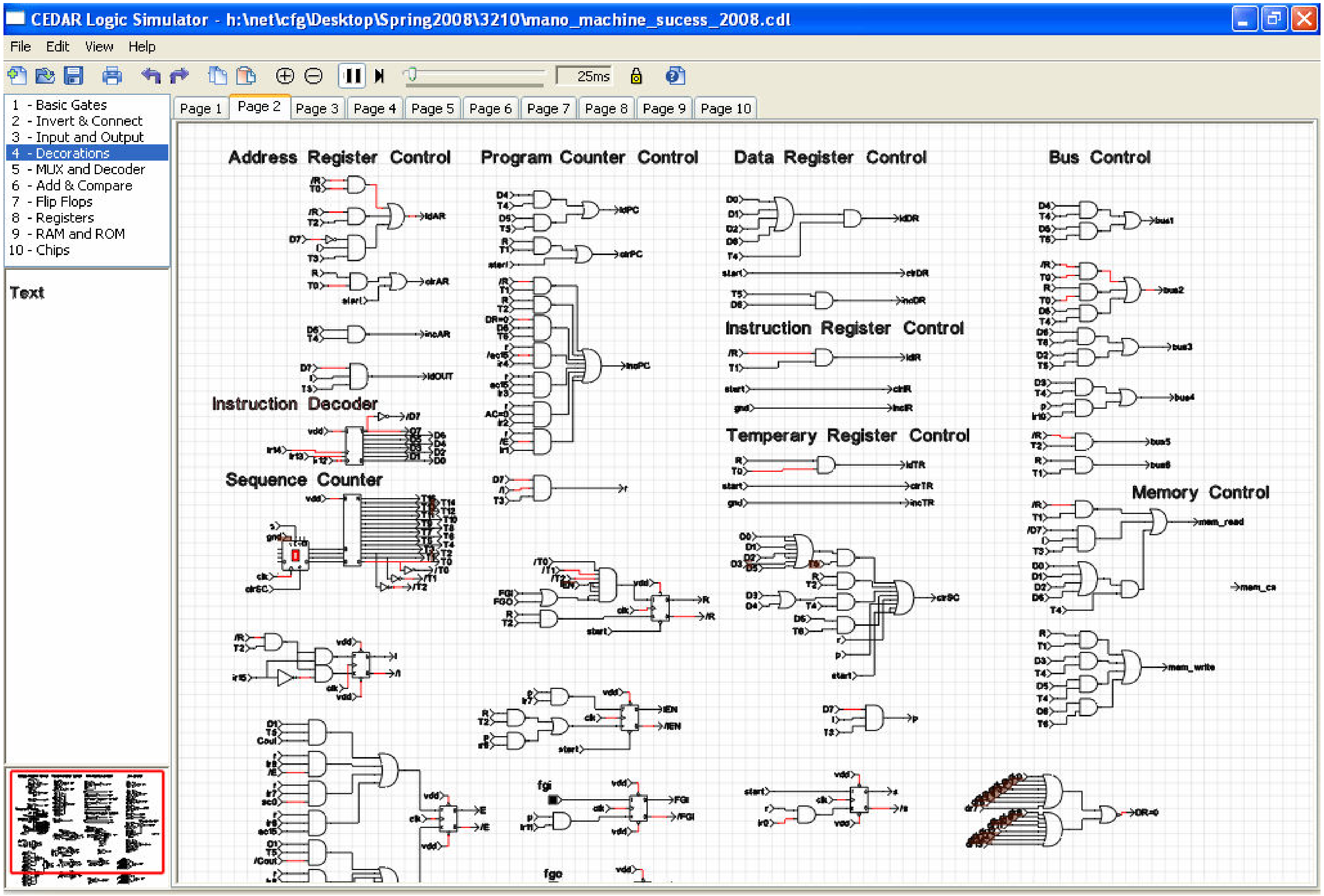


Figure 11. Control Unit Implementation

A Working Mano Machine Demonstration

A working Mano machine will clearly demonstrate that, at a specific machine cycle, certain control signals are active depending on which instruction is propagating in the microprocessor. It will also show the data transfer among registers and memory and switching on the buses.

Figure 12 shows one moment of the execution of the test program running on the Mano Machine. From this snap shot, it can be seen that the *read* control signal on the memory chip is active, the address buses are all black, the data bus shows red on bit 14 and bit 8. This indicates that memory at address 0000H is reading out; the content in that memory location is 4100H, which is a jump to 100H instruction. Meanwhile, the load control signal of the instruction register is also active, waiting for the clock to arrive. Once the rising edge of the clock arrives, this instruction will be latched into the instruction register as shown in Figure 12-2. The content of IR now has been modified to 4100H after the clock ticks one time.

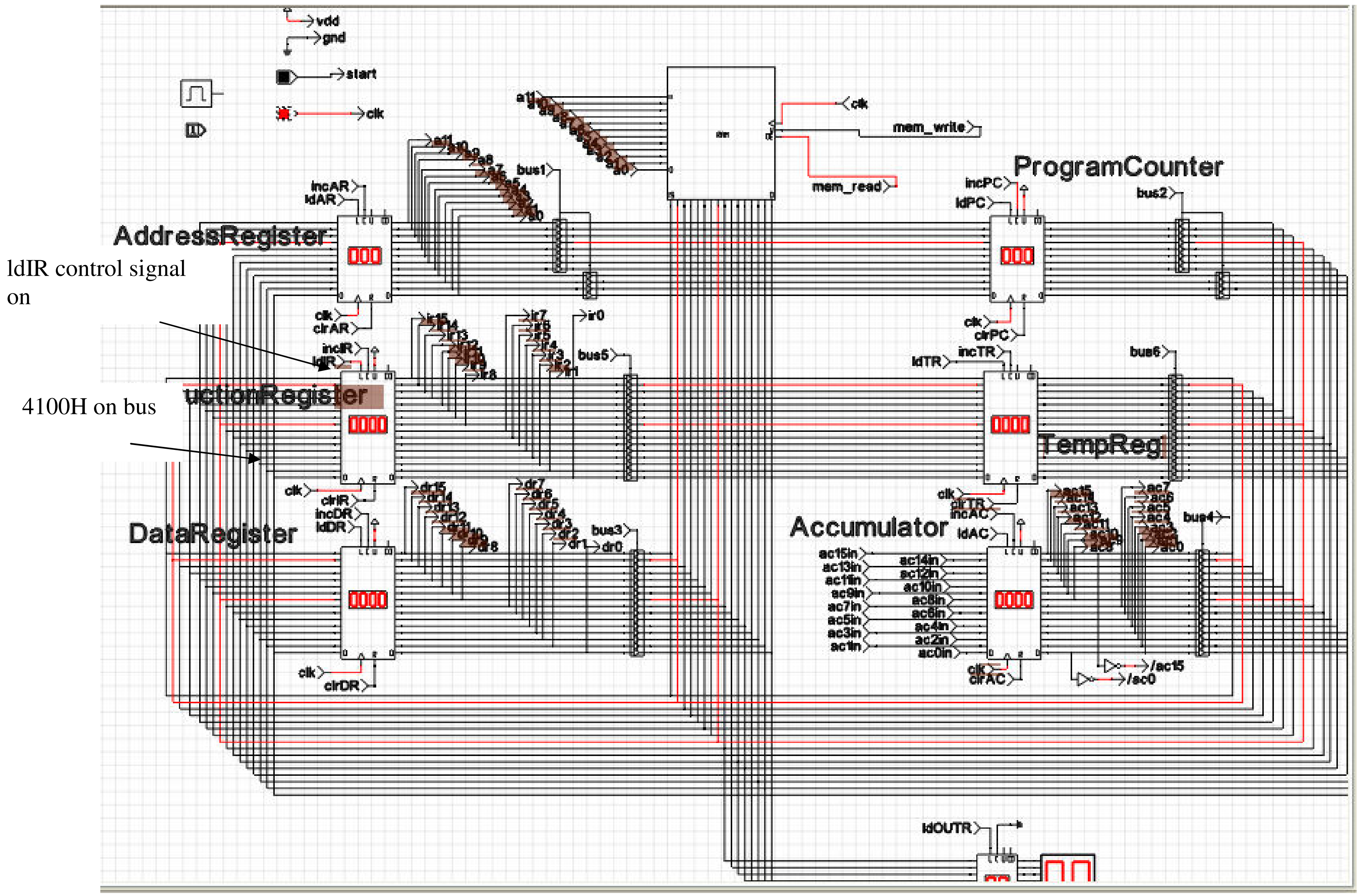


Figure 12-1. Demo 1 on a Working Mano Machine: Memory Location 0000 is being Reading Out

Instruction 4100H
latched in

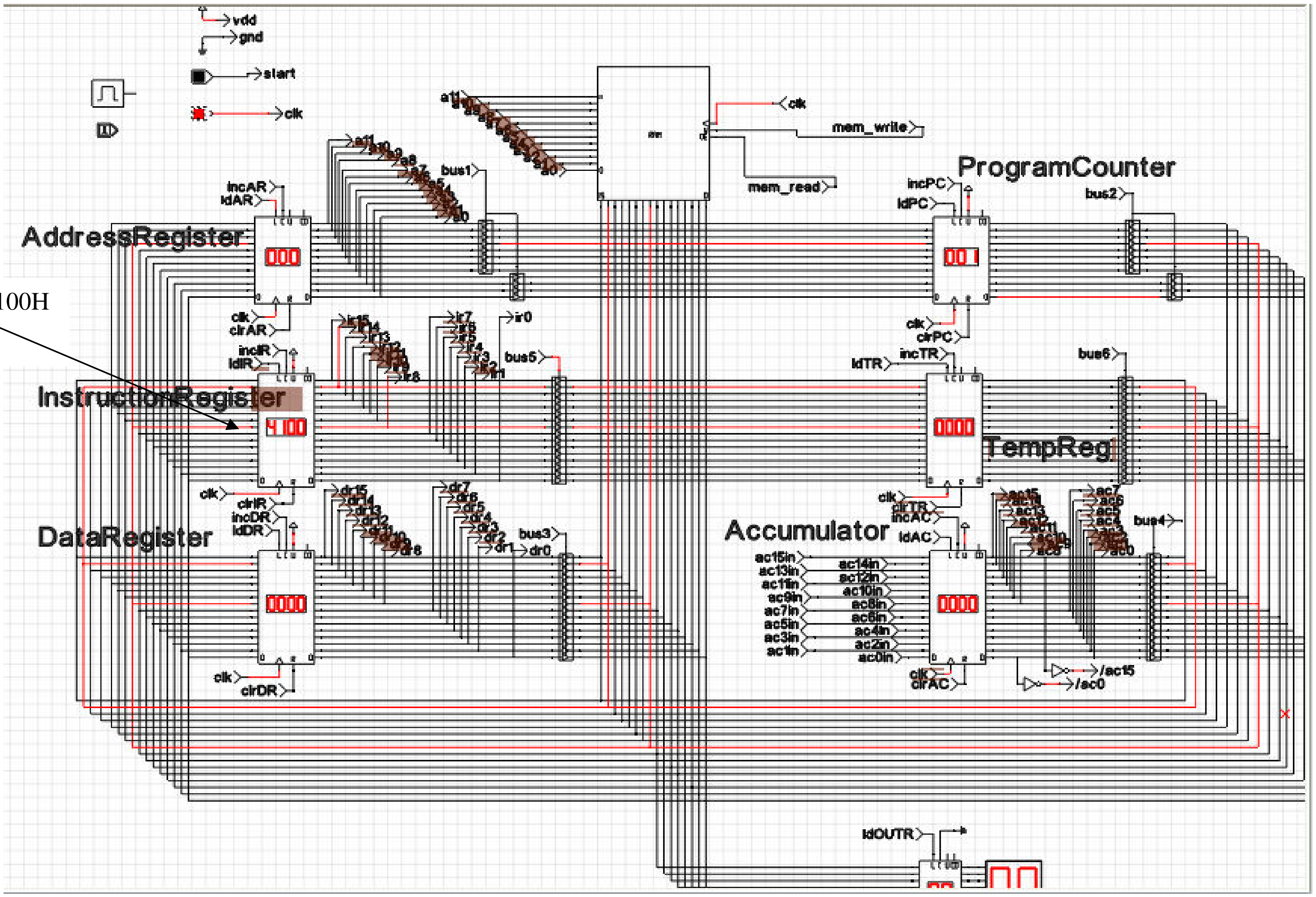


Figure 12 -2. Demo 2 of a Working Mano Machine: Instruction Register IR Latches the Instruction 4100H.

Figure 12-3 shows that at the moment of reading from memory, the sequence counter is at the T1 state of the instruction fetch stage. And the ldIR (load instruction register) control signal is high.

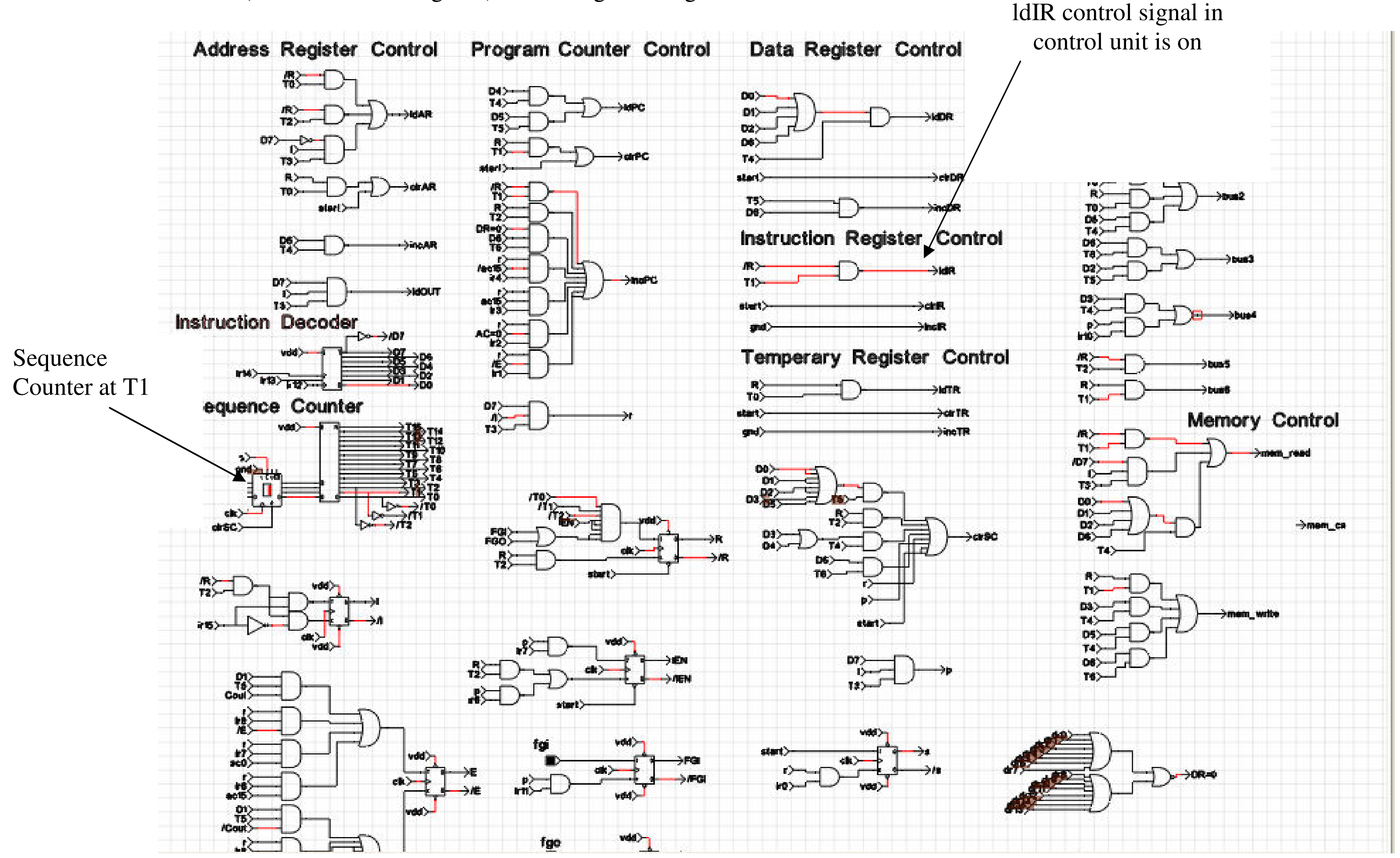


Figure 12-3 Demo 3 of a Working Mano Machine: at T1: ldIR Control Signal is Active.

Figure 12-4 shows that after the clock ticks, the T-state proceeds to T2 state and ldAR is inactive.

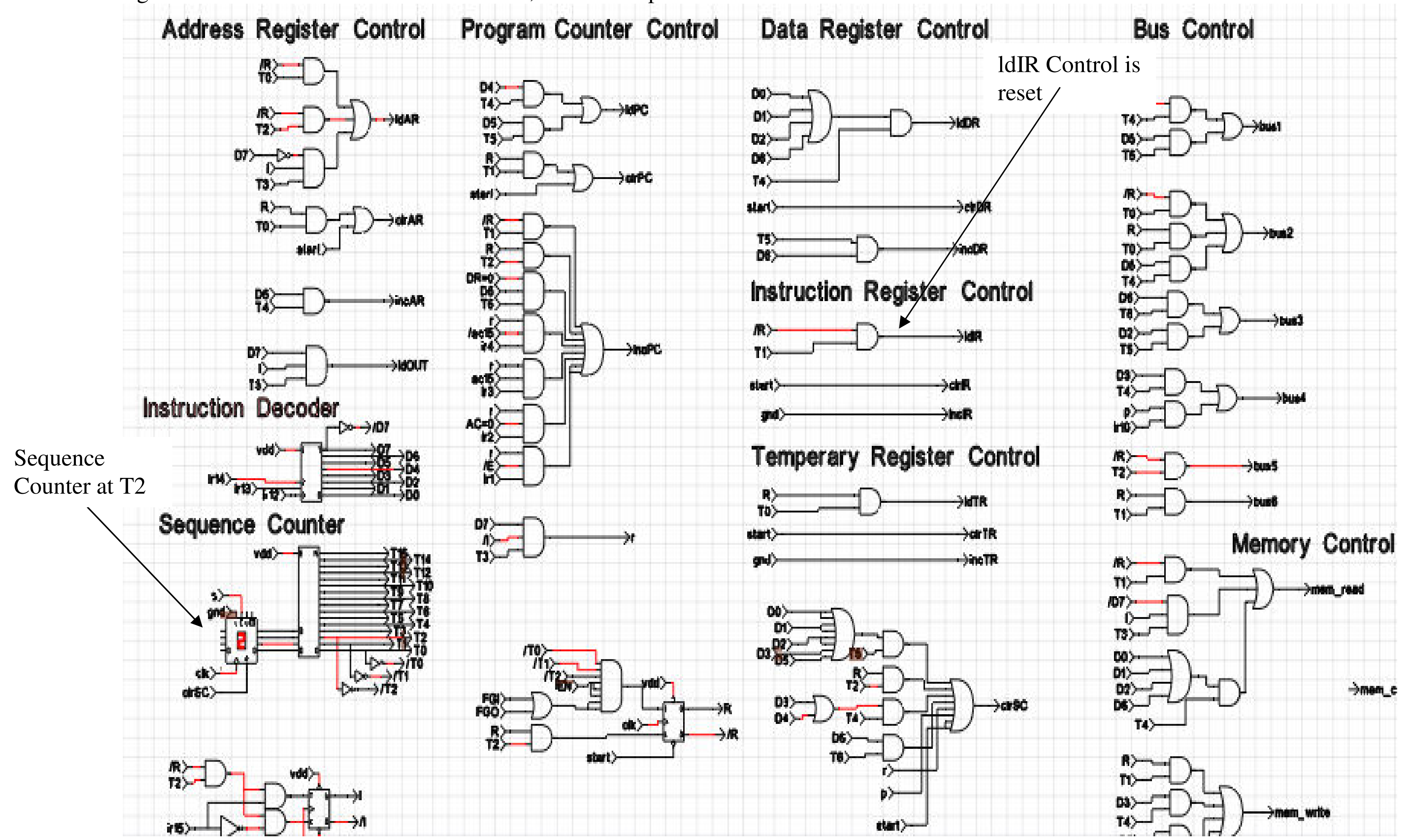


Figure 12-4. Demo of a Working Mano Machine: at T2: ldIR Control Signal is Reset.

A working Mano Machine implementation will allow the students to visualize every step of computer's operation of the test program running T state by T state. If the student makes an error in his hardware design it is relatively easy to catch and correct it since all signals are easily observed. For instance, since during state T1 the instruction fetch is to occur, the student would expect to see a high on the memory read and a high on the IR load, the sequence counter should be at 1. If one or more of those control signals does not function correctly, it will be easy to identify it. The students can single clock the machine if necessary.

Figures 13 and Figure 14 show the successful result of the test program

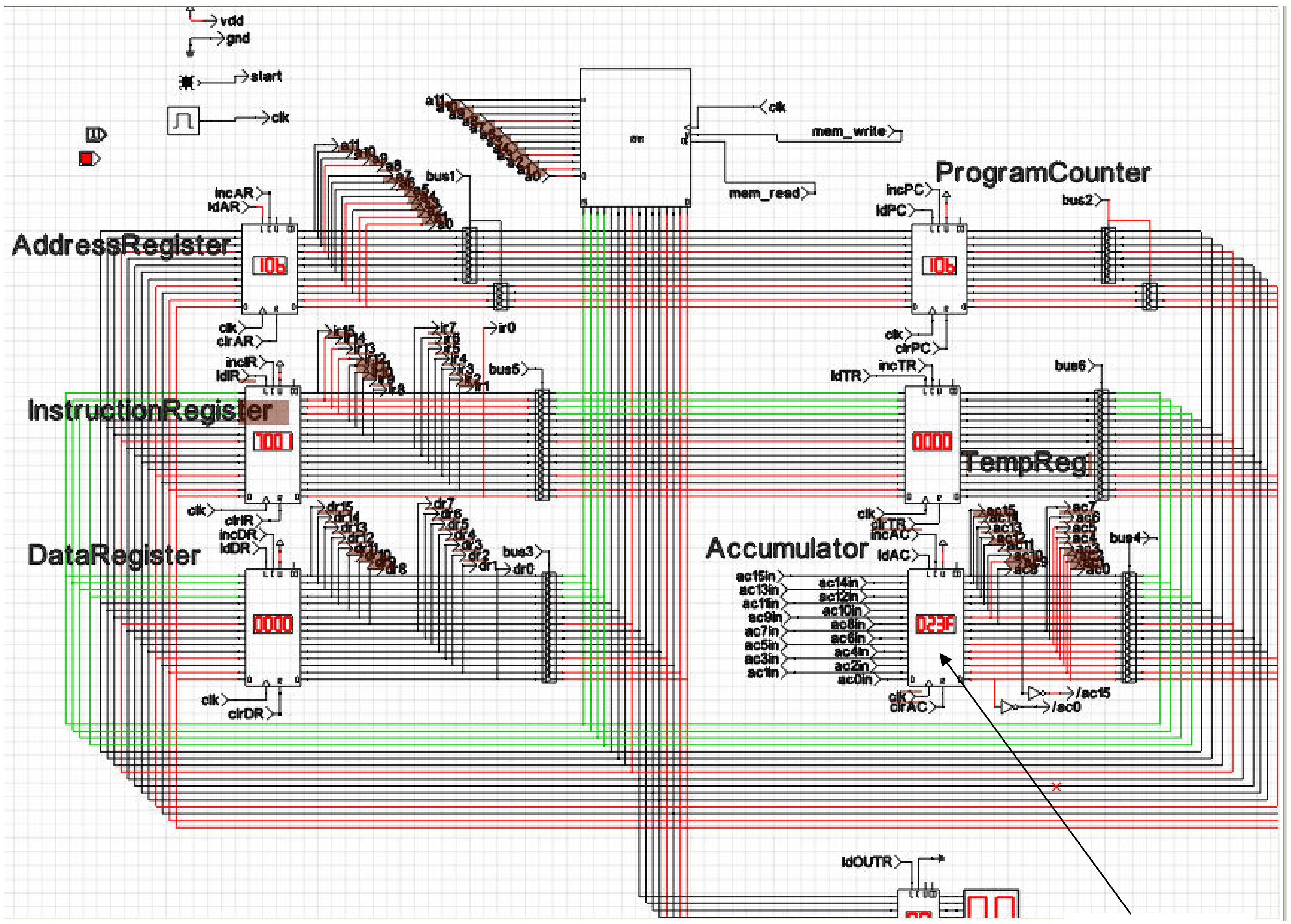
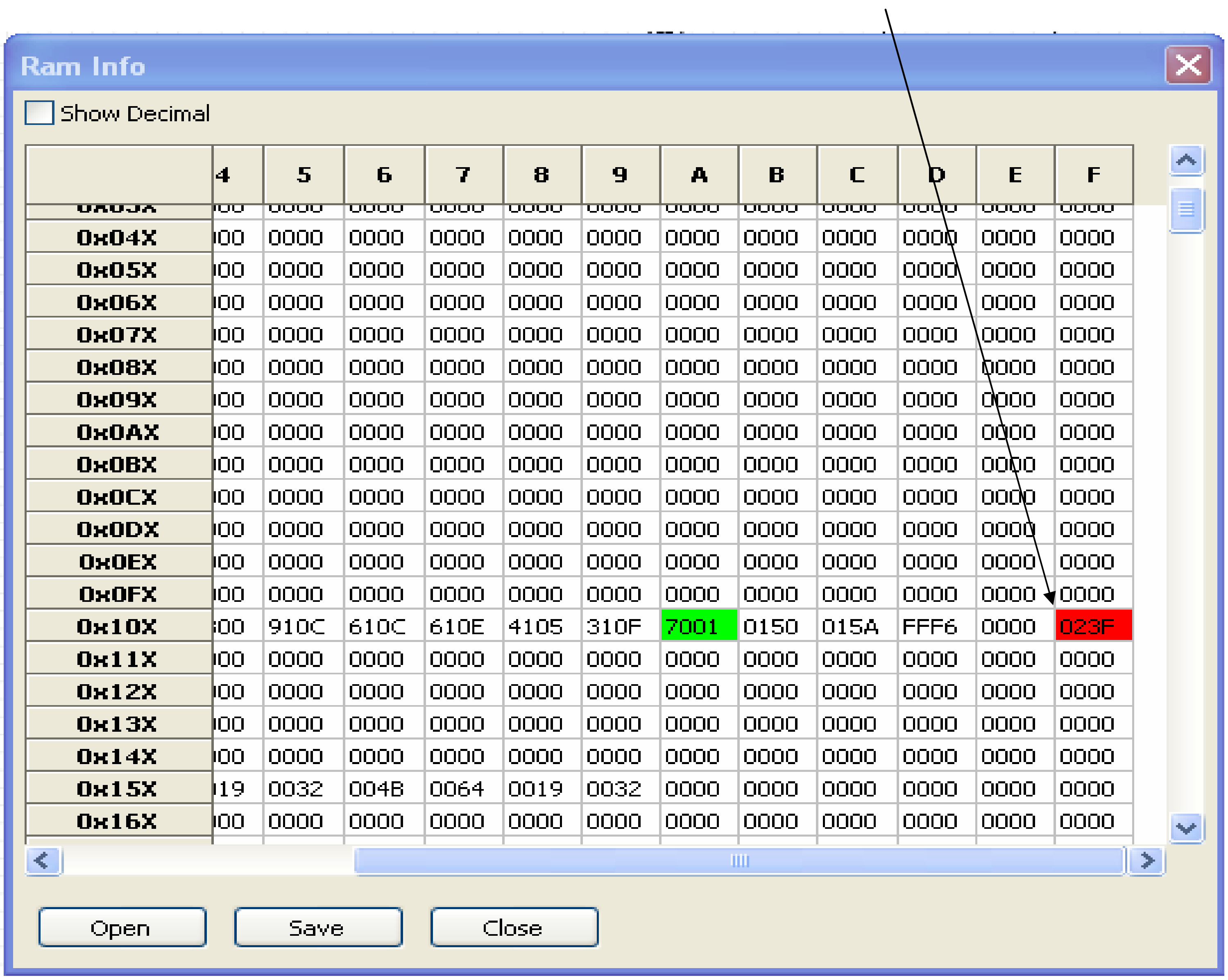


Figure 13. Correct Sum Shown in Accumulator

Correct Result



	4	5	6	7	8	9	A	B	C	D	E	F
0x04X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x05X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x06X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x07X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x08X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x09X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x0AX	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x0BX	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x0CX	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x0DX	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x0EX	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x0FX	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x10X	00	910C	610C	610E	4105	310F	7001	0150	015A	FFF6	0000	023F
0x11X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x12X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x13X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x14X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0x15X	19	0032	004B	0064	0019	0032	0000	0000	0000	0000	0000	0000
0x16X	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figure 14. Correct Sum Shown in Right Memory Location.

With an appropriate clock rate, students will see a "live" computer dancing under the control of the "program's" commands. They begin to really see through to the inside of how the computer signals are generated, how the data are transferred between registers and the memory and how each instruction propagates through the digital circuitry. When the students get their machines to work they are usually very excited and really seem to better understand how a CPU works.

Outcomes

In class evaluations many students have written very positive comments about how much they enjoyed this project and how helpful it was to their understanding of computer architectures.

The first midterm exam covers the concepts of computer basics. Prior to implementing this project in Cedarlogic, the average of the first midterm exam was 88%. After utilizing Cedarlogic with this project in spring 2007, the average of the first test increased to 91%. Also the overall rating on the course evaluations is improved from the middle 40-50% to the higher 20% rating.

This project extends the textbook and gives the students the opportunities to enhance their knowledge by actually implementing and visualizing the mano machine they designed. This project helps the students confirm the content they learned in class and consolidate their knowledge.

Cedarlogic can be freely downloaded from the site: http://sourceforge.net/projects/Cedarlogic*.

References

1. Guoping Wang, "Design Practice in Computer Architecture Teaching." Computer in Education Journal. Vol XVII No.1

2. 2. Yong-Kyu Jung, "Work In Progress – A Rapid Design Methodology for FPGA Based Processor Platform Design Education." FIE 2005 Proceedings.

3. Cedarlogic at http://sourceforge.net/projects/Cedarlogic

4. Morris Mano, "Computer System Architecture" 3rd Ed. Prentice Hall.

5. Clint Kohl and Keith Shomper, " Cedarlogic – a new Graphical Digital Logic CAD tool to aid in the teaching of Digital Logic Design". 2007 ASEE International conference, Honolulu, Hawaii

6. UC Berkeley, CS Division 387 Soda Hall, Berkeley CA 94720 (http;//www.cs.Berkeley.edu/~lazzaro/chipmunk/)

Acknowledgements

Figures 1, 2 and 3, and Tables 1 and 2 are reproduced directly from Morris Mano, "Computer System Architecture" 3rd Ed. Prentice Hall.