AMD 2900 Processor Simulator

stud. Vlad Ion
University POLITEHNICA of Bucharest, Computer Science Department
313 Splaiul Independentei,
sect. 6, 77206, Bucharest
ROMANIA
vladion@home.ro

stud. Sorin Toma
University POLITEHNICA of Bucharest, Computer Science Department
313 Splaiul Independentei,
sect. 6, 77206, Bucharest
ROMANIA
sorintoma50@xnet.ro

Ph.D. Decebal Popescu
University POLITEHNICA of Bucharest, Computer Science Department
313 Splaiul Independentei,
sect. 6, 77206, Bucharest
ROMANIA
decebal@csit-sun.pub.ro

ABSTRACT

A simulation software for the AMD 2901/2909 processor was created in order to be used for didactic purposes. It consists of two major components, the program editor and the simulator, and it is designed for an easy understanding of the processor internal workings. The application is built on the .NET technology using C# and it is used in our university as an educational resource.

Keywords

Processor simulator, AMD 2901, AMD 2909, educational software, .NET technology.

1. INTRODUCTION

The idea of developing a processor simulator was based on some observations we made at one of the most important courses studied at the Computer Science Faculty, which is the Digital Computers course. One of the main topics is the study of the AMD 2901/2909 microprocessor. For this topic students have to understand the general architecture of the microprocessor, with its resources and the way they are interconnected as well as how it can be programmed, and the specifications of its assembly language. The main activity at the seminaries is to solve certain tasks using/creating micro programs for this processor.

Because all the activities were focused on theoretical aspects, sometimes very hard to understand without a practical support, we felt the need for a tool that actually shows how a processor works at the register transfer level (RTL), executing a custom micro program. This way many of the practical aspects of programming a microprocessor can be evaluated, which leads to an easier and better understanding of the course subject. This was the project's starting point. Two types of objectives were set – features for the application and technology requirements. The application should have a user-friendly, easy-to-use micro program editor in order to cope with the stiffness of the assembly language syntactic rules, and describe the functions behind the mnemonics. There should be also a simulator to run the program, and show the resources – Arithmetic Logic Unit (ALU), RAM, Shift Registers, source selector, control unit, stack – and their status, several inputs and outputs, values of specific flags at every step of the execution.

For the technology the main requirements were about portability – the application was distributed to almost all of the Computer Science Faculty students, which attended the Digital Computers course, and should have run on everyone's operating system – and the existence of powerful tools for fast application development as well as a powerful and fast programming language – the deadline for the project was short. Under these requirements we chose the .Net technology and the C# language, and Microsoft Visual Studio .Net as a development environment. After testing and inherent bug fixing the application has reached its goal being used by almost all students studying Digital Computers, testing their programs, and offering a better understanding of the processor's inner functions.

.NET Technologies'2004 *workshop proceedings,*
ISBN 80-903100-4-4

2. GENERAL DESCRIPTION

The AMD# application (can be found at [Web00]) consists of two main components: the editor and the simulator. The first step of the simulation is the creation of a micro program (several lines symbolizing the hardware commands accepted by the microprocessor) in the editor and the second step is the actual simulation of the given program in the simulator window (the user can see the program flow and the evolution of flags, resources and results in the microprocessor).

The AMD 2900 family of integrated units consists of bit-slice processors, that is processors that work with data units that are not a multiple of 8 (there are units that process 4, 8, 12,16 bits of data, and so on). A complete unit is made of two basic units, the arithmetic and logic unit (ALU) AMD2901 and the control unit (CU) AMD2909, and some auxiliary circuits. For a detailed description of the structure and functionality see [Petr01], [Web02], [Web03].

The control unit's main component is the micro sequencer that decides the address of the next microinstruction based on the current one and the status flags (it can do conditional or direct jumps, calls for a subroutine and returns from the subroutine based on an internal stack, etc.). The CU generates in the end a command that the ALU can understand. The ALU has an internal memory of 16 4-bit words, some registers and the unit that accomplishes the actual requested operation and generates the status flags (the zero flag indicating that the result is zero, the result overflow flag, the carry flag, etc).

All the data that is processed in the unit is a multiple of 4 bits because the basic ALU and CU units work with 4-bit data and addresses (there are 4 bits required to access the 16 locations in the ALU memory). The basic ALU's can be connected in a serial fashion in order to process larger data and the CU's can be connected in parallel to extend the memory address space.

The micro program format that can be seen in the editor closely reflects the internal structure of these units. Each micro instruction that makes up the program is a concatenation of bit groups, each used by a certain part of the unit (in the current implementation 4-bit and 8-bit programs can be simulated; in the 4-bit case the 32-bit instruction has a block of bits 27 to 24 representing the command for the micro sequencer, the 31 to 28 group representing the jump address, used only if the other group requests a jump operation, another group selects the needed ALU operation and so on; the instruction bits have been labeled from 31 to 0). The unit can accept external 4-bit or 8-bit data and for reasons of simplicity and ease of understanding this data has been integrated in the instruction code as the last 4 or 8 bits.

This seemingly complex instruction form is necessary because the programming is done at a very low machine level. The necessary bits are taken directly form the instruction and used by the unit's components with almost no conversions. This makes the unit work very fast and be completely automated. The negative aspect of this hardware programming is that the unit cannot generate dynamic code and data,

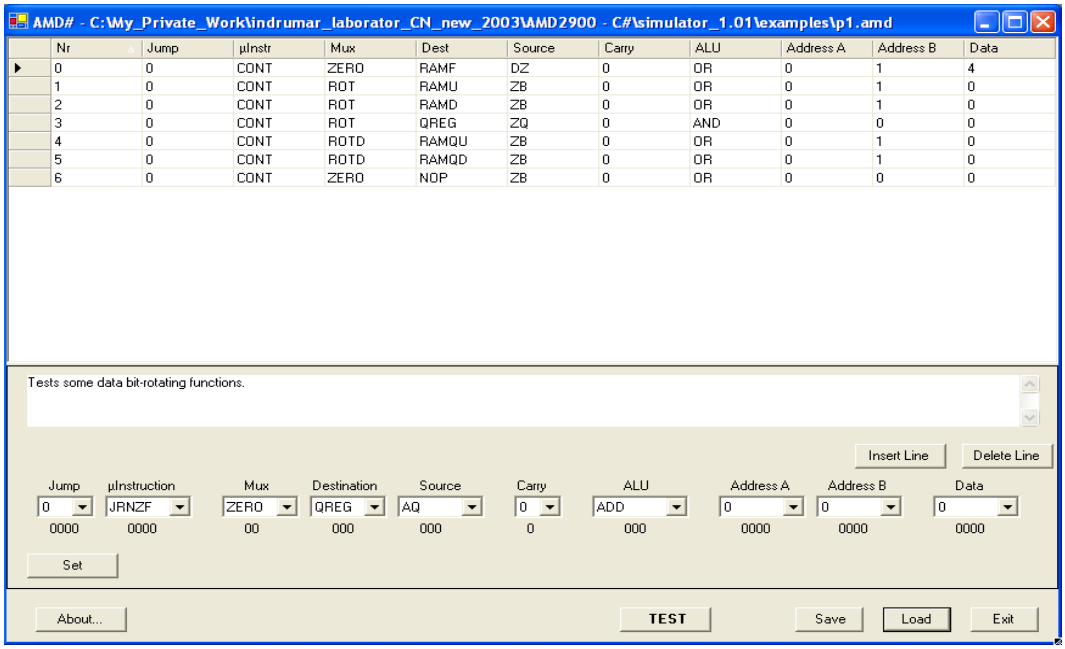


Figure 1. The Editor Window

and all the external data and jump addresses are set in hardware from the beginning.

3. APPLICATION INTERFACE

As it was previously mentioned, there are two program components which directly reflect the hardware structure of the AMD unit.

At program start the user can choose the number of bits the ALU and CU work with, 4 or 8 in the current implementation.

The editor window (as can be seen in Figure 1) is made up of a table and a comments field in the upper part, that show the structure of the currently entered program and a brief description of its purpose. In the lower part, there are a series of combo boxes and buttons that allow the creation of a micro instruction. Each block that makes up the instruction is selected by one combo box and its purpose is explained by a tool tip; this helps the students write the programs fasters by removing most of the need to check the detailed tables explaining each function, from the course. There are also buttons for inserting new lines and deleting previous lines from the program. The students can also save their programs in the editor or load previously written demos.

After finishing the program one can push the "Test" button to get to the simulator window (the simulator window style can be changed from the radio buttons next to the "Test" button). This window, as can be seen in Figure 2, has three main parts. On the left there is a table presenting the currently simulated program and the binary form of the current instruction. In the middle there is the representation of the ALU internal structure, showing the internal memory and registers content, the input instruction data blocks and the result and status flags. Lastly, the CU on the right shows the current instruction and jump address, the micro sequencer structure (stack content, stack top pointer and flags) and the computed next instruction index. The content changing of all these resources during execution can be observed by stepping forward and backward in the program.

4. APPLICATION STRUCTURE

The modularity of the C# object-oriented language was used to separate the interface from the functionality. Thus two interface modules were created, one for the editor and one for the simulator (the "**`sim.cs`**" and "**`edit.cs`**" files), using windows forms and the visual designer. This greatly sped up the application building. There were two modules needed for the ALU and CU and some abstract classes that define the general structure of the AMD instruction, the ALU RAM model etc.

The classes implementing the basic units follow the hardware model. The CU class contains an instantiation of the ALU and one of the sequencer. It passes the needed blocks of bits from the instruction to the sequencer and ALU just like the real CU does. Then the simulator class only instantiates the CU and receives a vector of instruction data that it sends to the CU, line by line, during the process of stepping through the program. The simulator and the instruction data vector are created by the editor class.

Because of this modularity and close imitation of the real unit the program can be easily updated and extended.

The complete source files for the project can be found at [Web00].

5. A MICRO PROGRAM EXAMPLE

There are several demo micro programs built for

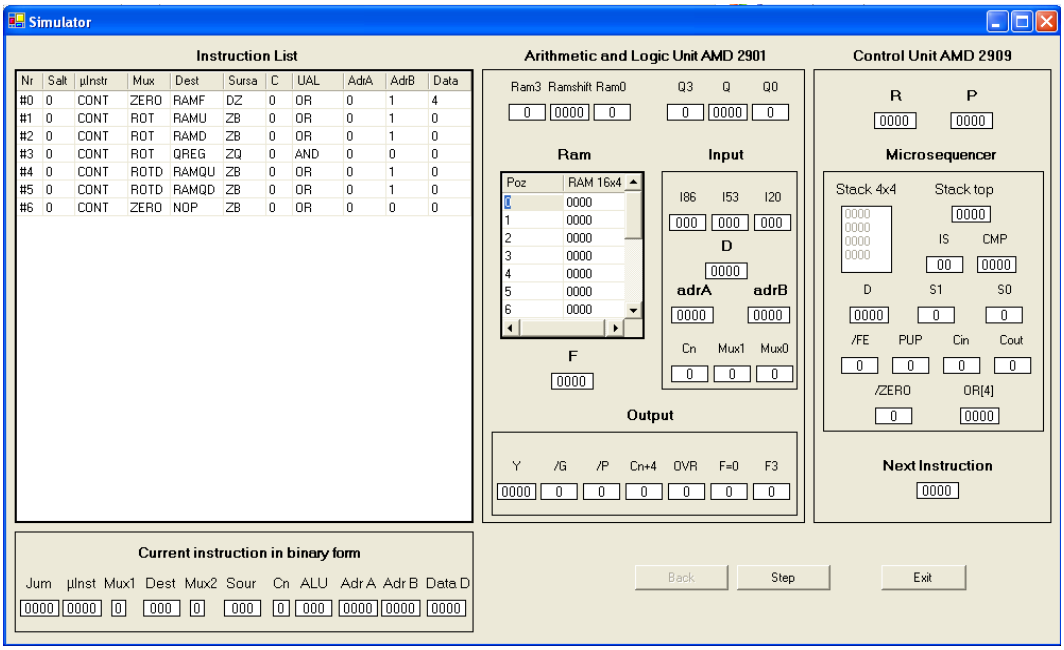
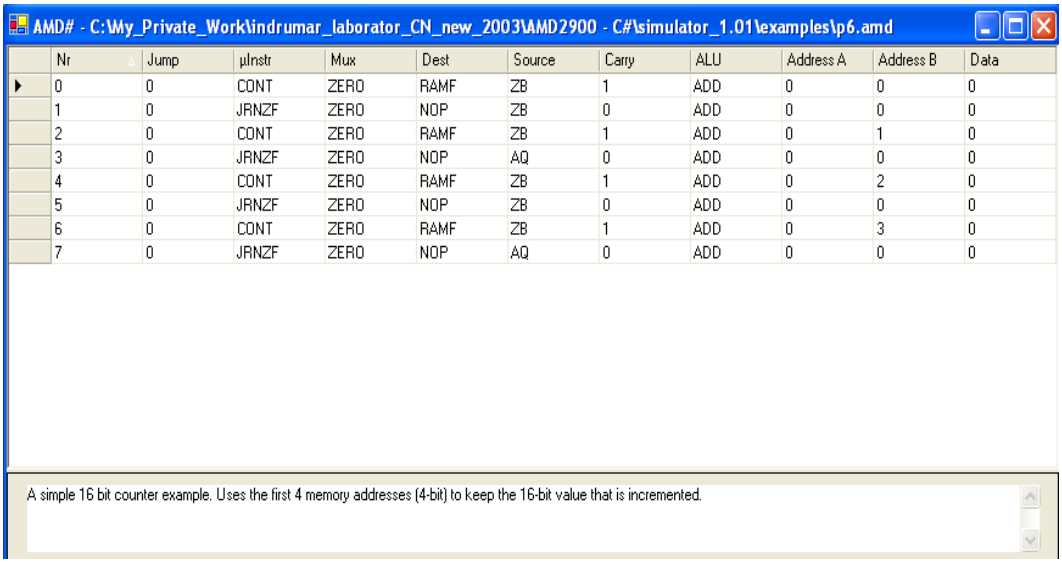


Figure 2. The Simulator Window.

laboratory activities. One can be seen in Figure 3. It is a 16-bit counter that uses 4 4-bit data locations in the ALU RAM in order to store the 16-bit counted value.

version of the simulator allows the user to select between 4 and 8 bit data – that means how many 4-bit processors will be used. In the future more data sizes will be allowed. This feature will require



Nr	Jump	µInstr	Mux	Dest	Source	Carry	ALU	Address A	Address B	Data
0	0	CONT	ZERO	RAMF	ZB	1	ADD	0	0	0
1	0	JRNZF	ZERO	NOP	ZB	0	ADD	0	0	0
2	0	CONT	ZERO	RAMF	ZB	1	ADD	0	1	0
3	0	JRNZF	ZERO	NOP	AQ	0	ADD	0	0	0
4	0	CONT	ZERO	RAMF	ZB	1	ADD	0	2	0
5	0	JRNZF	ZERO	NOP	ZB	0	ADD	0	0	0
6	0	CONT	ZERO	RAMF	ZB	1	ADD	0	3	0
7	0	JRNZF	ZERO	NOP	AQ	0	ADD	0	0	0

Figure 3. An example 16-bit counter program

In real-life application, like audio/video decoding, cryptography and embedded applications in general, these units, or updated and extended versions of them, are widely used. Some examples of implementations using the AMD 2900 family can be seen at [Web01].

6. FEATURES FOR THE NEXT VERSIONS

The first version of the AMD Simulator has accomplished all the objectives and goals which were set for it. Although, after seeing and using the application, new opinions and ideas for improvement arose. Many of the issues subject to change came to our attention after an active feedback from users, which were actually our colleagues and teachers, so the communication was established very easy.

Because the program is executed sequentially it can be represented as an algorithmic diagram, with instructions and program flow jumps. Based on this observation, the idea of developing a diagram designer came up. Thus the user can create the representation of the algorithm as a diagram and then convert it into a micro program for the AMD processor.

Being a 4-bit processor the AMD 2901/2909 is practically unusable for a real application and it serves only for educational aspects. Yet it has the possibility to connect parallel with other AMD processors and create a processor that handles bigger blocks of data containing 4-bit parts. The current serious change on the simulator interface to accommodate larger data, RAM and stack.

A new function will be added – the possibility to run the program at once at a selected step rate (the actual simulator runs a program only step by step). This option will require the possibility to insert breakpoints into the program. This running at once and the breakpoints will help the user bring the program to a specific instruction very fast, by setting a breakpoint on it and running the program.

7. .NET ADVANTAGES

The .Net Technology along with the C# language was clearly the best choice for this application. In the development process, the advantages reflected on both the final users and the developers. The main advantage provided for users was the portability, and for developers, the fast development tools and features. There were many components and features that helped by speeding up the development process. Among them, the most important were the modularity provided by the object-oriented language which is C#, the good classes documentation that allowed to choose the best class for a specific problem, the managed environment which probably saved very much of the debugging time.

One of the most important advantages that inspired a future feature of the simulator is the easy way in which this desktop application can be converted into a web application – an ASP.NET page – only by replacing the visual controls from “Windows.

Forms" with visual controls from Web Controls; the main code that controls the application is written in C# and it is compatible with ASP.NET.

8. REFERENCES

[Petr01] Petrescu, A.C., Digital Computers Course (Romanian version only) Chapter 10, http://www.csit-sun.pub.ro/?op=2&sop=1

[Web00] AMD# Application and source code, http://www.csit-sun.pub.ro/resources /cn/windows/simulator_AMD.NET_1.01.zip

[Web01] GEC 4000 series processors (examples of AMD 2900 implementation), http://www.cucumber.demon.co.uk/geccl/4000series/processors.html

[Web02] AMD 2900 Structure, http://www.cast-inc.com/cores/c2901/c2901-x.pdf and http://vision.gel.ulaval.ca/~maldagx/publications/Id445.pdf

[Web03] AMD 2901 bit slice - VHDL implementations, http://tech-www.informatik.uni-hamburg.de/vhdl

9. ADDITIONAL SCREENSHOTS

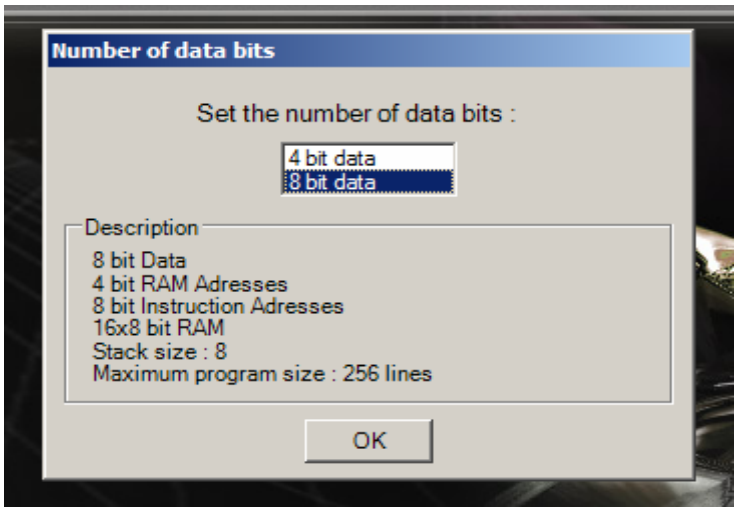


Figure 4. Program Start

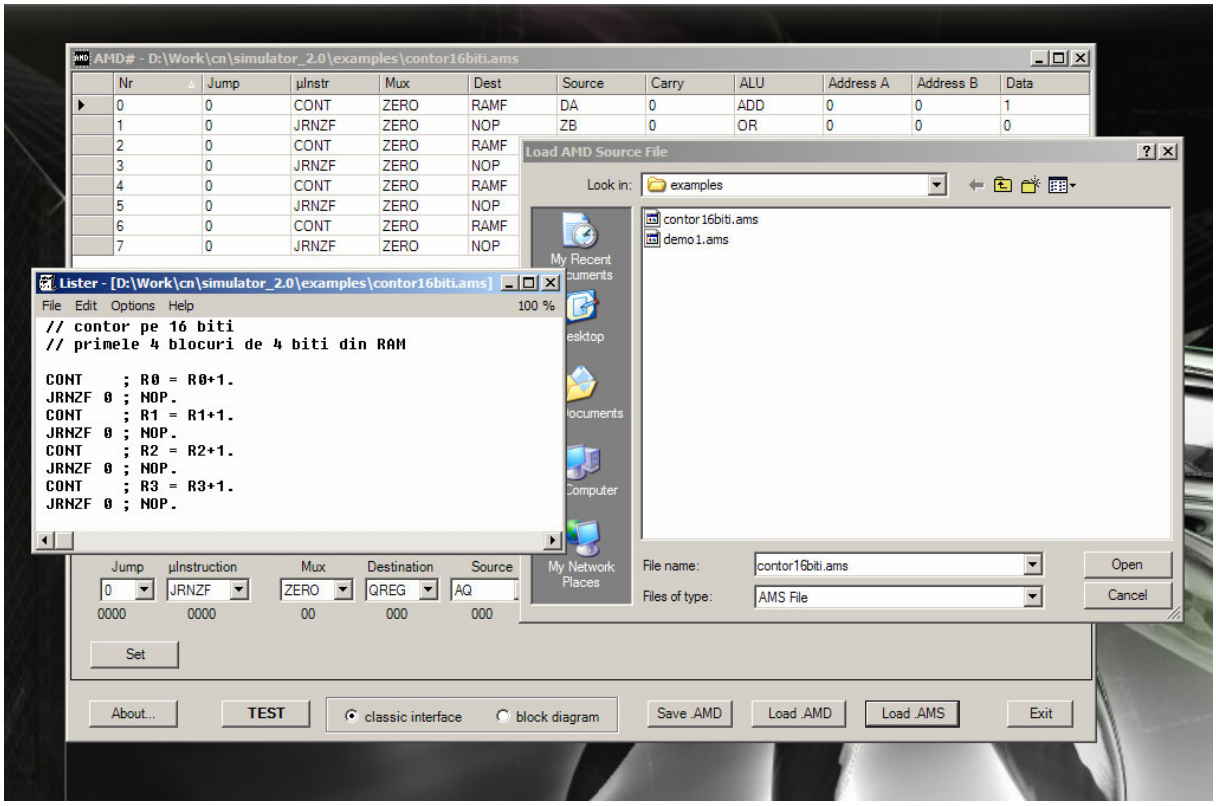


Figure 5. Loading an .AMS Assembly file in the new editor interface

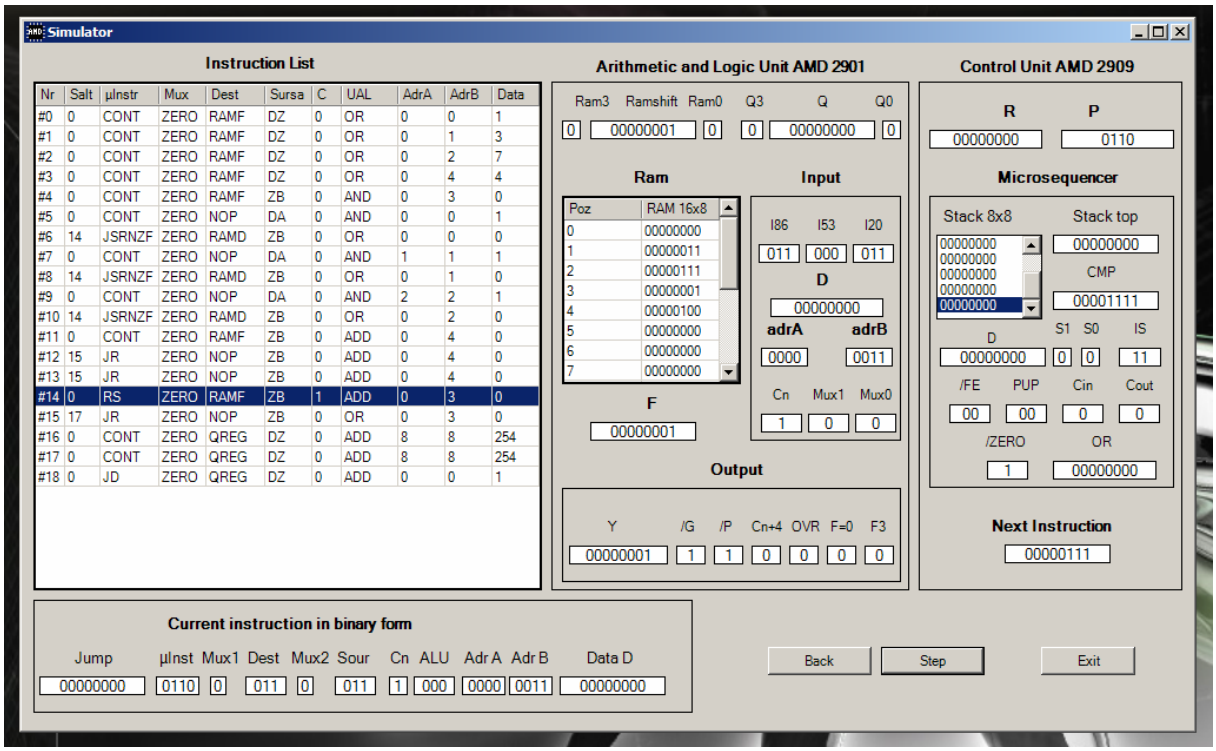


Figure 6. The new "classic interface" simulator window

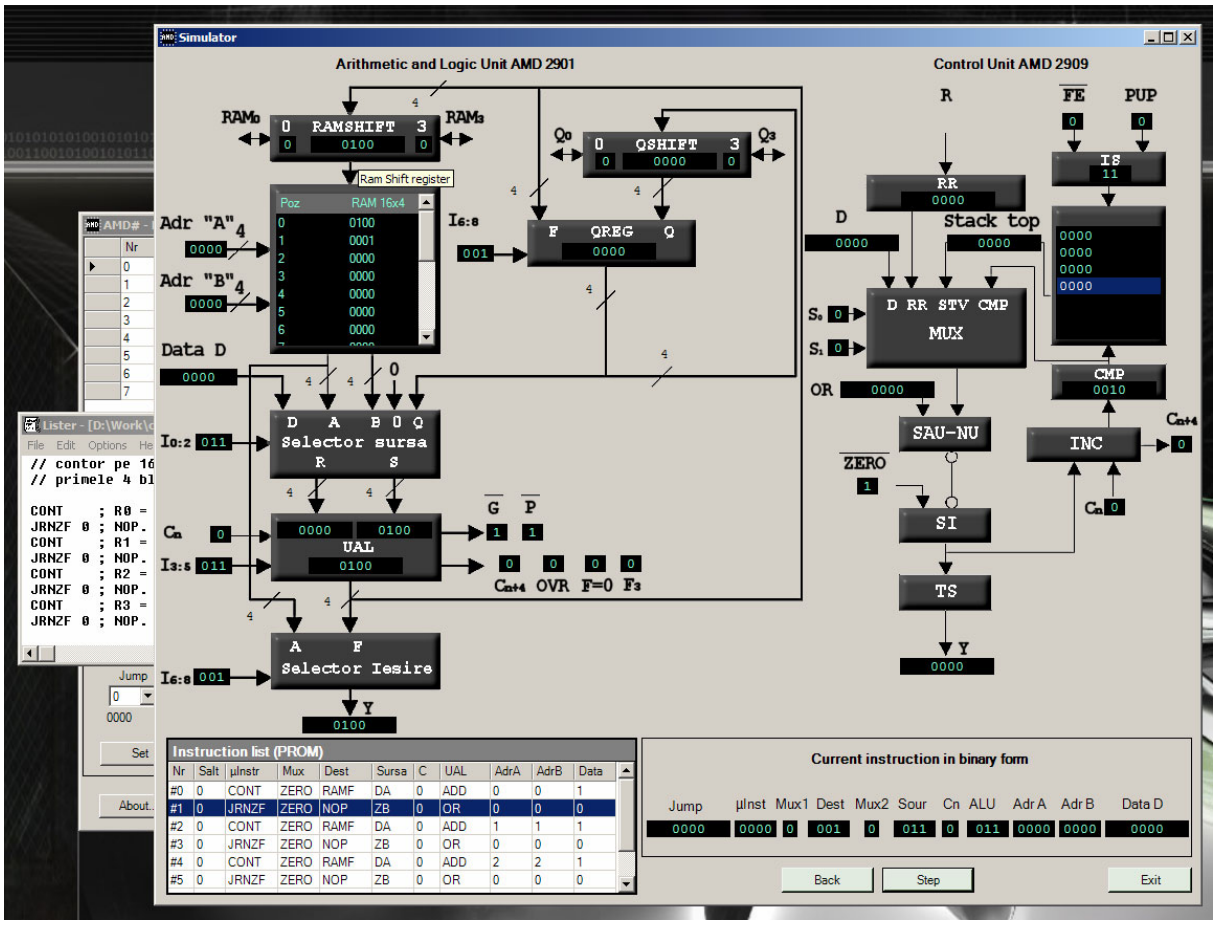


Figure 7. The new "block diagram" simulator interface