Interactive Applets for Introductory Courses on Computer Architecture

Authors:

Alberto González Téllez, Universidad Politécnica de Valencia, Valencia 46022, Spain agt@disca.upv.es
Juan Carlos Martínez González, Universidad Politécnica de Valencia, Valencia 46022, Spain jc@disca.upv.es

Abstract — *The basic concepts in the field of computer architecture have proved to be difficult to teach by just describing them. The discrete nature of processor behaviour, and the lack of powerful abstract models for modelling this behaviour, means that a complete detailed description is necessary. Many students find passive classroom study insufficiently motivating and interesting - specially in subjects such as: assembly language programming, input-output management, data-path behaviour, etc. To practice with real systems is the ideal solution but real systems are often costly or too complex. Interactive simulators do not have these disadvantages and then they are a very valuable complement of lectures and readings, because they allow the student to try and to see what happens. This interactive practice with a simplified representation of a computer, that mimics a real one, increases the student interest in the field and makes his understanding more complete and durable. We present a Java applet that supports student understanding of assembly language programming. After testing its success among students it will serve as a base to develop more applets oriented to other computer architecture issues. We have chosen Java technology because it is integrated in the web and then universally available. Java applets are also easier to maintain and distribute than native applications, and they can be embedded in web documents.*

Index Terms — *Computer architecture, interactive teaching tools, Java applets, simulation.*

INTRODUCTION

Introductory courses in computer architecture are challenging due to the students lack of previous knowledge in the field and the nature of the subject, quite different than more classic disciplines like physics, calculus, algebra, etc. Computer systems are complex devices with a discrete behaviour, its complexity and non-continuos features take them apart from the kind of systems (mechanical, electrical, etc) that the students are used to.

We find also that the state of the art does not provide with abstract tools to face the problem. Then the explanation of the structure and the behaviour of computer systems is based on detailed descriptions. This tends to be inevitably boring and useless if students are not able to “try and see”. In computer architecture is certainly true that the only way to achieve an effective learning, particularly in introductory courses, is “learning by doing” because “learning by earring/reading” is definitively insufficient.

Then the question is how to provide the students with tools that allow them to practice. The best choice in a first approach is “real computers”, but this has two main disadvantages: real computers are too complex for being used in introductory courses and they are expensive, and then their availability is limited. An alternative to real systems are simulators, in fact they solve the two disadvantages mentioned, because the complexity can be limited and they have no cost as freely available software.

After having chosen the simulator alternative there are some decisions that remain: to use already implemented simulators or to design new ones, if a new one has to be implemented then a programming language has to be selected. Unfortunately there are not enough simulators available to fit all the particular needs that can be found in teaching practice, anyway this has to be the first attempt because it will avoid the expend of a lot of time on the teacher side. If this attempt is unsuccessful then we have to decide what tools we will use to implement our custom simulator.

The tools available to design interactive simulators can be of two types: multimedia authoring tools and general purpose programming environments. Multimedia authoring tools are much easier to use and are maybe the only feasible choice to non programming experts. Unfortunately they have some disadvantages, they are very expensive and use proprietary formats. Computer architecture teachers have the advantage of not being bound by this unique choice because they have a background on general purpose programming languages. These environments are much more available and can be less platform dependent, this is particularly true is we choose Java.

The Java language was born thinking on Internet instead of a particular operating system, then Java programs are portable and even more they can be embedded in web pages, in the form of applets. There are also freely available programming tools like Sun SDK[1] and particularly JBuilder Personal Edition from Borland[2]. The later one includes all the components required to design applets based on the AWT[3] and Swing toolkits, it also offers a very good visual developing environment. This is the tool that we have decided to use to develop our simulators as Java applets.

Simulator applets can be used by means of Java aware web clients like Internet Explorer with Microsoft JVM or any other navigator with a Java plug-in available (almost every one). This guaranties that students can use the simulator on lab PCs and at home, even with modem connections due to the small size of applets (less than 100Kbytes). The only required maintenance is on the server where the applets are installed. To avoid Internet connection applets can also be installed locally by simply copying some files: the html file that embeds the applet, the applet library file and maybe some auxiliary files.

THE EASY8 COMPUTER

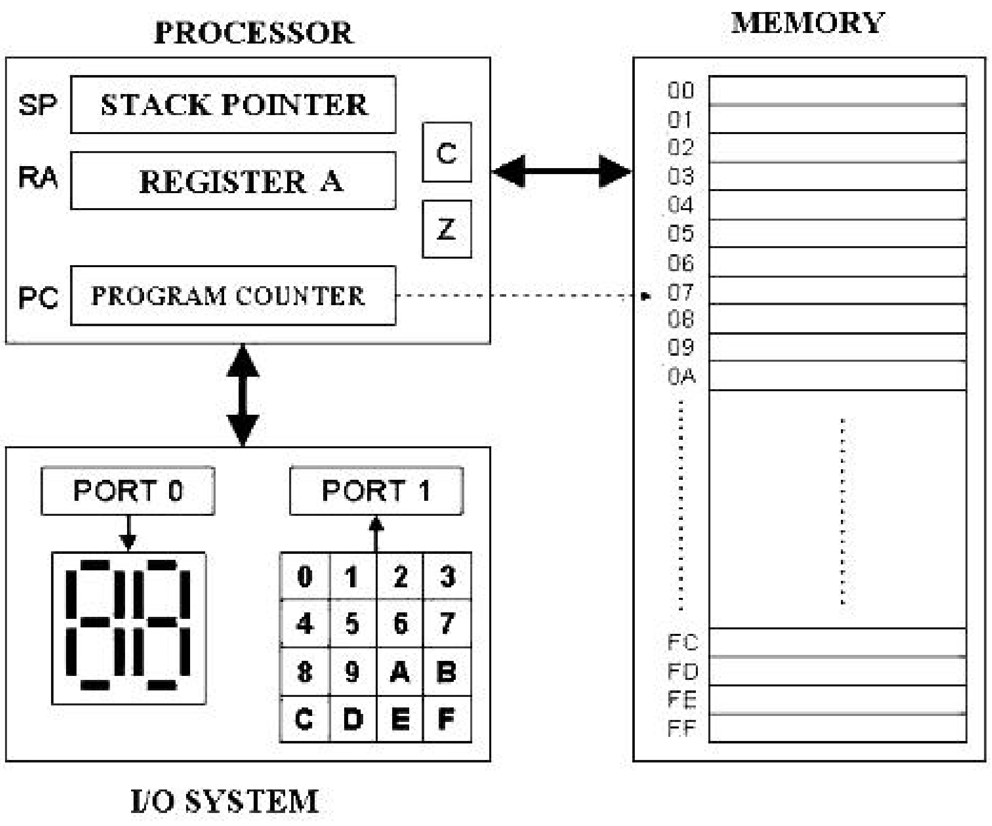
In this section, we explain the architecture of the Easy8 processor and a simple computer made upon it, also named Easy8. First we describe the functional units of the Easy8 computer and the memory and input/output subsystems properties. Then we explain the instruction formats and the instruction set.

Easy8 architecture

Easy8 is an academic processor, so we made it simple. Then we included an small number of registers and state bits. Easy8 is an extension of the Easy4 processor [6]. Easy4 is a four-bit academic processor, whose register size is four bits, so the address bus is also four bits wide. The main consequence of this limitation is the memory size that is just sixteen words. The memory word width is four bits. The number of I/O ports is sixteen too. The number of instructions is limited to sixteen, because the operation code in the instruction format is one word.

Easy8 uses eight-bit registers, so memory system size is 256-words, eight bits wide. The number of I/O ports also increases to 256, and the Easy8 instruction set can have up to 256 instructions.

FIGURE 1
EASY8 COMPUTER COMPONENTS.



In figure 1, we can see the Easy8 architecture. It is composed by three subsystems: Processor, Memory, and I/O system. Memory system size is 256 bytes and I/O system has 256 ports, each port can transmit one byte. The I/O system defined in the simulator has two devices. The first is a hexadecimal display that can show two hexadecimal digits. This device uses port 0 as an output port, so when the processor writes a word in this port the display shows this number using hexadecimal representation. The second device is an input hexadecimal keyboard. This device uses port 1 as an input port. When the processor reads this port the last two digits pressed, as a byte.

The processor has three registers. There is only one data register (RA) that acts as an accumulator register (source and destination simultaneously). The Program Counter (PC) is used as a pointer to the next instruction to execute, this register is incremented automatically by the processor at beginning of every instruction. The Stack Pointer (SP) is used to implement a stack at the bottom of the memory space. SP is set to 0 when the processor is reset. Before inserting a word in the stack SP is decremented by one. After extracting the last word inserted into SP is incremented by 1.

The system has two flags for arithmetic and compares: Carry (C) and Zero (Z) bits. The first one is set to the value of the most significant carry bit of the arithmetic unit. The second one is set to one when the result of the last arithmetic operation is zero, otherwise Z is set to zero.

Easy8 instruction set.

Before enumerating the instruction set of the Easy8 processor, we explain its instruction formats. The Easy8 processor uses two instruction formats:

- **One word instruction format:** It uses only one byte that contains the operation code of the instruction. If any operand is needed it has to be an implicit operand, usually RA.
- **Two words instruction format:** It uses two bytes. The first one is the operation code of the instruction. The second one is an immediate or a direct operand. This operand is an eight-bit data or an eight bit address, respectively. If the instruction uses more than one operand, the remaining operands have to be implicit operands (usually RA).

Next, we enumerate and explain every instruction of the Easy8 processor. The Easy8 instruction set has 22 instructions, they are described in table 1.

USING THE EASY8 SIMULATOR

As you can see in figure 2, the simulator is made by four panels: Editor, Memory, Processor, and I/O devices. The first panel is the assembler editor. Programmers can use the list and the buttons to insert, modify, and delete instructions in the editor. Once the program is completely edited it can be assembled using the assemble button. If the virtual machine is properly configured to allow file access, as described in table 2, then loading and writing files can be done by means of the load and save buttons, naming the file in the text field “File”. The solution of the exercises proposed in the subject lectures are also available selecting in the choice list “Exercises” then the solutions are loaded from the server and they appear in the choice list.

The second panel shows the memory content. This window is split in two, in order to be able to show the contents of the memory at the SP and PC pointers. Users can change the content of any word in memory using the *Addr* and *Val* fields and the button *OK* at the bottom of the column.

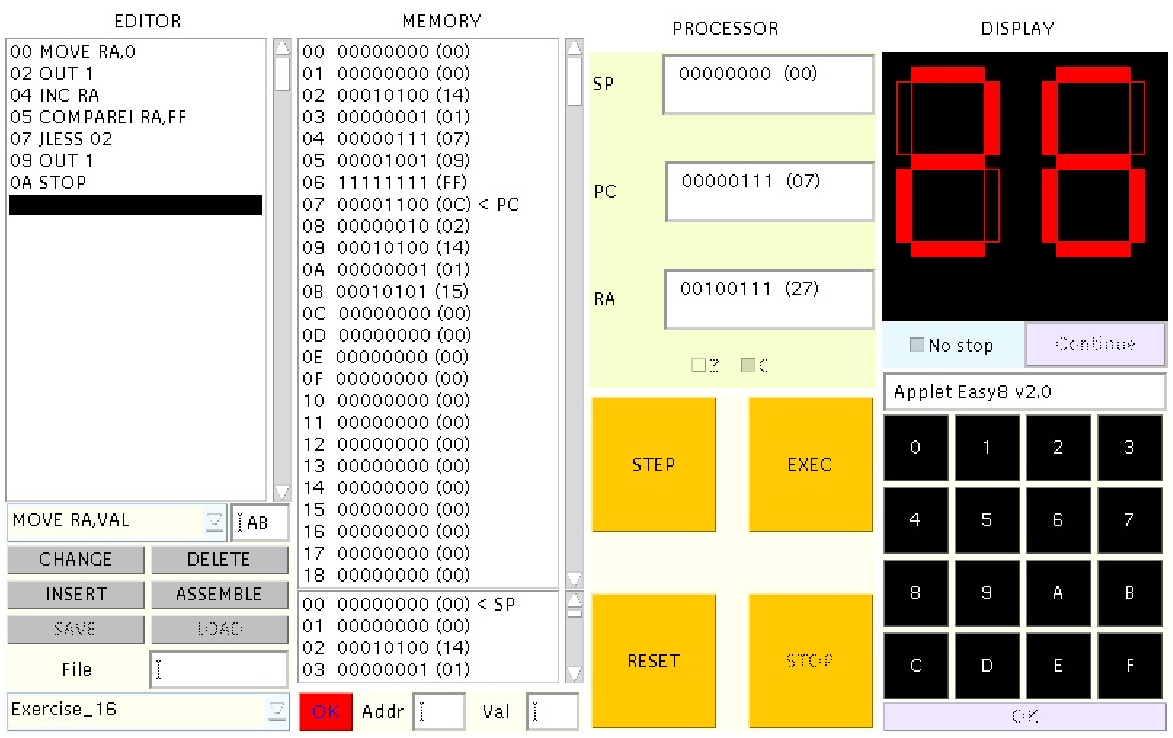
The third panel shows the contents of the processor registers and flags. There are four buttons. The *RUN* button starts the simulation of the program assembled, the simulation ends when a STOP instruction is executed or the *STOP* button is pushed. The *STEP* button allows to execute the instruction pointed by PC. The *RESET* button sets the contents of every processor register to zero, Z and C bits are also reset.

The last panel contains the I/O devices. The first I/O device is a hexadecimal display. When the processor writes a byte to the output port 0, this byte is shown in the display. If the “No stop” box is not checked, the simulator pauses the program until the user pushes the “Continue” button. The second device is a hexadecimal keyboard. This device is connected to the input port 1. Simulation pauses when an input operation is made to the port 1. Then, the user has to introduce two digits in the keyboard, pressing OK to advice the processor to continue.

TABLE 1
EASY8 INSTRUCTION SET.

Group	Instruction	Operation Code	Description
Control instruction	STOP	0x15	Stop simulator.
Transfer instructions	MOVE RA, VALUE	0x00	Exchange data between Memory and processor. The first operand is the destination operand.
	MOVE RA, [ADDRESS]	0x01	
	MOVE [ADDRESS], RA	0x02	
Arithmetic instructions	ADDI RA, VALUE	0x03	Arithmetic instructions using immediate or direct addressing. INC and DEC add and subtract 1 to RA, respectively
	ADD RA, [ADDRESS]	0x04	
	SUBI RA,VALUE	0x05	
	SUB RA, [ADDRESS]	0x06	
	INC RA	0x07	
	DEC RA	0x08	
Compare instructions	COMPAREI RA, VALUE	0x09	These instructions do the same that SUBI and SUB. The difference is that compare instructions do not write result in RA, but the flags are modified.
	COMPARE RA, [ADDRESS]	0x0A	
Jump instructions	JUMP ADDRESS	0x0B	The first one is an unconditional jump. In this case the jump is always taken.
	JLESS ADDRESS	0x0C	
	JGREATER ADDRESS	0x0D	The other instructions are conditional jumps. To evaluate the condition, these instructions check the flags: Z=1 → Equal; Z=0 & C=0 → Greater; C=1 → Less
	JEQUAL ADDRESS	0x0E	
Stack instructions	PUSH RA	0x0F	The first instruction inserts one byte at the top of the stack. The second one gets the last byte inserted in the stack. SP is updated accordingly.
	POP RA	0x10	
Subroutine instructions	CALL ADDRESS	0x11	The first instructions calls the subroutine located at the specified address and put the return address in the stack. RET gets the return address from the stack and writes it to the PC register.
	RET	0x12	
I/O Instructions	IN PORT	0x13	These are the Easy8 input and the output instructions. Data are got from or put into the RA register.
	OUT PORT	0x14	

FIGURE 2
EASY8 SIMULATOR APPLET.



```
EDITOR
00 MOVE RA,0
02 OUT 1
04 INC RA
05 COMPAREI RA,FF
07 JLESS 02
09 OUT 1
0A STOP
```

```
MEMORY
00  00000000 (00)
01  00000000 (00)
02  00010100 (14)
03  00000001 (01)
04  00000111 (07)
05  00001001 (09)
06  11111111 (FF)
07  00001100 (0C) < PC
08  00000010 (02)
09  00010100 (14)
0A  00000001 (01)
0B  00010101 (15)
0C  00000000 (00)
0D  00000000 (00)
0E  00000000 (00)
0F  00000000 (00)
10  00000000 (00)
11  00000000 (00)
12  00000000 (00)
13  00000000 (00)
14  00000000 (00)
15  00000000 (00)
16  00000000 (00)
17  00000000 (00)
18  00000000 (00)

00  00000000 (00) < SP
01  00000000 (00)
02  00010100 (14)
03  00000001 (01)
```

```
PROCESSOR
SP  00000000  (00)
PC  00000111  (07)
RA  00100111  (27)
```

TABLE 2
JAVA VIRTUAL MACHINE CONFIGURATION TO ALLOW FILE SYSTEM ACCESS.

Java Virtual Machine	Configuration steps
Microsoft JVM in Internet Explorer (Windows 98, ME and 2000)	1. Open Internet Explorer and do Tools->Internet options and select the Security panel 2. Select the icon "Local Intranet" and press the "Sites" button, then press “Advanced Options” and add the server URL (http://poseidon.disca.upv.es) . The “https” requirement has to be unselected. 3. Press "Custom level" y look for the "Microsoft VM" section, in “Java permissions” select “Custom” 4. Press "Java custom configuration" and in the "Permissions edit" panel choose in the “Unsigned content” section the box "Execute inside the sandbox" and in the subsection "Access all files" select "Activate" 5. Finally... press OK in all windows and restart Internet Explorer
Java 2 Plug-in (this will be the same in every web client on every operating system that has a Java 2 plug-in available)	1. Execute the program "policytool" located in the subfolder "bin" of the plug-in installation folder (i.e. c:\Program files\java\j2re1.4.1_01\bin) 2. In the "Rules file" text field write ".java.policy" 3. Press in "Add rule entry". In the opened window write the server URL in the text field "Code base" (http://poseidon.disca.upv.es) and then press "Add permission" In "Permission" select "File permissions" In "Destiny name" select "ALL FILES" In "Actions" select "read" and then "write" Press "OK" 4. Finally... Press "Done" and do File->Save, then close the program

APPLET INSTALLATION

The Easy8 simulator applet is made up of three files:

- *The HTML file that embeds the applet* This is achieved by means of the HTML standard tag <applet>. The content of the HTML file is:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
<TITLE>
Easy8 simulator
</TITLE>
</HEAD>
<BODY>
<APPLET
  CODEBASE = "."
  ARCHIVE  = "e8asm.jar"
  CODE     = "e8asm.e8asmapp.class"
  NAME     = "Easy8CU"
  WIDTH    = 800
  HEIGHT   = 500
  HSPACE   = 0
  VSPACE   = 0
  ALIGN    = middle
>
<PARAM NAME="lang" VALUE="en">
</APPLET>
</BODY>
</HTML>
```

  The attributes ARCHIVE, CODE, WIDTH and HEIGHT specify the applet library file, the main Java class and the size of the applet panel, respectively. The parameter “lang” specifies the language (“es” for Spanish, “ca” for Catalan and “en” for English).
- *The applet library file* It is the file named “e8asm.jar” it includes all the Java classes required in binary format.
- *The exercise solution file* It is an XML file (ejercicios.xml) that uses a customised Docbook grammar[7]. It is automatically generated from the class notes also developed in the same language. Only the <programlisting> tag content is relevant but the file has to be “well formed” according to XML[8] following the structure in the next listing.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Theme
  PUBLIC "-//DISCA//DTD TeachdB DocBook XML V4.1.2.5//EN"
"http://www.disca.upv.es/xml/dtd/theme.dtd">
<Theme lang="es">
  <Content>
    <Section>
      <Exercise>
        <Enunciate>
          <para>Realiza un programa que calcule 9 + 5 - 2, dejando el
          resultado en el registro RA</para>
        </Enunciate>
        <Solution>
          <programlisting>
00 MOVE  RA, 9
02 ADDI  RA, 5
04 SUBI  RA, 2
06 STOP
          </programlisting>
        </Solution>
    </Section>
  </Content>
</Theme>
```

The applet installation on the web server consists on copying these three files in a subfolder on the server document root folder. To load the applet in the web client you have to append, to the server URL, the HTML relative path to the document root folder. In our case the server URL is:

`http://poseidon.disca.upv.es`

The three applet files mentioned are located in the next path relative to the server document root:

`fct/e8asm`

Then the complete URL to load the applet in the web client is:

`http://poseidon.disca.upv.es/fct/e8asm/e8asm.html`

The HTML file that includes the applet can also be linked from a default file that gives an explanation about the applet features and use. In our case this default file is opened by the URL:

`http://poseidon.disca.upv.es/fct/e8asm`

In order avoid the need of an internet connection to use the applet, it can also be installed in the local file system. This is done by simply copying the three files (e8asm.html, e8asm.jar and ejercicios.xml), around 60KBytes, in a folder. Then the applet is started by loading in the web client the "e8asm.html" file. The disadvantage of this alternative is that applet updates are not automatically performed.

CONCLUSIONS AND FUTURE WORK

The educational simulator presented has been very useful and effective in teaching introductory concepts and techniques about assembly language programming. Furthermore it has received a very enthusiastic acceptance by the students. On this grounds we plan to develop applets to cover other aspects of the subject like: binary data representation, cache memory, I/O interrupts and DMA, basic operating system services, etc.

We plan also to complement the applets with server side applications using Java servlets. This will allow to create in the server a folder for every student in which they will be able to save their work, in this way not only the simulator but the data that they generate will be available from the server. It will also allow us to follow the students use of the tools and their realisations.

ACKNOWLEDGEMENT

The work presented has been supported by the EUROPA program of the Universidad Politécnica de Valencia.

REFERENCES

[1] Sun Microsystems Java web site, "http://java.sun.com"

[2] JBuilder Borland web site, "http://www.borland.com/products/downloads/download_jbuilder.html"

[3] Zukowski, J., *Java AWT Reference,* O'Reilly, 1997

[4] Niemeyer, P., Knuds, J., *Learning Java (2nd edition),* O'Reilly, 2002

[5] Easy8 applet web site, "http://poseidon.disca.upv.es/fct/e8asm"

[6] Hergert, D., Thibeault, N., *PC architecture from assembly language to C,* Prentice Hall, 1997

[7] Docbook web site, "http://www.docbook.org"

[8] eXtened Markup Language spec 1.0, "http://www.w3.org/TR/REC-xml"