

A Visual Simulator for Understanding Structure and Behavior of Computer

Yoshiro Imai, Shinji Tomita¹, Hitoshi Inomo, Zengo Furukawa,
Wataru Shiraki, Hiroshi Ishikawa and Akiyoshi Miyatake²

Faculty of Engineering, Kagawa University

Graduate School of Informatics, Kyoto University¹

Takuma National College of Technology²

{imai, inomo, zengo, shiraki, ishikawa}@eng.kagawa-u.ac.jp
tomita@lab3.kuis.kyoto-u.ac.jp, miyatake@di.takuma-ct.ac.jp

Abstract

Computer systems become more powerful and more popular. It is not easy for computer users to comprehend how computers work because of their complex properties like as black box. It is necessary to understand the internal structure and behavior of computer. We have developed a visual simulator written in Java, which can illustrate graphical computer structure, interpret sample assembly programs, demonstrate data transfer between registers of CPU and memory, and visualize computer's internal mechanism. This paper describes that our visual simulator will be a useful tool at computer introductory courses. It can realize graphical demonstration for lecturers as well as self-learning software for students.

1. Introduction

Recently several curricula have information literacy education courses and introductory lectures for computer science. On the other hand, it is too difficult for beginners to comprehend how computers work because of their complex properties like as black box. In order to study computer more precisely, it is necessary to understand the internal structure and behavior of computer. Any abstractive lectures on Computer System cannot provide visual and applicable understanding on structure and behavior of computer.

We have developed a visual simulator called "VIsuSIM (/vizim/)" as computer education tool. It is designed for understanding internal structure and behavior of computer

visually. VIsuSIM offers window-based graphical view to explain how computer works, interprets sample assembly programs stored in memory and demonstrates data transfer mechanism between registers of CPU and memory.

This paper describes basic design concepts of our visual simulator, its system configuration, characteristics of its GUI for manipulation, and practical application to classroom lecture of computer system and/or computer literacy. And it is illustrated that VIsuSIM will be a useful tool at computer introductory courses.

2. Basic Concepts of Visual Simulator Design

Our visual simulator: VIsuSIM has been designed and developed for the following objectives:

- 1) A demonstration tool for assistance to complete oral explanation how computer works in ordinary classroom lectures,
- 2) Self-learning software, which can be obtained from web server, to review and confirm contents of lecture after school.

These design objectives may be suitable for not only our visual simulator but also almost education tools or self-learning softwares.

It is important that education tool will be available as widely as possible and can be manipulated easily. If such a tool is designed only for a handful of specific users, it cannot enjoy periodic improvement for lack of feedback and request from other users.

We think that Java programming language and its products can provide us the above benefits such as machine-independent execution environment, simple GUI building facilities and so on. And moreover we can take advantage of low-cost software development environment. We had decided to design and implement education-oriented software by means of Java language and JDK: Java Development Kit from Sun Microsystems Inc. It is why our visual simulator has been developed as one of the Java applications.

And then we will mention the basic concepts to design our visual simulator. At first, basic design concepts of our simulator are enumerated, and secondly commented for each item as follows:

- 1) Education-aid demonstration tool
- 2) Self-learning assistant software
- 3) Enhancement of visual facility
- 4) Graphical user interface
- 5) Easy distribution and simple manipulation
- 6) Employment of Java programming language
- 7) Browser-based execution environment

Almost teachers will want to improve their classroom lectures and their education-aid tools which can help their students to understand their lectures. Item 1) is one of the most important incentives to build education-aids tools. Especially, lecture of computer systems has been suffering from lack of demonstration tool to explain how computers work. Item 2) is second but not least motivation to possibly give students a kind of benefit free for some constraints of classroom-oriented lecture. Although students were absent from school, they can use such a software as self-learning tool and understand content of lecture suitably.

Both of items 3) and 4) are the key concepts to design recent softwares. Visual facility and GUI are essential technologies to implement several functions of software. Because all the PC's have GUI-based operating systems, so application programs will never avoid functions such as visual capability and graphical interface.

Item 5) includes guidelines to design software which will be available for several people. Recently distribution of software may utilize network connectivity of computer. Web sites provide downloadable data and binaries on their home pages, accept access from anyone through network, and transfer data or binaries according to request. So it is

quite natural that we choose a design policy to utilize software distribution style based on network and web data transfer service. We must choose another design policy to build software based on windows-based operability with simple manipulation such as button pushing and keyword writing, because all the PC's already adopt multi-windows system.

Selecting programming language is one of the most serious problems whether system and/or software will be successful or not. In the general, C/C++, Visual BASIC or Java may be powerful and practical candidate of software description languages. With most regard to cumulative results till now, C/C++ must be chosen. Selection based on popularity will persuade us that Visual BASIC has obtained maximal numbers of users in the PC world. We decide to choose Java as description language, however, because Java can provide both of environment-independent executable binaries and window-based operability for multiple platforms. It is why item 6) has been included into our design concepts.

Item 7) specifies that software to be developed may possess property of platform-independency. Because All the PC's have been installed to utilize a browser such as MS-IE, Netscape Communicator, and so on. Almost browser can interpret Java byte code and prepare an execution environment for Java applet.

3. Implementation of VIsuSIM with Java

The configuration of our visual simulator, VIsuSIM, includes the following subsystems:

- 1) GUI Components + Layout,
- 2) Routines for Standalone Apps,
- 3) Routines for Applet,
- 4) Several Threads,
- 5) Event Handlers, and
- 6) Simulator (body).

Subsystem 1) plays a role of user interface, whose components consists of bottoms , labels, and text fields for information interchanging between Java applet and users. Layout may be one of the most important factors to define practice of GUI and to determine whether according system is easy to use or not. With Java programming style, it is easy to select suitable GUI components and put them in effective position.

Subsystem 2) and 3) can work when VIsuSIM is invoked as standalone application of Java and when it is invoked as Java applet, respectively. The both routines reset internal states and several variables, and prepare for services such as initializing, loading program, interpreting and so on. Our code for visual simulator, VIsuSIM, includes both of two subsystems, so that two-way usages can be supported by a single code. Namely, our simulator can work correctively as either standalone application or Java applet. Additionally, it is convenient to update and easy to maintain our code because both of subsystem 2) and 3) are included in only single code of VIsuSIM.

Subsystem 4) has some threads, which are introduced for VIsuSIM to operate concurrently. It is designed and implemented according to Java formal thread description, because we think it is difficult to write corrective program of Java for operative concurrency. It can realize that both simulating and drawing are concurrently carried out.

Subsystem 5) is very much principal in VIsuSIM because event-handling routines play essential roles for interaction between user and other routines. This subsystem will be invoked to start event handling service, process and calculate suitably, and then return the results in accordance with events.

Subsystem 6) is the main part of VIsuSIM. It consists of the following three major routines: instruction fetch routine, instruction decode routine and execution routine. All three routines and the following data area and text fields organize virtual hardware of computer according to real hardware structure. Instruction fetch routine transfers instruction stored in memory array into instruction register(IR) built in control unit according to the specification of program counter(PC). Then instruction decode routine reads the content of IR, analyzes it into operation code and operands, and throws decoded signals to the suitable units. Those units recognize what they must do by means of signals from decode routine. Finally, execution routine indicates that every unit should operate correctively along the received order. The three addressing modes such as direct addressing, indirect one and immediate one, also interpreted by execution routine.

4. Simulation Capability of VIsuSIM

This section describes detail of simulation capability

concerning VIsuSIM. Its capability may be partly evaluated by means of instruction repertory which an objective simulator can interpret. Namely, instruction set of target system will be one of measurements for simulation capability. Table 1 shows instruction set of our visual simulator, VIsuSIM.

Table 1 Instruction set interpreted by VIsuSIM

Instruction Set defined for VIsuSIM			
	Opcod	1st Operand	2nd Operand
Control Instruction	halt		
	noop		
Jump Instruction	jump	DirectAddr RegId IndirAddr(Reg)	
	jpgt		
	jpge		
	jplt		
	jple		
	jpeq		
	jpne		
call			
ret			
Unary Op. Instruction	neg	DirectAddr	
	push	RegId	
	pop	IndirAddr(Reg)	
Binary Op Instruction	add	DirectAddr RegId	DirectAddr RegId
	sub	IndirAddr(Reg)	IndirAddr(Reg)
	move	ImmedVal	
	and or xor	DirectAddr RegId IndirAddr(Reg) ImmedVal	DirectAddr RegId IndirAddr(Reg)

Instruction repertory can be divided into following four groups:

- Control instructions including **halt** and **noop** (No Operation),
- Jump instructions including subroutine-related, conditional-jump and unconditional-jump operations such as **call**, **ret**, **jpgt**, **jpge**, **jplt**, **jple**, **jpeq**, **jpne**, and **jump**,
- Unary operation instructions including arithmetic and stack-related operations such as **neg**, **push** and **pop**, and

- Binary operation instructions including arithmetic operations such as **add**, **sub**, **move** and logical operations such as **and**, **or**, **xor**.

Each operations work simultaneously together with condition-code register, which consists of Negative flag and Zero flag. Our condition-code register is represented as (N, Z)=(-, -), (-, Z) or (N, -).

All the instructions above may take maximum two operands, which can be specified as the following four ways: direct address (denoted **DirectAddr** in table 1), register identification (denoted **RegId**), indirect address with register modification (denoted **IndirAddr(Reg)**) and immediate value(denoted **ImmedVal**). Direct address is most normal addressing, which uses only operand field and specifies target location of memory. Register identification is another addressing which means using register as source or destination of operation. **RegId** or **Reg** is defined to represent specific register from GR0 to GR7 in our case. Indirect address with register modification is more powerful addressing so that it is indispensable for utilization of indexed addressing, for example. With this addressing, complicated iteration processing will be available on our visual simulator. Immediate value is a very convenient addressing and let us describe brief assembly programs. Without this mode, we must secure location of memory for all the data to be manipulated in programs.

Figure 1 shows a sample assembly program which can be interpreted by VIsuSIM. The program represents a sample of utilizing indexed addressing in order to calculate summation of five numbers (723, 1217, 320, 802 and 820) with iteration method. Notation and syntax of our assembly language will be briefly explained as follows.

```

0: move #0, GR0
1: move GR0, 20 // s = 0
2: move GR0, 21 // i = 0
//return:
3: move 20, GR0
4: move 21, GR1
5: add 22(GR1), GR0
6: move GR0, 20 // s = s + a[i]
7: add #1, GR1
8: move GR1, 21 // i = i + 1
9: sub #5, GR1 // ? (i - 5)

```

```

10: jpeq 12 // if (i - 5)==0 goto equal5
11: jump 3 // goto return
//equal5:
12: halt
20: ds 1 //s:
21: ds 1 //i:
22: 723 //a:
1217
320
802
820

```

Figure 1 A sample program for indexed addressing

By means of writing the number and colon (i.e. “num:”) at front of each line, we can explicitly specify location of memory where the concerned instruction will be loaded. Notation sharp (#) can represent immediate value for operand. We can perform comment-out for the statement which follows notation double slash (/). Statement “ds 1” is pseudo-operation, defining storage of one location. It is used to secure location of memory for data to be stored.

In order to understand internal structure and behavior of computer, we think that it must be necessary to comprehend addressing mechanism as well as jump operation mechanism. We think that students who can distinguish the three addressing mode clearly will pass through the first stage of understanding computer. In the usual lectures about computer system or computer literacy, we did perform an abstractive lecture to explain how computer works without demonstration tool. In such a case, even though explanation about addressing is no more than introductory level in the lecture of computer system, it is difficult to illustrate detail of addressing mechanism so that we cannot give a visual relation between actual behavior of computer and practical addressing. This time, it is useful to utilize our visual simulator for concrete explanation of addressing mechanism.

5. GUI and its Operability of VIsuSIM

First of all, we show the overview of VIsuSIM in figure 2. This figure demonstrates that VIsuSIM is working on the browsing window of Internet Explorer, which is a Japanese Edition bundled in Microsoft Windows 98. Needless to say, another major browser such as Netscape

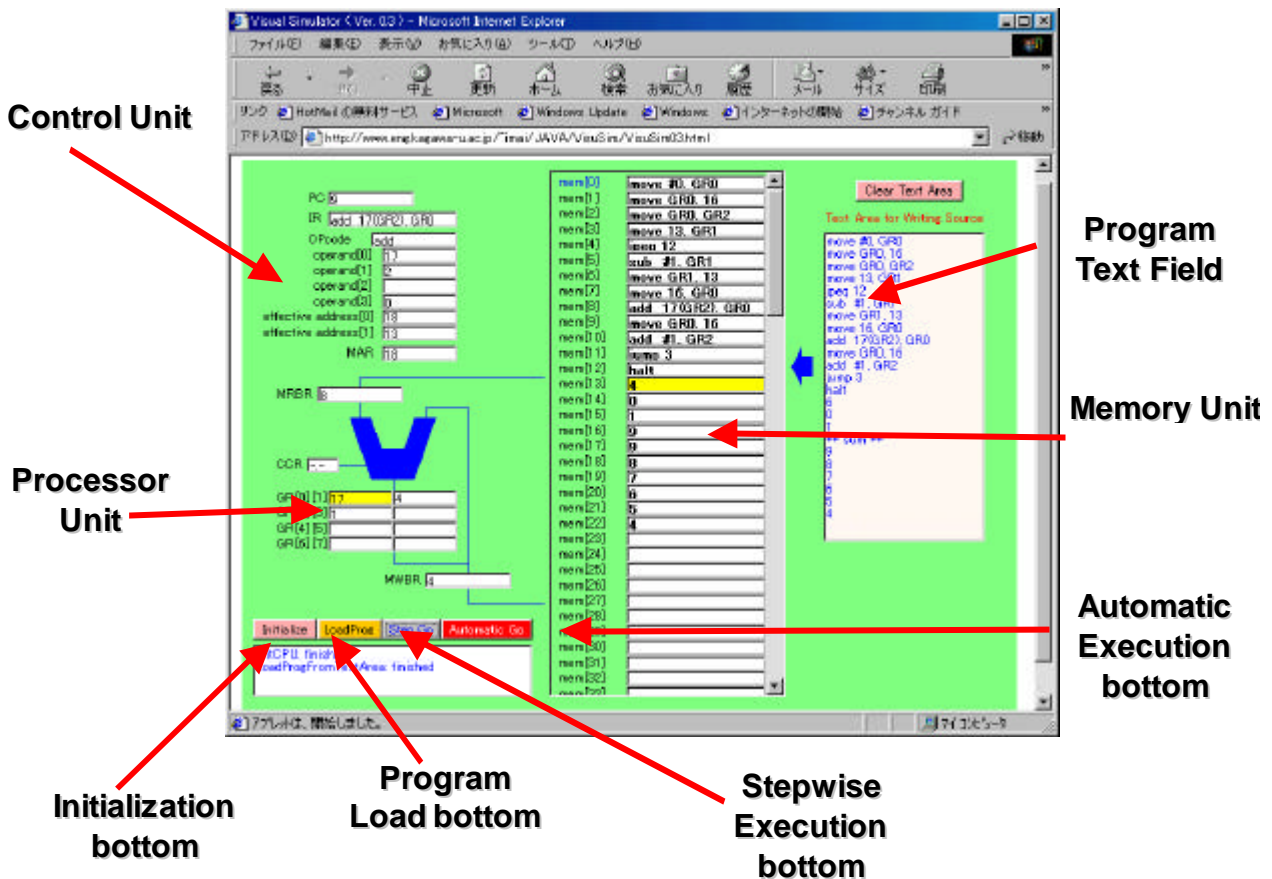


Figure 2 GUI overview of VisuSIM on the window of Internet Explorer

Composer can also provide an execution environment for VisuSIM. In such a case, VisuSIM is invoked as Java applet so that it has hidden some components for file access service facilities in order to inhibit security violation.

The GUI window of VisuSIM consists of major three parts for computer hardware and some components for interaction between user and VisuSIM. As shown in figure2, the major three parts for computer hardware are represented with control unit, processor unit and memory unit. Control unit has some objects constructed with text fields which play roles of PC, IR and other registers respectively. Processor unit as well as control unit has some objects constructed with text fields which play roles of General-purpose registers (GR[0]-GR[7]), two-type memory registers (MRBR, MWBR), Condition-code register (CCR) and so on. Memory unit is nearly the same. It has been entirely implemented on slide-movable panel object so that it can show location of array-structured memory cells partly and view of contents can be slid as the occasion demands.

VisuSIM has four bottom objects such as Initialization bottom, Program Load one, Step-wise Execution one, and Automatic Execution one. These are prepared to control VisuSIM from user and accept external requests. For example, pushing Initialization bottom, the major three parts of VisuSIM are reset so that IR, eight General-purpose registers, all the memory cells has been clear and PC is set into zero. Program Load bottom is used for transferring an assembly program written in Program text field into memory unit. If VisuSIM is invoked as standalone Java application, pushing of Program Load bottom can perform direct access to the file system and read a program stored in file, and transfer it into memory unit. The two execution bottoms are used for start and stop simulator subsystem of VisuSIM directly. One is for step-wise execution instruction by instruction, and another is for automatically executing a series of assembly codes until **halt** instruction in the codes is fetched and decoded.

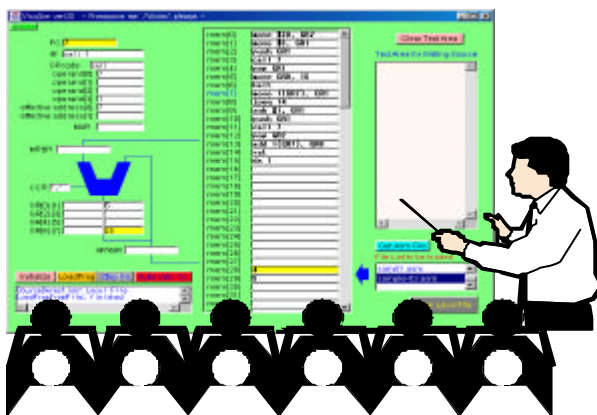
A message field below four bottoms is placed to display current internal state of VisuSIM. After pushing bottom, request from user is accepted and processed by according

routines of VIsuSIM, which generates corresponding message related to its current internal state, such message is like “LoadProgFromTextArea: finished.”

6. Evaluation for Lecture with VIsuSIM

There are some ideas of indication to make easy understanding of computer's behavior. We have two types of indicating function for memory access. Color for the memory location turns to blue when data fetching happen. And color of memory cell has changed into yellow on storing data at that location. In the same manner, color of register has changed into yellow on saving data in that register. With these functions, it is more effective to comprehend how programs are processed and then computer works. Figure 3 illustrates schematic diagram for demonstration of lecture with education-aid tool such as VIsuSIM. And photo 1 shows a scene of actual lecture using VIsuSIM.

Figure 3 Demonstration of Lecture with VIsuSIM



Execution modes of VIsuSIM are twofold: stepwise type and automatic one are summarized as follows;

- 1) Stepwise Execution: When the bottom for “Step Go” is clicked, one machine cycle execution of simulator in VIsuSIM has occurred. A machine cycle includes instruction fetch, instruction decoding and execution. This mode is convenient for interactive modification of registers and/or memory and advance of next machine cycle, after one machine cycle has finished.
- 2) Automatic Execution: When the bottom for “Automatic Go” is clicked, continuous repetition of machine cycle execution of simulator until decoding **halt** instruction or

execution interruption, which is occurred on pushing the bottom for “Automatic Go” again while simulator of VIsuSIM repeats its machine cycles in the mode of automatic execution. This mode is also convenient for relatively long-term demonstration to interpret program with several iterations in VIsuSIM, which looks like playing slide show.



Photo1 Scene of Actual Lecture Using VIsuSIM

7. Concluding Remarks

We can conclude our study as follows:

- 1) With our visual simulator, visual demonstration can be realized in classroom lecture, so that it is efficient for students to understand internal structure and behavior of computer precisely.
- 2) Visual simulator can be easily obtained from the web-site, sufficiently executed platform-independently, and simply operated by means of GUI and Interactive facilities.

Acknowledgements

The authors would like to express special thanks to Prof. Haruo Niimi, Kyoto Sangyo Univ., and Prof. Kazunori Yamaguchi, Univ. of Tokyo for their constructive advices and fruitful instructions. This study is partly supported by grand 12040107 in aid for scientific research from the Ministry of Education, Culture, Sports, Science and Technology.

References

- [1] Imai, Tomita et.al., "A Visual Simulator for Computer System Education(in Japanese)", IEICE Tech. Report ET-99-110, pp.113-120, 2000
- [2] <http://www.eng.kagawa-u.ac.jp/~imai/JAVA/VisuSim/>