Serdica J. Computing **8** (2014), No 4, 389–408

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

VISIBLEZ: A MAINFRAME ARCHITECTURE EMULATOR FOR COMPUTING EDUCATION

David Woolbright, Vladimir Zanev, Neal Rogers

Abstract. This paper describes a PC-based mainframe computer emulator called VisibleZ and its use in teaching mainframe Computer Organization and Assembly Programming classes. VisibleZ models IBM's z/Architecture and allows direct interpretation of mainframe assembly language object code in a graphical user interface environment that was developed in Java. The VisibleZ emulator acts as an interactive visualization tool to simulate enterprise computer architecture. The provided architectural components include main storage, CPU, registers, Program Status Word (PSW), and I/O Channels. Particular attention is given to providing visual clues to the user by color-coding screen components, machine instruction execution, and animation of the machine architecture components. Students interact with VisibleZ by executing machine instructions in a step-by-step mode, simultaneously observing the contents of memory, registers, and changes in the PSW during the fetch-decode-execute machine instruction cycle. The object-oriented design and implementation of VisibleZ allows students to develop their own instruction semantics by coding Java for existing specific z/Architecture machine instructions or design and implement new machine

ACM Computing Classification System (1998): C.0, K.3.2.

Key words: Mainframe architecture emulator, visualization, computer organization, assembly language, computing education.

instructions. The use of VisibleZ in lectures, labs, and assignments is described in the paper and supported by a website that hosts an extensive collection of related materials. VisibleZ has been proven a useful tool in mainframe Assembly Language Programming and Computer Organization classes. Using VisibleZ, students develop a better understanding of mainframe concepts, components, and how the mainframe computer works.

1. Introduction. Columbus State University offers mainframe-related courses for students in the Enterprise track of our BS in Computer Science. Students can select classes in mainframe Computer Organization, JCL, Assembly Language, Cobol, DB2, CICS, and IMS. For many years, hundreds of enterprises have based their business information technologies on IBM mainframe platforms, and since the introduction of the IBM 360 in 1964, millions of lines of code in Assembler and Cobol have been written by a generation of mainframe programmers. This code is still valuable and drives the transaction systems of many of the largest enterprises around the world, particularly in banking and insurance. As older mainframe programmers have retired, a growing demand for new mainframe programmers and software developers has appeared. There is a particular demand for students with enterprise programming skills – programmers who can help maintain, and extend this legacy software. The IBM mainframe architecture has not only endured, but is growing in popularity as a platform of choice for new information technologies in the areas of virtualization, visualization, Web applications, and cloud computing.

The mainframe courses we offer are attractive to students because they provide students unique skills and employment opportunities in a growing market. As a result, we have a stable enrolment in these courses each semester. We teach mainframe-related courses in a traditional way - with lectures, textbooks, slides, assignments, projects, labs, quizzes, and exams. Under the IBM Academic Initiative we have access to a virtual machine hosted on an IBM System/z located in Dallas, Texas. On the client side, our lab computers use IBM's Personal Communications software to connect and work with mainframe applications in Assembler, Cobol, DB2, CICS, and IMS. Students have twenty-four hour access to this machine both locally and off-campus.

Teaching mainframe courses with full access to a real mainframe is critical for developing student understanding of the operating system, and promotes the necessary skills for working as a programmer and developer. In delivering these courses, particularly in teaching assembly language and computer organization, we have found that students are often challenged by the complexity of

the machine. Learning any assembly language is a complex process that requires a broad knowledge of a machine's architecture and instruction set. Topics that are addressed in these two courses include main memory content (data representation and machine instructions), general purpose and floating point registers, the Program Status Word (PSW), machine instruction address schema, the fetch-decode-execute cycle, I/O channel programs, supervisor calls, reading memory dumps and debugging.

In seeking ways to improve how we teach mainframe assembly language, architecture and concepts, we considered the advantages of using modeling and simulation tools with visualization and animation capabilities to help improve student outcomes and performance. One important consideration was the fact that our students come into these courses with strong backgrounds in Java (CS1, CS2 and Data Structures), and we wanted to leverage their Java skills as a way of approaching mainframe assembly language and machine organization.

2. Mainframe Simulation and Modeling for Assembly Language and Computer Organization. The growing diversity and complexity of enterprise computer system hardware presents certain challenges and problems for students in Computer Organization and Assembly classes. Many instructors have turned to computer modeling and simulation tools as teaching aids for these classes. Some good Computer Organization and Assembly textbooks [15] come with embedded or additionally developed computer simulators [22], [23]. Computer simulators are very useful from a pedagogical point of view because students can become familiar with different computer components, their properties, functions, and relationships. Using a simulator, students can refine their computer system knowledge while developing a deeper cognitive understanding of an entire system. Simulation with visualization and animation promotes comprehension of system objects, phenomena, and processes in a way that is difficult to replicate with traditional teaching techniques.

We surveyed the literature, including currently published textbooks, as well as the Internet for existing computer simulators used for teaching a class in Computer Organization or Assembly language programming.

We had two goals: 1) To find mainframe computer simulators we can use in our Computer Organization and Assembly courses, and 2) To explore functionality, visualization and the interfaces of the simulators.

Two back-to-back issues of the Journal of Educational Resources in Computing (JERIC) are devoted to general computer architecture simulators (vol. 1, No. 4, December 2001) as well as specialized computer architecture simulators (vol. 2, No. 1, March 2002), and their use in computing education. More

than a dozen simulators are presented in these two issues of JERIC. Of these, we attentively reviewed those simulators which are complete computer architecture software environments with rich functionality and tools. CPU Sim [19] is a complete development environment written in Java with an Assembler and assembly language editor, windows for registers and RAM, and debugging features. CPU Sim allows simulations of a variety of architectures – accumulator-based, RISC, or stack-based CPUs. EasyCPU [26] simulates a simplified model of Intel 80x86 processors. It works in two-modes – basic and advanced. It has good visualization tools for the main computer units, an assembly editor, step-by-step or continuous execution of programs with different speeds, visualization and access to registers, data, and stack segments. The PDP-8 emulator [18] is a historic machine emulator that simulates the PDP-8 architecture on an Intel 80x86 computer. It is an environment with a text editor, assembly translator, virtual engine upon which to execute machine code, and a debugging tool that allows visualization of registers, memory, and I/O interface. A survey paper of Wolffe and Yurcik [25] lists seven categories of early computer simulators, most of them obsolete now. Black and Komala considered several computer simulators and listed their main properties and functions in their paper [2]. The Bochs [3] emulator, written in C++, allows the simulation of a variety of Intel x86 CPUs, common I/O devices, and a BIOS on different platforms. Emu8086 [7] is an IDE with code editor, assembler, disassembler, microprocessor emulator, and debugger, but it is limited to 16-bit code and does not support I/O. SPIM [22] is a MIPS simulator designed to run assembly code, and has good tracking capabilities, separate frames for data visualization, and a built-in Assembler for this architecture. The MarieSim simulator [14], written in Java, accompanies The Essentials of Computer Organization and Architecture text, and is based on the MARIE architecture. It consists of an editor, assembler, loader, and microprocessor with GUI interface, organized as an IDE with visualization of a step execution mode, symbol table, data segment, memory, registers, output, and data path animation. MARS [23] is an IDE developed in Java that was designed to be used with the Computer Organization and Design text of Patterson and Hennessy [15]. It has a convenient GUI with assembly code, registers, memory, and data segment visualizations. LARC [2] architecture includes functional simulator of MIPS similar instructions, machine and assembly language debugger.

All of the above mentioned simulators are appropriate for use in teaching Computer Organization and Assembly classes, however they do not satisfy our first goal of being simulation tools we could use for IBM mainframe-based classes in Computer Organization and Assembly programming.

We did find and examine several IBM mainframe emulators including FLEX-ES, zPDT, Hercules [7], and z390 [8], [20], [21]. FLEX-ES was developed by Fundamentals Software, Inc., and is an emulator package that allows an IBM 31-bit machine to run on an x86 Intel machine. FLEX_ES still exists, but was discontinued because IBM declined to renew licenses for the new IBM 64-bit z/Architecture. For legal licensing issues, Hercules [8], is also not considered by some to be completely legal software. zPDT is an IBM released emulation product that can provide a virtual System/z architecture environment supporting selected mainframe operating systems, middleware and software running on x86 processor-compatible platform. It runs well and offers the full z/Architecture. The z390 [24] is a Java-implemented set of portable tools with a GUI and a command line interface, a macro assembler (HLASM), a linker/binder, a z/Architecture instruction emulator, memory dumps, and trace facilities. The O'Kane's text, Basic IBM Mainframe Assembly Language Programming [12] uses the z390 as a replacement for a real mainframe, but we could not find any execution traces or reviews of the use of z390 in an educational setting. The most significant problem with the mainframe emulators mentioned above is that they are commercial software systems whose main purpose is to replace an existing IBM mainframe machine. They were not created for educational purposes and cannot easily be used to convey the main system objects, relationships, and functionality of an IBM mainframe to a student. While their supporting documentation is usually Web-based, it is also cumbersome, and voluminous.

After surveying existing simulators, we decided to implement our own emulator for the IBM mainframe z/Architecture – VisibleZ.

3. Analysis and Design of the VisibleZ Emulator.

3.1. Learning Goals. In designing the VisibleZ emulator, there were a number of learning goals. All of these goals were conceived in the context of teaching assembler and computer organization for a specific family of machines – IBM's System/z. Our learning goals seek to develop an effective and deeper student understanding of:

- The mainframe machine components, including the CPU structure, and machine instruction execution with the fetch/decode/execute cycle
- The flow of control of the execution of a program with sequencing of machine instructions, comparisons, and branching
- The impact that architectural decisions make on instruction semantics.
- The use of base/displacement addressing as a technique for locating data in memory.

- Data representations and instruction formats
- The role of registers in arithmetic and address creation
- The instruction semantics of a fundamental set of assembler instructions.

Beside these learning goals, there was one overriding goal that motivated us: To find a more effective method of teaching all of these concepts by building a product that we could use in a classroom or with individual students. The VisibleZ emulator that we built presents each student with an environment that simulates and visualizes System/z architecture and its functioning.

3.2. Design Principles. After analysing the computer simulators above, and examining our own specific needs, we established the following design requirement principles for the VisibleZ mainframe emulator. It must:

- Promote the teaching and learning of System/z concepts. Our aim is not to replicate a machine, but to use a highly visual representation of the machine as a teaching and learning tool. This requirement feature is unique to VisibleZ in that other System/z emulators are designed to act as machine replacements, specifically for professional programmers.
- Have a GUI front end that is projectable in a classroom.
- Model a basic von Neumann architecture, specifically, the IBM System/z.
- Run on PCs with existing operating systems.
- Include, as a minimum, the following basic mainframe architectural components: main storage, CPU (with ALU, registers, and PSW), and an I/O channel.
- Emulate a subset of mainframe instructions and be extensible in the sense that students should be able to invent new instructions for the architecture as part of a Computer Organization class.
- Allow students to build the executable code for an instruction by implementing the instruction semantics in terms of Java objects.
- Provide for visualization of the execution (emulation) of machine instructions and other main emulator components – main storage, registers, PSW, CPU, and ALU
- Be able to interpret object code directly. The goal is to take the executable code generated by an assembler and have it directly executed in the emulator. Students should also be able to build object code programs by hand and have them directly executed.

- Be intuitively easy to use.

Our choice of Java as the implementation language was very natural – Java is portable, and object-oriented, and would allow us to develop all the architectural components as Java objects, providing the system with a rich functionality. More importantly, our students learn Java as their first language, and we wanted to have them study instruction semantics by writing Java code that becomes part of the emulator.

Teaching mainframe introductory Computer Organization and Assembly Programming classes using a real z/Architecture machine is not without problems. While learning the main components of the z/Architecture and how they work together is itself a worthy endeavour, the complexity and the abstraction layers of the z/Architecture machine can be overwhelming for undergraduate students. IBM z/Architecture machines are complex: multiple processing units (PU) – up to 64 in six different functional categories for z9 ES; there is a "trimodal" way of addressing with 24, 31, and 64-bit addresses; four different sets of CPU registers: general purpose, floating-point, access, and control registers; a Program Status Word (PSW) with 128 bits in several formats; a virtual storage system that maps virtual addresses onto real addresses; a complex system of six classes of interrupts and timing facilities; and a Channel I/O Subsystem which is in fact a separate specialized computer. The CPU now has over a thousand machine instructions in twenty six categories. VisibleZ addresses the complexity of a modern mainframe by providing only those components of a real machine that are directly needed by a programmer. This allows the student to focus on essential mainframe concepts, components, and organization, and avoids the overwhelming details of the mainframe z/Architecture.

The design of VizibleZ is based on some general and multimedia learning and cognition principles [1], [4], [16]. We are applying the spatial contiguity principle by designing VisibleZ with a main window that includes separate panels for each component. Students assimilate mainframe concepts easily when machine components are juxtaposed and exposed rather than appearing on separate panels or screens. Following the temporal contiguity principle, we designed VisibleZ with components presented simultaneously rather than successively. We have tried to mitigate the overwhelming details of the z/Architecture by hiding extraneous details following Mayer's coherent principle of effective learning [11]. According to the context principle [4], student learning depends on the context in which concepts are presented. In VisibleZ, machine instructions are presented in the context of an explicit representation of memory formats, registers, an animated fetch-decode-execute cycle, and a coloring of instructions and components that

provides visual semantic clues. All of these features help students develop a better understanding of the mainframe architecture, and promotes effective learning of mainframe concepts and components.

3.3. VisibleZ Architecture. The VisibleZ architecture is shown in Figure 1, and includes the main components of a von Neumann machine (with mainframe terminology and components): main storage, central processor unit (CPU) with control unit, program status word (PSW), arithmetic logical unit (ALU), registers, and channel I/O subsystem.

Main storage is viewed as a sequential collection of addressable bytes. It contains the program as a sequence of machine instructions (in object code) and all program data. The size of the main storage is configurable, but in most cases, size N=1K or 2K bytes is sufficient for educational programs and data.

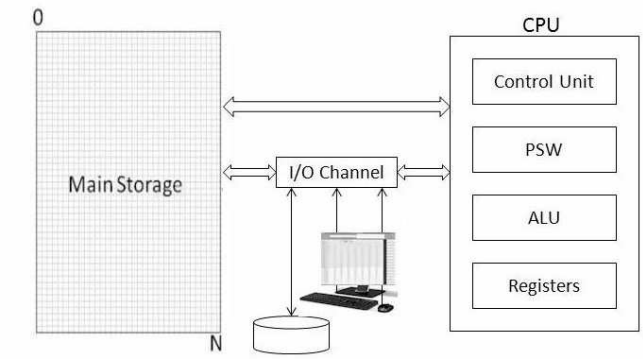


Fig. 1. VizibleZ Architecture

3.4. VisibleZ Functionality. The VisibleZ Central Processing Unit (CPU) is the controlling system of emulator and consists of a Control Unit, Program Status Word (PSW), Arithmetic Logical Unit (ALU) and Registers: general-purpose registers (GPR) and floating-point registers (FPR) are provided. The CPU controls the data flow from the main storage to and from the ALU and the other CPU components. It fetches machine instructions from the main storage, decodes them and sends the data in proper sequence to the ALU for processing. The timing sequences of the Control Unit for machine instruction execution are initiated by the user on a step-by-step basis. The PSW, on a real machine, is a 128 bit special control register that contains information required for the execution of the currently active program. The current state of the CPU is displayed in a variety of PSW fields. The VisibleZ PSW includes the following

components of an actual PSW: instruction address (bits 64-127), Condition Code (CC) (bits 18-19), and addressing mode (bits 31-32). Instructions may designate information in one or more of 16 general purpose and 16 floating-point registers. All registers are 64 bits. The GPRs are used as base-address registers, index registers and as accumulators in some arithmetic and logical instructions. The FPRs are used in floating-point instructions and contain floating-point operands. The VisibleZ ALU is equipped to execute machine instructions in RR, RX, RS, RI, SI, SS1, and SS2 formats. The structure of the VisibleZ ALU allows for easy extension of the set of machine instructions with new instructions from the z/Architecture or by user design. The following data formats are used with machine instructions: binary (signed and unsigned formats), decimal (zoned and packed formats), floating-point (IEEE format), and alphanumeric [9], [10].

Input/Output (I/O) operations involve transfer of data between main storage and an I/O device. I/O devices are attached to the I/O Channel which models a simplified version of the z/Architecture channel subsystem by providing for up to three input files with specific Data Definition (DD) names: FILEIN1, FILEIN2, or FILEIN3. Three output files are also supported with Data Definition (DD) names: FILEOUT1, FILEOUT2, and FILEOUT3. Record sizes for input and output records are fixed at 80 bytes. Physical file names are specified using full pathnames on the local machine. The I/O channel supports a limited collection of I/O macros including OPEN, CLOSE, GET, and PUT. Within these constraints, it is possible to assemble an object module program that reads and writes to files, and have it execute in VisibleZ.

3.5. VisibleZ Visualization. The VisibleZ emulator includes a set of graphical user interfaces for each of the main architectural components with the ability to visualize the state, content and functionality of each component. The set of GUIs run in a single window with separate panels for each component. The following GUI panels were considered at the design level:

- A Memory panel displaying addresses and memory content (machine instructions and data) during program execution, with coloring that emphasizes the beginning locations of source and target fields.
- A PSW panel displaying the instruction address of the machine instruction about to be executed, the condition code, and the addressing mode.
- A Register panel displaying the content of all registers with coloring that emphasizes source, target, and index registers.
- A Decoding panel where the main parts of each instruction are exhibited (object and explicit code formats).

- A Control panel with buttons that allow program loading and reloading, machine instruction execution (cycling), and PSW resetting.
- Info panels describing the main components of VisibleZ.

4. VisibleZ Implementation, Interface and Operation.

4.1. VisibleZ Implementation. VisibleZ is implemented as a collection of Java classes in a project. Objects representing all the major architectural components that are important to an assembly programmer are organized into three main components:

- Architecture
- Instruction Set
- Interface

You can see a VisibleZ UML class diagram in Figure 2 with the structure of these three VisibleZ main components. The Instruction set is modeled with the Instruction class hierarchy. Currently there are six instruction format classes (SS1, SS2, SI, RX, RS, and RR) that provide basic functionality. Each instruction is equipped with an execute() method that is invoked during the fetch-decode-execute cycle and represents the semantics of the instruction within the system.

By examining (or coding) the execute() method of an instruction, students become familiar with the details of the architecture and the effects of each instruction on the state of the entire system. The system is designed for extensibility so that new instructions and instruction formats can be added to the system easily.

The Architecture component consists of Java classes that implement the functionality of the VisibleZ architecture presented in Figure 1 - CPU with PSW, GPRegisters (general-purpose registers), FPRegisters (floating-point registers), and two data format classes (Hex, PackedDecimal) with utility methods.

The Interface component contains a set of panels together with buttons, textboxes, text areas, and component listeners accepting input from the keyboard and mouse, transferring the input action to the Architecture components, and displaying information about the memory content, registers contents, PSW instruction address and Code Condition, and fetch-load-execute phases. The Interface component is represented as a single Enterprise panel which is composed of all other panels and GUI components. The Enterprise panel is the main tool that users interact with when working with the emulator. For teaching purposes, objects from the Interface components – PSW panel, general registers panel, floating-point registers panel, and the Enterprise panel have implemented listeners that

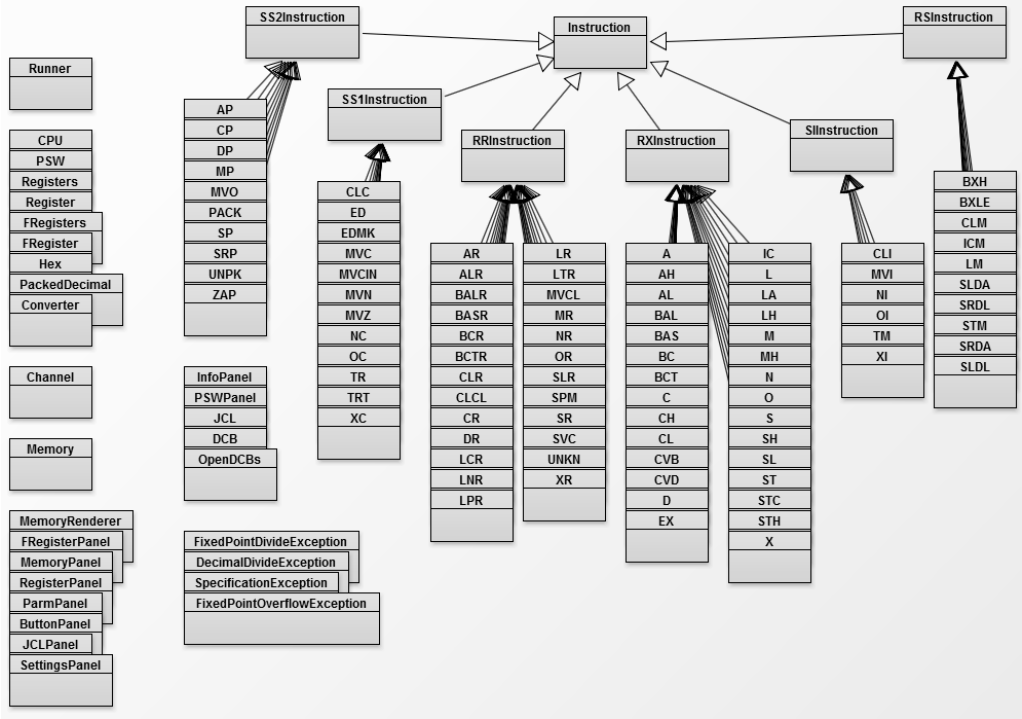


Fig. 2. VisibleZ UML Class Diagram

display information panels with explanations about the corresponding object.

4.2. VisibleZ Interface and Operation. The main user component of the VisibleZ emulator is the virtual machine environment (see Figure 3) with which the students interact. It is organized as a window containing a set of panels implementing the main components of the mainframe architecture and their functionality. VisibleZ includes a set of interfaces, one for each of our architecture components. All components of the mainframe model – main storage, registers, PSW, together with additional panels – JCL panel, fetch-decode panel, info panels, and control buttons are visible on the main Enterprise panel.

Main storage is represented as a panel with an addressable space of bytes in hexadecimal format. The size of the main storage is configurable. In Figure 3 it consists of 1K of bytes. The main storage area contains the executable program in machine format (the upper portion of the memory), the program data area (in the middle), and a rudimentary operating system area (at the bottom). The operating system is presented with a save area for registers, a parameter area,

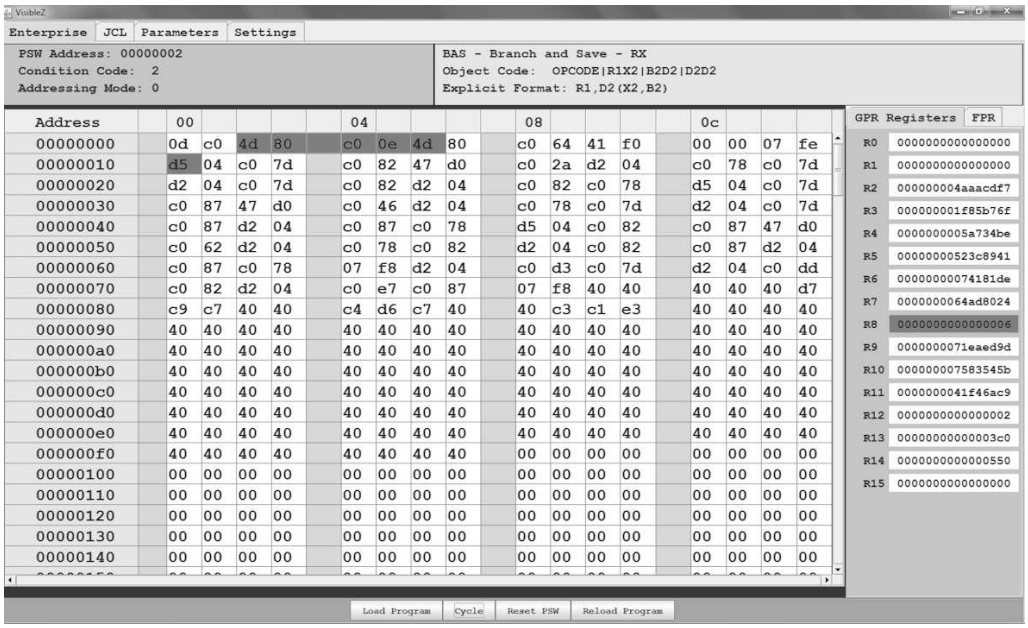


Fig. 3. VisibleZ Environment

and some simple channel I/O instructions with Supervisor Calls (SVCs).

Clicking on the Cycle button causes execution of the current instruction and the next instruction is fetched and decoded. The instruction which is about to be executed is colored and highlighted, the instruction address in the PSW is updated to point at the beginning of the instruction, and the fetch-decode-execute panel displays the instruction code of operation, object code, format, and operands. Target fields are highlighted in red, source fields are highlighted in green, and the current instruction in yellow. Index registers, when present, are colored grey. Execution of the current instruction may also cause the Condition Code to be updated on the PSW panel. Users see immediately the effects of each instruction execution. By highlighting each instruction before it executes in the memory, coloring the source and target fields, describing the current instruction in the fetch/decode area, and showing the results of executing an instruction on the memory, registers, and the PSW, VisibleZ draws a visually helpful and animated picture of the state of the machine.

The Load Program button allows the user to load machine programs in the main storage panel. VisibleZ is distributed with over a hundred pre-written object code programs that are ready for execution and illustrate basic assembler

principles and machine and instruction specifics. The Reset PSW button sets the instruction address of the PSW to the beginning of the program and allows the user to start execution over at the beginning. Programs can also be easily reloaded with the Reload Program button. Object code programs have a simple text file format: each byte of object code is represented as two adjacent hexadecimal digits and is separated from other bytes by whitespace (usually a single space). Small assembly programs can be prepared for execution on VisibleZ by hand with any text editor. Figure 4 depicts the process of machine program development for a sample mainframe Assembly program [5]. For large assembler programs, the machine (object) code file that is loaded into VisibleZ is prepared by first assembling the program on a mainframe. The object module produced by the HLASM assembler is used as input for a mainframe utility program that converts the machine code to the VisibleZ format. This utility program is also available for distribution with VisibleZ.

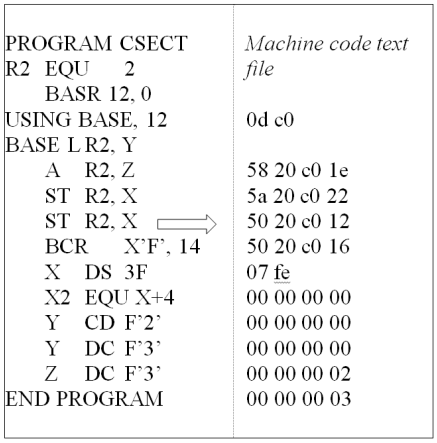


Fig. 4. Machine Program Development

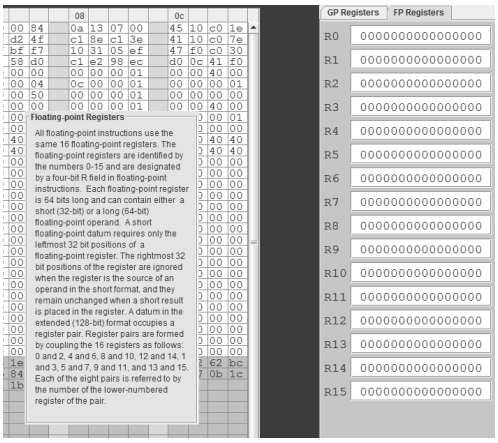
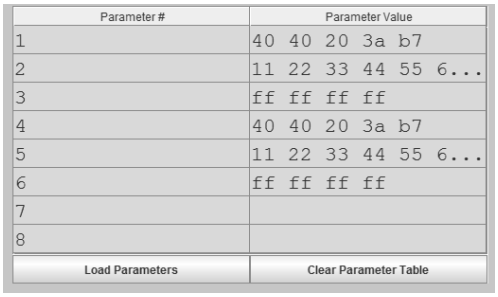


Fig. 5. Info Panel for FP Registers

Clicking on the main Interface components of the Enterprise panel opens information panels with explanations about each component. In this way, students are introduced to mainframe components with explanations, properties and functions of the component. Figure 5 shows an information panel that describes floating-point registers.

Clicking on the JCL tab opens the JCL Panel and clicking on the on the Parameters tab opens the Parameters Panel.

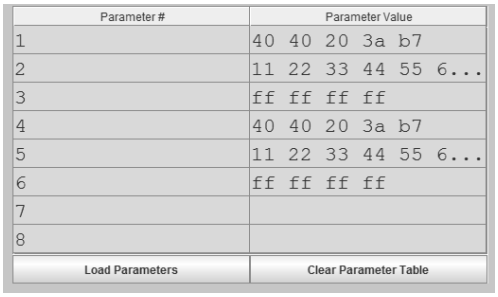
VisibleZ object programs can read and write up to six sequential (QSAM-type) input and output files. DD names are limited to six specific choices. The



Parameter #	Parameter Value
1	40 40 20 3a b7
2	11 22 33 44 55 6...
3	ff ff ff ff
4	40 40 20 3a b7
5	11 22 33 44 55 6...
6	ff ff ff ff
7	
8	

Load Parameters | Clear Parameter Table

Fig. 6. JCL Panel



Parameter #	Parameter Value
1	40 40 20 3a b7
2	11 22 33 44 55 6...
3	ff ff ff ff
4	40 40 20 3a b7
5	11 22 33 44 55 6...
6	ff ff ff ff
7	
8	

Load Parameters | Clear Parameter Table

Fig. 7. Parameters Panel

JCL panel can be used to establish the connection between the DD names used inside an assembly program and the external physical file names. EBCDIC data is automatically converted to ASCII data during input and output, so that text-oriented data is easy to read on the host machine. VisibleZ supports OPEN and CLOSE processing, PUT and GET macro code, and a limited set of supervisor calls (SVC).

Support is also provided for testing subprograms that pass parameters. The Parameters panel is used for entering up to eight values which can be passed to a subprogram. Clicking the "Load Parameters" button stores the values in the operating system memory area, builds a table of addresses for the parameters, and initializes register 1 with the address of this table. After loading the parameters, a subprogram is then free to process the parameters during program execution using standard linkage techniques.

5. Teaching Mainframe Computer Organization and Assembly Language with VisibleZ. We are using VisibleZ in lectures, labs, and assignments. VisibleZ is distributed in two formats, depending on how an instructor intends to use it:

1) Executable Version

Java .class files are provided as an executable .jar file with full functionality for all instructions. Source code for instructions is withheld. In this format, students can create and load object programs, step through execution of each instruction, and use the system as learning and debugging tool.

2) Extensible Version

Extensible source code is provided for the basic system as a Java BlueJ project. Source code is also provided for all instructions, but only a working set of instructions are fully functional with completely coded execute() methods. In

this format, an instructor can assign students the task of researching an instruction's semantics, and building the Java code that reflects the semantics using the emulator classes. Using the working set of completed instructions, students can still step through execution of many instructions, and use the system as learning and debugging tool. Note: Source code for the majority of instructions is not provided as a means of preserving the integrity of student assignments which involve building this code.

VisibleZ has proven to be a valuable classroom demonstration tool for covering the following topics:

- Introduction to System/z architecture
- Main storage organization
- Registers and register operations: general purpose, floating-point, control, and access registers
- Data types
- The CPU and ALU
- Program Status Word fields and format
- Base/Displacement Addressing
- The fetch-decode-execute cycle
- Machine state and control
- Timing and interruptions
- The instruction set and instruction formats
- Assembly coding and translation of programs to machine code
- Interruptions and operating system support
- I/O processing

VisibleZ is supported by a web site that has links to articles, lessons, and videos covering assembler concepts. The support site, `http://csc.columbusstate.edu/woolbright/vzHomepage.htm`, is divided into three parts:

1) Concept Lessons – These lessons cover those ideas that are fundamental to learning assembly language and machine architecture.

2) Instruction Lessons – Instruction semantics for each of the supported instructions is described, numerous examples of each instruction are provided, and tips for using each instruction are given. Additionally, VisibleZ is distributed with a collection of object code programs that exercise every supported instruction.

For each instruction, there are two or more object code programs that use the instruction and allow a student to quickly see each instruction in action.

3) Building Instruction Lessons – A unique feature of VisibleZ is that it provides an environment in which students can be asked to research the semantics of an instruction, and to write the Java code that insures the instruction behaves as described. Each machine instruction has an execute( ) method which contains the required code. In fact, because of the object-oriented design, it is easy to add new instructions and instruction formats to the system, providing students an opportunity to experiment with the instruction architecture.

Several labs were developed in order to build students' knowledge and skills, and to help prepare the students for completing the assignments. These labs are briefly summarized as follows:

- As new instructions are introduced, students execute the object code programs that are provided for each instruction. By observing the effects that instruction execution has on the entire system, students become familiar with the machine architecture, base/displacement addressing, fetch-execute cycle, and the semantics of the specific instructions.
- Students learn the details of data representations like zoned decimal, packed decimal, and two's complement by seeing data displayed in memory, and by examining the helper methods provided in the utility classes that come with the system.
- After a sufficient number of instructions have been covered, the class begins writing some simple mainframe assembly programs, and assembling them as object modules. After converting the object code to a VisibleZ format and loading the machine code into VisibleZ, students can single step through each instruction to verify that each program works correctly, or debug the code as needed.
- A mainframe program which reads and writes sequential files is introduced. The object code is loaded into VisibleZ. Execution of the program introduces I/O macros including OPEN, CLOSE, GET, PUT, and DCBs. This program also provides a chance to introduce supervisor calls and interrupt processing.
- Standard linkage techniques are introduced. Students use VisibleZ to build parameters and a parameter address table in memory. Students observe how a subprogram uses the data structure during execution to process the parameters.

As a minimum, four assignments with VisibleZ are scheduled. The first

assignment covers base/displacement addressing and involves modifying the object code for a small program in such a way that the base/displacement addresses require modification. The second assignment investigates the role of index registers when computing an effective address. In the third assignment, each student has to develop the Java code for the execute() method of several machine instructions. In this assignment, students also have to build object code programs that demonstrate that each instruction is working correctly. The final assignment requires that a team of two or three students design, develop, and test some of the more complicated instructions like Translate (TR), Translate and Test (TRT), Pack (PACK), Unpack (UNPK), Branch On Condition (BC) and Edit (ED).

For a course in computer organization, students could be asked to design and code an entirely new instruction (or even a small instruction set) for the system. This process could involve creating a new instruction format or using an existing one. In fact, a whole new architectural component (like a stack), could easily be added because of the system's extensibility.

For the past four years, the authors have taught assembly language and computer organization courses using VisibleZ at Columbus State University and Marist College, New York. It has been used in conducting corporate training classes in assembly language programming. In each case, we have found that VisibleZ is a helpful tool for teaching and learning assembly language concepts. Informal surveys of these classes indicate that students found that VisibleZ is an effecting teaching and learning tool. So far we have taught a total of twelve classes with about two hundred students. VisibleZ is free software. It has been downloaded over one hundred times. The link for downloading VisisbleZ is at the end of the paper.

6. Conclusions and Future Work. The main conclusions we draw based on a long experience of teaching assembly language, is that VisibleZ is an effective visualization tool for teaching mainframe architecture and Assembly language. We believe the VisibleZ emulator offers a new approach to learning mainframe concepts and architecture, and exploits the object-oriented skills of our students. This approach causes students to think deeply about instruction semantics in a way that our previously traditional approach did not.

VisibleZ project is an on-going work. The object-oriented design and implementation allow development of new extensions and enhancements of VisibleZ features, functionality, and interfaces. Future work in the short term includes: extending the instruction set with floating-point instructions (decimal, hexadecimal, and binary), adding additional control instructions, and developing a new visu-

alization and animation of the fetch-decode cycle. Our long term plans include implementing a better I/O Channel subsystem and a simple mainframe Supervisor. Finally, if time and resources permit, we would like to develop a Web-based VisibleZ emulator.

The VisibleZ emulator homepage is located at the URL below and contains links for downloading the product, a video demo, and a collection of lessons.

`http://csc.columbusstate.edu/woolbright/visiblez.xml`

REFERENCES

[1] Ambrose S. A. et al. How learning works. Seven research based principles for smart teaching. John Wiley & Sons, ISBN 9780470484104, 2010.

[2] Black M., P. Komala. A full system x86 simulator for teaching computer organization. In: Proceedings of the SIGCSE'11, Dallas, Texas, 2011, 365–370.

[3] Bochs IA-32 emulator. `http://bochs.sourceforge.net/`, January 23rd, 2015

[4] Domjan M., J. Gray. The principles of learning and behavior. Cengage Learning. 6th edition, ISBN 9780495601999.

[5] Corliss M., R. Hendry LARC: A little architecture for the classroom. *Journal of Computing Sciences in Colleges*, **24** (2009), No 6, 15–20.

[6] Carrano F. Assembler language programming for the IBM 370. The Benjamin/Cummings Publ. Co. Inc., 1988.

[7] EMU8086Emulator.`https://archive.org/details/tucows\_325007_Emu8086_-_Microprocessor_Emulator`, January 23rd, 2015.

[8] Hercules System/370, ESA/390, and z/Architecture emulator. `http://www.hercules-390.org`, January 23rd, 2015

[9] IBM z/Architecture principles of operation. 9th Edition, SA22-7832-08, 2010.

[10] IBM Redbooks. *ABCs of z/OS system programming*, **10** (2008).

[11] Mayer R. Multimedia learning. Cambridge University Press, 2001, ISBN 0521787491.

[12] O'Kane K. Basic IBM mainframe assembly language programming. CreateSpace Paramount, CA, 2011.

[13] J. Kincaid, K. Westerlund. Simulation in education and training. In: Proceedings of 2009 winter simulation conference, Austin, Texas, December, 2009, 273–280.

[14] Null L., J. Lobur. MarieSim: The Marie computer simulators. *ACM journal of educational resources in computing*, **3** (2003), No 2.

[15] Patterson D., J. Hennessy. Computer organization and design. 4th Ed, Morgan Kaufmann Publ., 2008.

[16] N. Rutten, W. Van Joolingen, J. Van Der Veen. The learning effect of computer simulations in science education. *Computers & Education*, **58** (2012), No 1, 136–153.

[17] Schools teaching mainframe subjects. `http://www.mainframes.com/schools.htm`, January 23rd, 2015.

[18] Shelburne B. A PDP-8 emulator program. *ACM journal of educational resources in computing*, **2** (2002), No 14, 17–47.

[19] Skrien D. CPU Sim 3.1: a tool for simulating computer architectures in computer organization classes. *ACM journal of educational resources in computing*, **1** (2001), No. 4, 46–59.

[20] Smith P. III, The state of IBM mainframe emulation. *zJournal*, Apr/May2007, 62–64. `http://enterprisesystemsmedia.com/magazines/enterprise-tech-journal`, January 23rd, 2015.

[21] Smith P. III, IBM mainframe emulation: reloaded. *zJournal*, December 2008/January 2009, 36–38. `http://enterprisesystemsmedia.com/magazines/enterprise-tech-journal`, January 23rd, 2015.

[22] SPIM: A MIPS32 simulator. `http://spimsimulator.sourceforge.net/`, January 23rd, 2015.

[23] Vollmar K. , P. Sanderson. MARS: An education-oriented MIPS assembly language simulator. In: Proceedings of SIGCSE'06, March 1-5, 2006, Houston, Texas, 239–243.

[24] z390 portable mainframe assembler and emulator project, 2008. http://www.z390.org/, January 23rd, 2015.

[25] Wolffe G., W. Yurcik, H. Osborne, M. Holliday. Teaching computer organization/architecture with limited resources using simulators. In: Proceedingsof the SIGCSE'02, February 27-March 3, 2002, Covington, Kentucky, USA, 176–180.

[26] Yehezkel C., W. Yurcik et al. Three simulator tools for teaching computer architecture: EasyCPU, Little man computer, and RTLSim. *ACM journal of educational resources in computing*, **1** (2001), No 4, 1–15.

David Woolbright
Columbus State University
TSYS School of Computer Science
4225 University Ave,
Columbus, GA 31907
e-mail: woolbright_david@columbusstate.edu

Vladimir Zanev
Columbus State University
TSYS School of Computer Science
4225 University Ave,
Columbus, GA 31907
e-mail: zanev_vladimir@columbusstate.edu

Neal Rogers Columbus State University
TSYS School of Computer Science
4225 University Ave,
Columbus, GA 31907
e-mail: rogers_neal@columbusstate.edu

Received November 11, 2014
Final Accepted January 23, 2015