This article was downloaded by: [Florida International University]
On: 11 January 2015, At: 14:46
Publisher: Routledge
Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: Mortimer House, 37-41 Mortimer Street, London W1T 3JH, UK

Computer Science Education
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ncse20

The contribution of visualization to learning computer architecture

Cecile Yehezkel [a] , Mordechai Ben-Ari [a] & Tommy Dreyfus [b]
[a] Weizmann Institute of Science , Israel
[b] Tel Aviv University , Israel
Published online: 11 Jun 2007.

To cite this article: Cecile Yehezkel , Mordechai Ben-Ari & Tommy Dreyfus (2007) The contribution of visualization to learning computer architecture, Computer Science Education, 17:2, 117-127

To link to this article: http://dx.doi.org/10.1080/08993400601165545

PLEASE SCROLL DOWN FOR ARTICLE

Taylor & Francis makes every effort to ensure the accuracy of all the information (the “Content”) contained in the publications on our platform. However, Taylor & Francis, our agents, and our licensors make no representations or warranties whatsoever as to the accuracy, completeness, or suitability for any purpose of the Content. Any opinions and views expressed in this publication are the opinions and views of the authors, and are not the views of or endorsed by Taylor & Francis. The accuracy of the Content should not be relied upon and should be independently verified with primary sources of information. Taylor and Francis shall not be liable for any losses, actions, claims, proceedings, demands, costs, expenses, damages, and other liabilities whatsoever or howsoever caused arising directly or indirectly in connection with, in relation to or arising out of the use of the Content.

This article may be used for research, teaching, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden. Terms & Conditions of access and use can be found at http://www.tandfonline.com/page/terms-and-conditions

المنارة للاستشارات

www.

Computer Science Education
Vol. 17, No. 2, June 2007, pp. 117–127

Routledge
Taylor & Francis Group

The Contribution of Visualization to Learning Computer Architecture

Cecile Yehezkel[a]*, Mordechai Ben-Ari[a] and Tommy Dreyfus[b]
[a]*Weizmann Institute of Science, Israel;* [b]*Tel Aviv University, Israel*

Downloaded by [Florida International University] at 14:46 11 January 2015

This paper describes a visualization environment and associated learning activities designed to improve learning of computer architecture. The environment, EasyCPU, displays a model of the components of a computer and the dynamic processes involved in program execution. We present the results of a research program that analysed the contribution of the visualization to learning. We found that EasyCPU facilitated the use of improved study methods and enabled the construction of a viable mental model of the computer.

1. Introduction

This paper describes a visualization environment and associated learning activities designed to improve learning of computer architecture and a research programme aimed at assessing the contribution of the environment to improving learning. The importance of learning computer architecture and the difficulties encountered by teachers and students have been well documented (Clements, 2000; Loui, 1988). Kumar and Cassel (2002) found that many faculty members who teach this subject are teaching outside their areas of specialization and are not entirely comfortable with the task. To improve the learning of computer architecture, instructors have searched for better pedagogical methods. It was also our motivation to develop a visualization environment and its associated learning activities.

The environment, EasyCPU, displays a model of the components of a computer and the dynamic processes and information flows involved in program execution at the architecture level (Yehezkel, Eliahu, & Ronen, 2001). EasyCPU also includes tools needed to write assembly language programs: an editor, a simulated compiler and linker, and debugging tools.

In section 2 we discuss existing software tools for learning computer architecture, a taxonomy of visualizations, and related research on the effectiveness of visualization

*Corresponding author. Department of Science Teaching, Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: ntcecile@wisemail.weizmann.ac.il

ISSN 0899-3408 (print)/ISSN 1744-5175 (online)/07/020117-11
© 2007 Taylor & Francis
DOI: 10.1080/08993400601165545

المنارة للاستشارات
www.

systems. The EasyCPU environment and the learning activities in the course are described in section 3. The environment and the learning activities were the subject of a research project that blended quantitative and qualitative methodologies, as described in section 4. Section 5 summarizes the paper.

2. Background

Professional tools such as simulators and debuggers used in the development of assembly language programs are too sophisticated and complex for introductory level students. There are many teaching tools for computer architecture (Cassel et al., 2001; Yurcik, 2002; Yurcik & Osborne, 2001). While these tools share many features, since they were designed by individual instructors they tend to be targeted to specific populations, illustrating the level of abstraction required by the specific curriculum. The lack of an appropriate framework to define the characteristics of visualization environments motivated the construction of a new taxonomy for program visualizations (Yehezkel, 2002), based on a very broad taxonomy of software visualization (Price, Baecker, & Small, 1998), but emphasizing didactic and cognitive aspects. This taxonomy guided the development of EasyCPU.

Evaluation of the effectiveness of visualizations is an important topic of research in computer science education. Hundhausen, Douglas, and Stasko (2002) conducted a meta-study on algorithm visualization evaluation research and concluded that how students use technology has a greater impact on its effectiveness than what technology is used. They suggested that ethnographic field techniques and observational studies can help to understand both how and why technology might be effective in a realistic situation. We used quantitative methods to assess the effectiveness of the environment, while qualitative methods were used to improve our understanding of both how and why visualization contributes to learning.

3. The EasyCPU Environment and the Learning Activities

The EasyCPU environment was designed as a learning tool for a high school course in introductory computer organization and assembly language programming. The textbook (Zilberman, 1999) covers the theoretical material that is taught in classrooms; this is supplemented by laboratory sessions. EasyCPU has been used since 1998 by more than 7000 students.

EasyCPU is based on a simplified model of an 8 bit version of the Intel 80X86 microprocessor family. The model consists of the CPU, memory segments, input/output components, and the bus connections between them. The model of the CPU includes the general registers, instruction and stack pointer registers, flags, and a clock. The memory is partitioned into three segments, data, stack, and code, each with 256 addressed bytes. Data can be entered directly into the CPU registers and memory. The I/O consists of eight simulated LEDs for the output and eight simulated buttons for the input. The control, data, and address busses are represented by lines in different colors. EasyCPU simulates a subset of the instructions of the

Downloaded by [Florida International University] at 14:46 11 January 2015

للاستشارات المنارة www.

Downloaded by [Florida International University] at 14:46 11 January 2015

Intel X86 which was selected to represent the various instruction categories, addressing modes, and data types. EasyCPU operates in two modes, basic and advanced, to enable a gradual increase in the complexity of the tasks assigned.

3.1. Activities in Basic Mode

In basic mode the control, address, and data busses connecting the different units are animated to illustrate the read/write cycle type (memory or I/O). For example, in a memory read cycle arrows slide on the address bus from the CPU to the memory, the control line MemR lights up, and then arrows slide on the data bus from the data segment to the CPU. The instructional goals addressed by the basic mode activities are:

- to learn the structure and classification of the instruction set and to identify their mnemonics;
- to learn the syntax of the assembly language and to understand addressing modes;
- to understand the mechanism of instruction execution and of memory and I/O read/write cycles.

3.2. Activities in Advanced Mode

The advanced mode is designed for students who have attained a basic knowledge of assembly language instructions, enabling them to develop programs. In effect, the advanced mode functions as an integrated development environment (Figure 1).

The environment visualizes the processes taking place within the computer by simultaneously displaying the source code, the data and stack segments, and an on-screen simulation of I/O ports. After writing code in the program editor, assembly and linking are simulated, followed by a simulation of execution of the program. Students can step through a program, observing the state of the computer after each step. The instructional goals addressed by the advanced mode activities are:

- to understand the structure of a program;
- to understand the process of executing a program;
- to acquire basic skills in the use of move, arithmetic, logic, and control instructions;
- to become familiar with the stack data structure and the actions executed on it;
- to learn to build structured programs with subroutines;
- to introduce the interrupts and understand their implementation.

4. Assessing the Visualization Environment

The widespread use of EasyCPU provided an opportunity to conduct an evaluation of its contribution to the development of the students' programming skills; the four phases of this research are described in the following subsections.

المنارة للاستشارات

www.

Downloaded by [Florida International University] at 14:46 11 January 2015

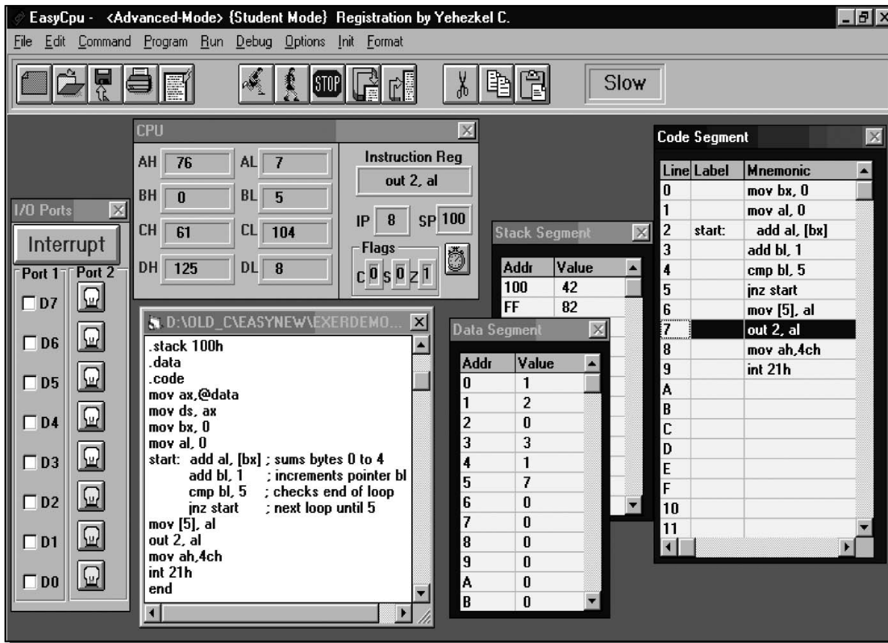

Figure 1. Developing a program in advanced mode

4.1. Does Visualization Help Learn Programming?

This evaluation study was conducted to assess the effectiveness of EasyCPU in the development of programming skills. We compared students' performance when writing programs using either EasyCPU or the professional tool Turbo Assembler (TASM). The experimental group was composed of students from classes who studied the course using EasyCPU, while the control group was composed of students from another class who used TASM in the same course. All students were in Grade 11 and the classes were taught by the same teachers who had taught these students computer science fundamentals the previous year. The teachers all had experience teaching this course on computer organization and assembly language.

A pre-test administered at the beginning of the academic year determined that students of the two groups were of equivalent ability. The evaluation was done using as a post-test the grade on a question of the matriculation examination, where students were asked to write and test a short program in a computer laboratory. Statistical analysis (ANOVA) was performed on the results, as shown in Table 1.

The experimental group scored slightly lower than the control group in the pre-test, although this was not statistically significant, while the experimental group significantly outperformed the control group in the post-test.

The results demonstrated that the use of a visualization environment can improve the performance of students when writing as well as when testing a program. The results motivated us to continue the investigation in order to better understand the role played by visualization.

المنارة للاستشارات
www.

Table 1. Student achievement

Group	Control[a] ($n = 26$, 1 class)	Experimental[a] ($n = 99$, 3 classes)	F
Pre-test	73.9 ± 11.2	68.9 ± 14.4	$F_{3,112} = 1.67$
Post-test	82.2 ± 23.4	92.8 ± 13.15	$F_{3,121} = 3.34$[b]

[a]Means ± SD.
[b]Statistically significant at the $p < .05$ level.

Downloaded by [Florida International University] at 14:46 11 January 2015

4.2. Visualization Environments and Activity Styles

The second study was intended to investigate the influence of the visualization environment on the students as they engaged in a development activity. The subjects were 12 Grade 11 high school students learning computer architecture with the EasyCPU visualization environment. To adapt the complexity of the task to the students' programming skills, we specified a simple control unit for an elevator. The hardware of the control unit was represented by eight buttons and eight light bulbs. Six bulbs specified the floor where the elevator was, while the other two were used to denote whether the elevator was to move up or down.

Following Levin and Mioduser (1996), we adopted the terms behavioural model (B model) and conditional model (C model). The B model is a conceptual model of a system that allows the student to describe verbally the system components and different situations entered as a result of the system – user interaction, while the C model is the control system implemented by programming a control unit. The students were expected to define the B model of the elevator and to describe the components of the system and the various states the elevator could be in as a result of interaction between the user and the system. Then they were requested to implement the C model of the control system according to the B model they had defined.

The activity was divided into two parts. In Part I the students were asked to carry out the entire task on paper: to define a B model and to write a program to implement the C model. In Part II the students were asked to use the visualization environment to test their program; they had to submit a revised version after completing the testing. This set-up enabled us to investigate the methods used by students to convert a B model into a C model without feedback provided by the environment and then to investigate the contribution of the environment in testing and correcting the program.

Two students, J and S, were asked to perform the activity in collaboration and were videotaped while working on the assignment. The remaining students were observed while performing the activity individually. The analysis was undertaken in two phases: an in-depth analysis of the videotape and the field notes, followed by a differential analysis of the two versions of the program from each student. (For lack of space the differential analysis is not included here.)

The analysis was carried out at two levels: at the micro level we identified *foci of conversation* (the main subject discussed in each utterance) from the transcript of the videotape, while at the macro level we used these foci of conversation to identify the

المنارة للاستشارات

www.

foci of operations (the central activity being carried out in the problem-solving process). For example, the main conversation foci were the elevator, the user of the elevator, the program, and a more holistic reference to the embedded system, the hardware. In the transcript the foci of conversation changed as a function of time, reflecting changes as the students passed from one objective in the task to another. Examples of foci of operations are a description of the B model, implementation of the C model, and its verification relative to the B model. The goal of this analysis was to identify the different phases in the implementation of the embedded system in terms of foci of operations and then to identify when students make use of visualization components. Table 2 shows examples of this analysis, where items 1 – 3 pertain to Part I and items 4 – 5 to Part II. (The examples are not sequential; they were selected for the purposes of presentation.)

From this analysis we created a diagram depicting the foci of conversation as a function of time. The diagram was analysed visually, concentrating on timing, changes, sequences, and frequency of the foci of conversation. This enabled us to identify the foci of operations that reflected the phases in implementing the control

Table 2. Samples of foci of conversation and operation

	Student utterance	Student points to	Foci of conversation	Foci of operation
1	S: Wait a minute. . . . He [the user] requests [the elevator]. How can I know what he wants when he requested it?		User and elevator	Discussion on B model
2	J: Do you understand? You have to control the elevator [by programming].	Lights that represent the floors	Elevator and program	Understanding the meaning of programming the control unit
3	S: Please tell me, how can you stop the program inside the program?		Program	Implementation of the C model
4	J: We will use the compare CMP instruction. Suppose that we are at floor 0. Here it is no longer possible to go down. So, do you understand? This is not relevant to it [the system], so this part of the program is unnecessary. He [the user] can't request a negative floor!	Light that represents the 0th floor	Program and B model	Implementation of the C model relating to the B model
5	J: Look, the elevator is going down. That's curious!		Elevator	Confrontation of the C model and the B model

Downloaded by [Florida International University] at 14:46 11 January 2015

المنارة للاستشارات

www.

system. Next, from an examination of the sequences of foci of operations and their change as a function of time, we were able to characterize episodes of task performance. It is impossible to render here the full 100 minute diagram; Figure 2 shows a 13 minute section illustrating foci of conversation and operation. (The full diagram can be downloaded from http://stwww.weizmann.ac.il/g-cs/benari/articles/figure.pdf.)

Although in Part I students were not supposed to use the visualization, we noticed that they pointed to components displayed on the screen that were relevant to implementation of the operation panel of the elevator—this was done to explain their ideas one to another (e.g. item 2 of Table 2). The visualization components provided them with a common visual vocabulary that was essential for collaboration. Initially the students behaved as naive users, thinking that the elevator worked by itself—J explained to S that she had to drive the elevator by programming (item 2 of Table 2) and then emphasized: "What do you mean, it [elevator] can stop—you have to make it stop!" They gradually learned to discern between the control unit and the operational unit and to understand their role as programmers of the control unit (item 3 of Table 2). When implementing a C model appropriate to the B model they discussed issues using a formal language containing expressions belonging to the B model (item 4 of Table 2). The students were able to provide a detailed description of the B model, but they failed to implement the required C model and to properly check the correctness of the program they wrote.

In Part II they used the visualization environment to confront the C model with the B model. In Figure 2 we can see the transition between the diagnosis and correction of the C model by reference to the B model and the systematic checking of the C model. The former is characterized by recurrent references to the program; the latter by few references to the program, more iterative references to the comparison of the

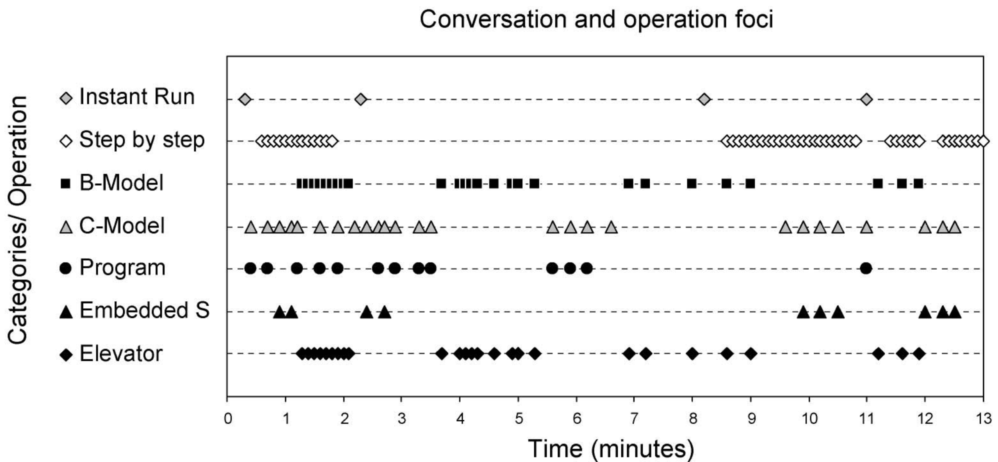

Figure 2. Conversation and operation foci during a 13 minute period of the activity

Downloaded by [Florida International University] at 14:46 11 January 2015

المنارة للاستشارات

www.

C model and the B model, and intensive use of the step by step function of the environment.

A glance at the 100 minute diagram shows the great diversity in conversation foci in Part II, as opposed to the lack of diversity in Part I. In Part I they made recurrent use of holistic references like: "He skips [the floor] now and he will check it when he passes [the next floor]," where we attribute "he" to the entire embedded system. In Part I they made 56 such references, compared with 36 in Part II. In contrast, they referred to the elevator and its user 18 times in Part I, compared with 36 times in Part II. We conclude that in Part I they did not distinguish between the different components of the embedded system, but in Part II the concretization of the embedded system components by the visualization enabled them to distinguish between the components and to realize that their program had to play the role of "director" of the embedded system.

4.3. How is Visualization Used to Test a Program?

We analysed the methods students used to test assembly language programs in the visual EasyCPU environment and the professional TASM environment. Here we summarize this study (a comprehensive description was published in Yehezkel, 2003).

We tried to identify behavioural patterns that characterized the methods students used and to relate these methods to the environments. The subjects were Grade 10 high school students who studied the course in computer organization and assembly language. One group of 18 students used EasyCPU, while a second group of 12 students used TASM. The students were asked to detect two types of errors that were intentionally inserted into a program. One (the I error) required in-depth analysis of the execution of the program, while the other (the E error) was caused by an incorrect boundary condition that could be easily detected by examining the output as a function of the input. Two pairs of students were videotaped working on the assignment, one using EasyCPU and the other using TASM; observations of the remaining students were carried out.

The work of each pair was represented by a plot of the time the students devoted to basic operations: data input, instant run, step by step, edit, and mental run (see figure 1 in Yehezkel, 2003). This enabled us to conduct a detailed graphical analysis of the characteristics of the basic operations, such as timing, frequency, and duration, to identify patterns and to compare the ways of performing the activity.

The students using TASM employed primarily the data input and instant run operations. We concluded that they adopted a strategy of trial and error as they struggled to retrieve the relevant information from the screen and that they related to the program as a black box. In contrast, the students using EasyCPU tended to run the program step by step. They seemed to feel more confident in investigating the program's execution using the feedback provided by the environment and they showed that they knew how to exploit the potential of the environment. Further

Downloaded by [Florida International University] at 14:46 11 January 2015

www.

qualitative analysis of the transcripts and the field notes of the observations of the remaining students confirmed these findings.

Only 58% of TASM students, as compared with 94% of EasyCPU students, succeeded in identifying the I error, which required an in-depth analysis of execution of the program. On the other hand, there was no significant difference in identification of the E error, which required only an analysis of the input and output (94% of the EasyCPU students and 83% of the TASM students). These results strengthened the findings of the qualitative study.

We concluded that the visualization of the EasyCPU environment facilitated the students' investigations of the detailed behaviour of the program, whereas learning without visualization may be detrimental to comprehension of program execution by inexperienced students. Our next step was to evaluate the contribution of visualization to understanding the mechanism of instruction execution, which is essential to understanding models of computers.

Downloaded by [Florida International University] at 14:46 11 January 2015

4.4. Improving Mental Models of Computers

In this study we wished to evaluate the contribution of the EasyCPU visualization environment to the understanding of a conceptual model of a computer. This was done by investigating the mental models that students constructed. This research is summarized here (a comprehensive description was published in Yehezkel, Ben-Ari, & Dreyfus, 2005).

The mental models of the students were investigated at two points in time: first, after studying the textbook but before exposure to EasyCPU and, second, after performing exercises with the system. The research tools were two tasks, a pre-test and a post-test, in which the students were asked to describe both the static viewpoint (the topology of the interconnections between the units CPU, input, output, and memory) and the dynamic viewpoint (six scenarios describing the data transfer resulting from executing specific instructions) of the computer. The research was carried out in a class of 11 Grade 10 high school students learning with EasyCPU. They were asked to describe the system model both graphically and in written text alongside the graphs.

On the pre-test five of the 11 students chose to draw systems consistent with a data flow model in which data flows from the input to the CPU to the output, two more drew memory-centric models, and only four drew the correct CPU-centric model. On the post-test, after the students had interacted with the conceptual model presented by the visualization, all the students drew models consistent with the correct one.

These findings support the view that the visualization was critical in enabling the construction of a viable mental model, a process that did not occur by textbook learning alone. After studying theoretical materials the majority of the students still held mental models that had been influenced by their experience as computer end-users, since the data flow model is consistent with how users interact with computers. Initially the students were not exposed to any dynamic visualization of the process of instruction execution, only to frontal teaching and the textbook. Once the students

المنارة للاستشارات
www.

were exposed to the dynamic visualization of EasyCPU they developed viable mental models.

5. Conclusions

Our research has shown that students who used the EasyCPU visual environment performed better than those who used the TASM professional environment. We found that the concretization of the embedded system helped students in conceptualizing the system and in developing programming skills. We were able to demonstrate that the students used different methods when working in the two environments: students using TASM tended to relate to the program as a black box, while EasyCPU facilitated studying the execution of the program at the level of individual instructions. We discovered that the students developed non-viable mental models during the textbook phase of learning, but that after using EasyCPU students' models pointed to an understanding of the roles of the various computer units and the interactions between them.

Acknowledgements

We would like to thank the teachers and the students who participated in this research and Bruria Haberman for valuable comments.

References

Cassel, L., Kumar, D., Bolding, K., Davies, J., Holliday, M., Impagliazzo, J., et al. (2001). Distributed expertise for teaching computer organization and architecture (ITiCSE 2000 Working Group Report). *ACM SIGCSE Bulletin*, *33*(2), 111 – 126.

Clements, A. (Ed.) (2000). Computer Architecture Education [Special issue]. *IEEE Micro*, 20(30).

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, *13*(3), 259 – 290.

Kumar, D., & Cassel, L. (2002). A state of the course report: Computer organization and architecture. *SIGCSE Bulletin*, *34*(3), 175 – 177.

Levin, I., & Mioduser, D. (1996). A multiple-constructs framework for teaching control concepts. *IEEE Transactions of Education*, *39*(4), 488 – 496.

Loui, M. C. (1988). The case for assembly language programming. *IEEE Transactions on Education*, *31*(3), 160 – 164.

Price, B. A., Baecker, R. M., & Small, I. (1998). An introduction to software visualization. In J. Stasko, J. Domingue, M. Brown, & B. Price (Eds.), *Software visualization* (pp. 3 – 34). Cambridge, MA: MIT Press.

Yehezkel, C. (2002). A taxonomy of computer architecture visualizations. *SIGCSE Bulletin*, *34*(3), 101 – 105.

Yehezkel, C. (2003). Making program execution comprehensible—one level above the machine language. *SIGCSE Bulletin*, *35*(3), 124 – 128.

Yehezkel, C., Ben-Ari, M., & Dreyfus, T. (2005). Computer architecture and mental models. *SIGCSE Bulletin*, *37*(1), 101 – 105.

Downloaded by [Florida International University] at 14:46 11 January 2015

المنارة للاستشارات

www.

Yehezkel, C., Eliahu, M., & Ronen, M. (2001). Teaching computer architecture with a computer-aided learning environment. *SIGCSE Bulletin*, *33*(1), 445.

Yurcik, W. (Ed.) (2000). Specialized computer architecture simulators that see the present and may hold the future [Special issue]. *Journal on Educational Resources in Computing*, *2*(1).

Yurcik, W., & Osborne, H. (2001). A crowd of little man computers: Visual computer simulator teaching tools. *Proceedings of the 33rd Conference on Winter Simulation* (pp. 1632–1639). Los Alamitos, CA: IEEE Computer Society Press.

Zilberman, H. (1999). *Computer organization and assembly language.* Tel Aviv: Open University.

Downloaded by [Florida International University] at 14:46 11 January 2015

المنارة للاستشارات

www.