An Integrated Approach to Effective Computer Architecture Learning by Simulation

Besim Mustafa

Abstract — A highly interactive, integrated and multi-level system simulator has been developed specifically to support both the teachers and the learners of computer architecture teaching modules at degree level. The simulator provides a highly visual and user configurable environment with numerous pedagogical features aimed at facilitating deeper understanding of concepts which are often difficult to visualize and grasp by the students. The paper explains the rational behind the development and gives a description of the main features of the simulator. It presents some typical examples of the ways in which the simulator has been used to support the lectures and the practical tutorial sessions. A brief account of the research to assess and evaluate the simulator as an effective educational tool and the initial results are also presented.

Index Terms — Computer architecture and organization, compiler, operating systems, simulation, visualization.

1 Introduction

The teaching of computer organization and architecture which include key subject areas such as production of executable code using compilers and assemblers, instruction set architecture (ISA) design, performance enhancing technologies and principles of multi-tasking and memory management using operating systems form the core set of subjects for most computing and computer science undergraduate degree programmes. The ACM/IEEE computing curricula [1] and the QAA's subject benchmark in computing [2] reports both acknowledge and recommend these topic areas as the essential part of a good computer science education. In most cases, the teaching of these topics involves combination of methods such as traditional lectures, programming assignments, modification of educational operating systems by the students and simulations. Exactly which combination is used tends to depend on the educational institution delivering them and can often be influenced by the teaching expertise and the resources available within the computing departments.

At author's university the delivery of the three-year full-time modular computing degree programme is the responsibility of the Business School. In the first year, the majority of students study a module on the fundamentals of computer architecture. Some of these students elect to study more advanced topics in the second year. The delivery of the programme includes theory by lectures supported by practical tutorial sessions as well as coursework assignments. The students studying for the computing degrees are recruited from wide educational backgrounds (in "widening participation" context [3]) and competencies which may not include any previous computing experience or qualifications.

The tutorial and practical sessions on computer architecture have been supported mainly by paper-based exercises, on-line quizzes and by working with different operating systems such as Windows and Linux. However, there remained a requirement for studying those "hidden" architectural features which are difficult or impossible to access and demonstrate on real systems. With this in mind, it was decided that a software-based simulator would be developed with pedagogical aspects designed to support and enhance the teaching modules in computer architecture and operating systems.

2 On the Motivation

The motivation for the development of a new educational simulator is due to two main reasons: 1) the specific requirements of our teaching modules (described below) and 2) the lack of availability of an existing suitable simulator (see next section). However, an important requirement stands out and that is for a simulator that can support modules both

▪ *B. Mustafa is with the Business School, Edge Hill University, Ormskirk, U.K. E-mail: mustafab@ edgehill.ac.uk.*

in computer architecture and operating systems at different stages of students' educational development from basic to advanced levels. The following is a summary of the specific requirements.

2.1 An Integrated Simulator

In computer architecture, different technologies are interrelated and support each other across clearly defined interfaces. So, for example, a compiler for a high-level language generates a valid set of low-level instructions which can be executed by a particular CPU architecture; an operating system can then facilitate multi-programming by sharing the CPU and memory resources between different sets of instructions representing different programs. It is these interdependencies and interfaces that the highly integrated simulator realistically and accurately represents. In other words it reveals the "big picture" instead of bits of it in isolation from each other. Most students prefer to see the connections between different aspects of computer technology and understand how they "hang" together.

2.2 Rich Pedagogical Features

As an educational resource, the simulator enhances and enriches the learning experiences of students and facilitates deep understanding of the key concepts. It also actively encourages experimentation, exploration and investigative problem solving where students work either individually or in groups, in a lab or at home. It does this by promoting hands-on approach, enabling realistic simulations and providing immediate visual feedback. The visualization features include graphical animations, colour-coded representation of system functions, real-time plots of measurements, dynamic lists of events and trees of attribute structures.

2.3 Control and Monitoring Facilities

In a dynamic simulation environment it is essential to be able to suspend or stop the simulations on occurrence of some pre-determined point, event or stage. It must also be possible to re-start the simulator from suspension. It may be necessary to manually make changes to selected components (e.g. memory locations, register values, instructions, etc.) of systems being simulated prior to resuming the simulation. This enables students to experiment with the impact of such modifications at some key stages of system simulations.

2.4 Support for Educational Levels

The simulator should support students at different stages of their educational development from basic to advanced levels of competencies and should incorporate simulations of a wide range of technological aspects of computer architectures.

2.5 Integrated Visual Displays

Students have different learning styles. This requirement addresses the needs of those who prefer visual learning style. At the same time, real-time, colour coded, visual displays with "audible" state changes can provide more immediate feedback and impact which can highlight trends, clearly reveal state transitions and demonstrate relative comparisons.

2.6 Support for Problem-Based-Learning

Problem-based learning (PBL) is a method of encouraging independent student-lead learning. The simulator can support and facilitate PBL with ease. For example, given a description of a problem, the students can devise appropriate test procedures to demonstrate the problem and offer solutions.

2.7 Full Support for Computing Curricula

The computing curricula defined in ACM/IEEE and QAA reports cover various topics on computer architecture, low-level programming and operating systems. The simulator implements features which functionally cover all these areas.

2.8 No Programming Knowledge Required

One of the main features of the simulator is to facilitate the learning of computer architecture and operating systems technology without requiring prior programming experience. However, the simulator incorporates a "teaching" compiler which can compile a relatively simple high-level "teaching" language which is very easy to learn and compile.

2.9 Support for Advanced Features

The simulator is designed to support advanced architectural features such as multiple processors, instruction level pipelining, compiler optimizations, virtualization and distributed processing. These features are used to demonstrate advancements in system performances and support research projects.

2.10 Intuitive User Interface

The users of the simulator should not be faced with a steep learning curve as the simulator is designed to be used with one-semester modules as well as with the longer modules. However, as the same simulator can be used across different modules and over several years of study, the students soon become familiar with its usage. The simulator's functions are clearly separated into independent and distinct sections of user interfaces thus effectively hiding the total complexity; users concentrate only on certain sections of interest at a time, slowly building competency in mastering the many functions of the simulator over time as and when needed.

3 Prior Work

Over the years, many simulators of computer architectures have been developed which have been used as valuable educational resources at undergraduate computing courses [4], [5]. These range from simple, abstract, high-level simulators to advanced simulators of commercial CPUs.

TABLE 1
SURVEY OF SIMULATORS

Name	Simulator		Comments
	OS	CPU	
SchedulerSim [6]	Yes	No	CPU scheduling concept
Simulator + assembler [7]	No	Yes	IO processing + interrupt handling
MKit [8]	No	Yes	Inst. set + data paths + control unit
SOsim [9]	Yes	No	Process + memory management
Simulator [10]	Yes	No	Process scheduling, HTML-based, scripted
Simulator [11]	No	Yes	Assembler + inst. set
MarieSim [12]	No	Yes	Assembler + inst. set + data paths
PDP-8 simulator [13]	No	Yes	PDP-8 inst. set + assembler
Simulta [14]	No	Yes	Inst. set + microcode + input/output
CPU Sim [15]	No	Yes	Inst. set + microcode + assembler
MARS [16]	No	Yes	MIPS assembly language simulator
Starving philosophers [17]	Yes	No	Limited OS: Synchronization + monitor
Address translation [18]	Yes	No	Limited OS: Virtual memory
MPS [19]	No	Yes	Inst. set + input/output + MIPS CPU simulator
JASP toolkit [20]	No	Yes	Inst. set + assembler + high-level lang.
PsimJ simulator [21]	Yes	No	Various isolated OS component simulations

Table 1 lists some examples of software simulators developed for educational purposes. The simulators are categorized as operating system (OS) based and CPU based simulations. Most of the simulators developed appear to be CPU based. The OS simulators tend to be rather fragmented along the lines of distinct but isolated functionality. It is interesting to note that none of the listed simulators incorporate both CPU and OS simulations in one software package.

There have been surveys of many other software simulators and visualization tools designed to support computer architecture education at universities and colleges [22], each using slightly different approach to satisfy local educational requirements. Some simulators have been developed to accompany text books on operating systems and computer architectures [20], [21], [23], [24]. These simulators often concentrate on some specific technological aspects of the systems and do not offer a unified approach.

Another related approach taken by various universities and colleges is to develop or use existing teaching operating systems that the students are asked to modify and/or extend [25], [26]. This approach often requires good programming ability by the students and, although highly realistic, is not always suitable as a teaching and learning resource

4 Simulator Design Details

The design and development of the system simulator are based on clearly defined requirements. The integrated simulator is composed of three main components: a "teaching" compiler, a CPU simulator and an operating system (OS) simulator supporting each other. For example, the compiler will generate code which can be run by the CPU simulator either in isolation or under the control of the OS simulator for multiprogramming support. Each of the three components is described below.

4.1 Support for Pedagogy

The simulator is designed to provide extensive support for pedagogy as an educational teaching and learning tool. This includes the following features:

1. Feedback is available on all simulation progress and results.
2. Pace of simulations is adjustable.
3. Breakpoints allow stop/start at critical points for observations and recording.
4. Breakpoint values can be manually modified for experimentation.
5. Colour-coded visual effects are used to enhance observations.
6. Activities are logged and saved for later recall and analysis.
7. High configurability affords flexibility of control and observations.

8. Simulated technology features can be enabled and disabled to demonstrate with and without type comparisons.
9. Active participation, critical thinking and analysis are facilitated.
10. All simulation parameters are transparent and open to observations.

The ways in which the above pedagogy is applied are self evident in the design principles and the usage examples given.

4.2 The Compiler

A basic but complete high-level teaching language is developed to support the CPU and OS simulations. This language incorporates the standard language control structures, constructs and system calls which are used to demonstrate a modern computer system's key architectural features. A compiler is developed for this language which generates both assembly-level language and its equivalent binary byte-code as output. The compiler is also able to disassemble the binary byte-code back to its assembler code equivalent thus demonstrating reverse-engineering concept desirable in certain circumstances. Fig. 1 shows a snapshot of the main compiler user interface.

The compiler includes refinements such as code optimizations, support for profiling, display of compiler stages and the binary code generated as well as some statistical data. Additionally, the compiler includes an integrated tabbed source editor capable of handling multiple source code at the same time. The "teaching" compiler can support a module on compiler design.

The compiler optimizations can be used to demonstrate performance gains due to reductions in code size and enhancing CPU pipelining (see below) when jump instructions are eliminated. They are also used to demonstrate that an experienced human assembly coder is still a better producer of more efficient code than most optimizing compilers.

The compiler and its associated language naturally support the CPU and the OS simulations thus reflecting the importance of the language processors.

4.3 The CPU Simulator

The CPU is loosely based on RISC architecture with a prominent register file composed of from 8 to 64 configurable fast registers, a minimal set of variable-length instructions and a limited number of addressing modes. Except two instructions, viz. load and store, the instruction set is based on register to register addressing. Optionally the CPU instructions can be entered manually by selecting the valid instructions and any operand(s) from a list of instructions and operands. In selecting operands the associated addressing modes can also be specified at the same time. The selected assembler instruction is then added to the CPU instruction memory. The stored instructions can then be individually selected and manually executed. The simulator provides runtime debugging facilities for the selected instructions, registers and memory locations. A stack is provided which demonstrates support for interrupts, system calls, subroutine parameters and return addresses.

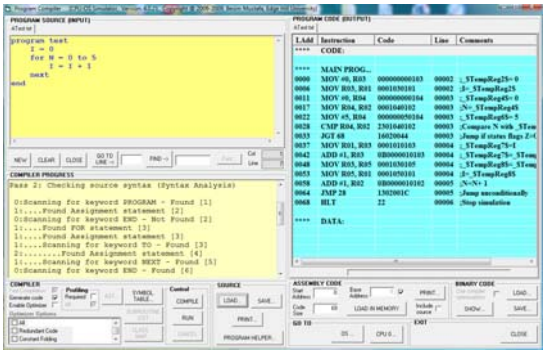

Fig.1. The Compiler Window

A further refinement to CPU simulator is the inclusion of cache and pipeline simulations both of which provide highly configurable and visual operations. These advanced simulators can be used to demonstrate technology specific details and their impact on system performance. The cache placement and replacement policies can be selected; the hit/miss ratios can be plotted and compared. The pipeline stages are colour coded and animated. Different methods of eliminating pipeline hazards can be clearly demonstrated to improve performance. A history of pipeline activity is maintained which can be used to investigate pipelining. Fig. 2 shows the main user interface for the CPU simulator.

In order to be able to study systems with multiple processors, the simulator can optionally start multiple processor simulations. Each processor is identical and loading code in one is duplicated in others thus simulating shared memory, tightly-coupled architecture. The processors can be used to demonstrate load balancing and virtualization with multiple operating systems.

The CPU simulator defines a list of vectored interrupts. Each interrupt vector is triggered by a pre-defined event, e.g. console input or timer event. The inbuilt high-level language has constructs for the definition of interrupt routines as interrupt handlers the addresses of which are placed in the interrupt vectors at program load time.

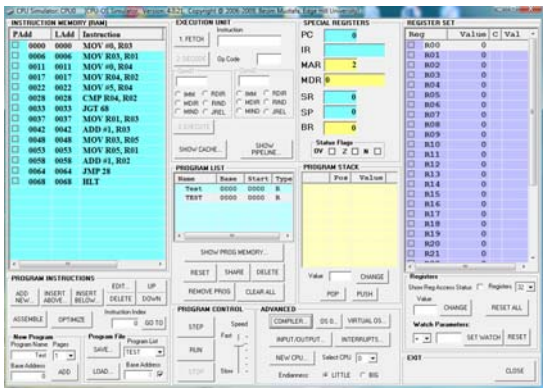

Fig. 2. The CPU Simulator Window

4.4 The OS Simulator

The OS simulator is designed to support two main aspects of a computer system's resource management: process management and memory management. Fig. 3 shows the main user interface for this simulator. Once a compiled code is loaded in CPU memory, its image is also available to the OS simulator. It is then possible to create multiple instances of the program images as separate processes. The OS simulator displays the running processes, the ready processes and the waiting processes. Each process is assigned a separate process control block (PCB) which contains information on process state. This information is displayed in a separate window. The memory display demonstrates the dynamic nature of page allocations according to the currently selected placement policy. The OS maintains a separate page table for each process which can also be observed. The simulator demonstrates how data memory is relocated and the page tables are modified as the pages are moved in and out of the main memory illustrating virtual memory activity.

The process scheduler includes various selectable scheduling policies which include priority-based, pre-emptive and round-robin scheduling with variable time quanta. The OS is able to carry out context-switching which can be visually enhanced by slowing down or suspending the progress at some key stage to enable the students to study the states of CPU registers, stack, cache, pipeline and the PCB contents.

The simulator incorporates an input output console device, incorporating a virtual keyboard, and is used to display text and accept input data.

The OS simulator supports dynamic library simulation which is supported by the appropriate language constructs in the teaching language. The benefits of sharing code between multiple processes are visually demonstrated. There is also a facility to link static libraries demonstrating the differences between the two types of libraries and their benefits and drawbacks.

The simulator allows manual allocation and de-allocation of resources to processes. This facility is used to create and demonstrate deadlocks associated with resources and enables experimentation with deadlock prevention, detection and resolution.

Threads are fundamental aspects of modern multiprogramming/multi-tasking systems. This feature is supported by the OS simulator via special language constructs which identify parts of programs for execution as threads. The threads are scheduled like processes but they share their parent's memory address spaces. The concepts of orphan and zombie processes are also explored.

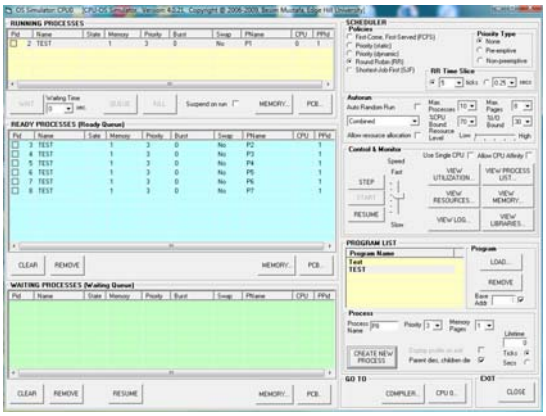

Fig. 3. The OS Simulator Window

In multiprogramming systems it is sometimes necessary and desirable to prevent multiple processes accessing shared resources at the same time. So the concepts of synchronization and critical regions are facilitated by special teaching language constructs. There is also a Java style subroutine synchronization facility.

The CPU utilization can vary depending on the types of applications. The concept of

CPU-bound and IO-bound applications is explored by the OS simulator by artificially varying the ratio of running to waiting processes. The simulator monitors the CPU and memory utilizations and displays the information in a graphical format at runtime.

5 VIRTUAL MACHINE

A separate stand-alone virtual machine (VM) has been developed which is able to interpret and execute the simulator's compiled byte-code. The VM is a native executable code, implemented in C language, and is currently available on Windows and Linux platforms. This is a console based facility and runs under the control of the host operating system and supports multi-threading. This code demonstrates the concept of VM by enabling the execution of the same code on different platforms in a manner similar to Java Virtual Machine (JVM).

6 TEACHING AND LEARNING STRATEGY

At Edge Hill University the simulator has been successfully integrated into teaching modules on computer architecture and operating systems and has been in use for the past two years. Each one-hour lecture is supported by a two-hour practical tutorial session. The students work in groups of two or three. The simulator software is provided on a removable disk drive and runs under Windows operating system. The groups follow instructions on the exercise sheets and are expected to provide responses to various questions at different stages of simulations. The questions are designed to promote critical thinking and deeper understanding of the concepts under investigation. The work completed during these practical sessions form the student's assessed tutorial portfolio.

7 HIGHLIGHTS OF USAGE

In this section we look at four different sample scenarios which demonstrate the ways in which the simulator has been used to provide educational support. Each scenario represents successively more advanced feature and is supportive of the next one.

7.1 Programming the CPU

In the introductory module on computer architecture, the students learn about the CPU instruction set and low level programming by entering the instructions directly into the CPU's memory, executing them individually and observing the results. The simulator provides a list of valid instructions but the students identify the correct instructions for a given task, e.g. comparing two integer values and jumping to a location if equal or forming a simple loop that counts or writing to a location in memory using direct and indirect addressing modes. Fig.4 shows the instruction list window students use to select the CPU instructions.

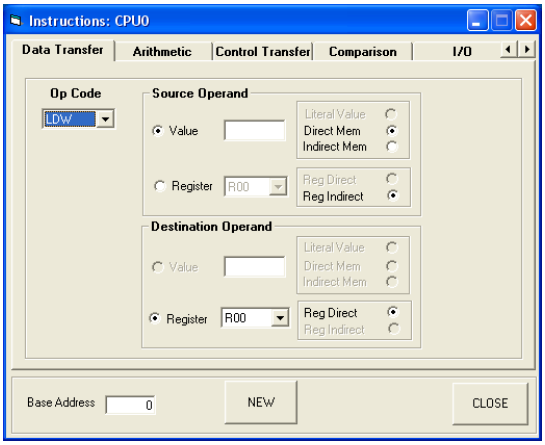

Fig. 4. Instruction Selector Window

Next, the students are asked to repeat the same simple tasks by using the appropriate high-level language constructs of the simulator's teaching compiler. They are then instructed to analyze the low level code generated by the compiler and compare that against their own code and comment on the differences.

7.2 Compiler Optimization and Pipeline

In an advanced module on computer architecture, concentrating on performance issues, the students are asked to write a simple loop using the simulator's teaching language and compile it. They are then asked to note the size of the code generated. The students then run the code on the CPU simulator and using the pipeline simulator observe the total number of instructions executed and the resultant average clock cycles per instruction (CPI).

Next, the students re-compile the same code but this time with the "loop unrolling" optimization of the compiler switched on. They are then asked to note the size of the code generated. The students run the new code on the CPU simulator and once again observe the total number of instructions executed and the resultant average clock cycles per instruction (CPI). Fig. 5 shows the

CPU pipeline simulator window. Each stage of a pipeline and any pipeline hazards are identified by colour coded blocks.

In the observations and analysis part of the practical tutorial session the students observe that although the optimization significantly increased the static code size, the total number of instructions executed is reduced and the CPI is also reduced thus indicating performance improvement.

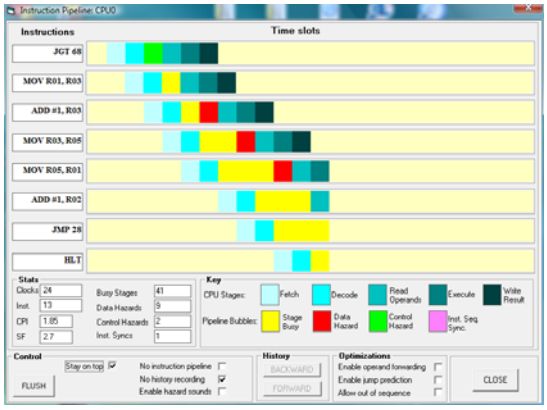

Fig. 5. The CPU Pipeline Window

Students then experiment with other popular compiler optimizations, e.g. removing redundant code, to observe dramatic reductions in the size of the code generated.

7.3 Multiprogramming and Multithreading

In a module on operating systems, the students are given a simple programming task and are asked to multi-program several instances of the same program. The students need to identify the steps necessary to do this. It is assumed that they have some experience of using the simulator.

First, they need to compile the code using the teaching compiler of the simulator. Next, they need to load this code in the CPU's memory. This makes the code "visible" to the operating system (OS) simulator. They then manually create several instances, i.e. processes, of the program using the simulator. The students then need to configure the OS simulator by selecting the round-robin scheduling option of the simulator and setting the time slot for context switching between the processes. The OS scheduler is then manually started to run the processes. The students observe context switching and get to analyze the contents of each process' Program Control Block (PCB) just before and after context switching takes place.

The students are next asked to create a subroutine in their program and run it as a thread within the main program; the teaching compiler provides a special construct for this. They then run the program and observe the thread running as a child of the main program sharing its parent's memory space. The students are next encouraged experimenting further by creating nested threads.

7.4 Investigating Deadlocks

In a module in operating systems the students investigate deadlocks. They are asked to create several processes and then create the conditions necessary for a deadlock between some of the processes. After this they are requested to resolve the deadlock condition.

The OS simulator provides a means of allocating resources to a process. It also allows a process to "request" other resources. The teaching compiler has constructs to allocate and release resources as an alternative method.

The students create the deadlock when a cyclic resource allocation condition is achieved. At this point the deadlocked processes are identified by the simulator and are put in the waiting queue. The students can then resolve the deadlock by manually releasing resources from processes or by terminating selected processes until the deadlock conditions are removed.

Fig. 6 shows the resource allocation window. Using this tool it is possible to allocate virtual resources to processes. It is also possible to configure conditions which can prevent deadlocks from occurring and enable the OS to recover from them.

8 Evaluation

Although the simulator has been in use in both the introductory and the advanced modules it became necessary to evaluate its effectiveness as a useful teaching and learning tool and at the same time use the resultant feedback to improve its functionality.

8.1 Methodology

The evaluation of the simulator was carried out using both qualitative and quantitative methods. The qualitative method used opinion surveys as a means of gathering primary data. The surveys were implemented using 5-point Likert scale (viz. strongly disagree, disagree, not sure, agree, strongly agree) as well as some open questions. The quantitative method used quasi experimental

procedures with non-random test and control groups as well as pre and post tests. These tests were implemented in multiple-choice quiz formats. All questions allowed only a single correct choice.

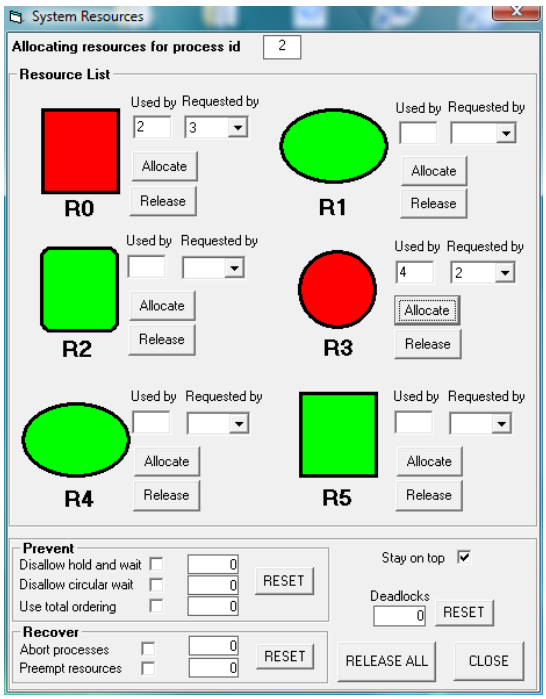

Fig. 6. The OS Resource Allocation Window

The participants were composed of 36 1st year computing degree students studying the introductory computer architecture module and 18 2nd year computing degree students studying the advanced computer architecture module. The 1st year students participated in two groups which were non-randomly selected. One of the groups was used as the control and the other as the test group. These roles were then reversed. The control groups attempted the same exercises as the test groups but without using the simulator. The 2nd year students participated in a single group. These students were given the pre and post tests. The tests mirrored the same topics covered by the tutorial exercises using the simulator.

One additional qualitative dimension was provided by the observations of the three tutors who facilitated the evaluations during the practical tutorials using the simulator.

8.2 Summary of Results

One of the opinion surveys was conducted at the start of the evaluation phase and another at the end of this phase. A total of 42 students participated in the surveys.

TABLE 2
OPINION SURVEY 1

Question	Strongly Agree + Agree (%)	Strongly Disagree + Disagree (%)
The simulator helped me understand the theory and concepts covered during the lecture better	79.3	10.3
I got more confused about the CPU architecture when I used the simulator during the tutorial	20.7	72.4
I was encouraged and enjoyed exploring different aspects of the CPU architecture using the simulator	79.3	13.8
I spent more time learning how to use the simulator than actually doing the tutorial exercises using it	20.7	72.4
The simulator encouraged the members of my group to work together in order to solve the tutorial problems	79.3	10.3
I found the simulator exercises supportive of and appropriate for the topics covered in the lecture	89.7	6.9
I know of/used other software or method more suitable than the simulator for understanding of the subjects covered in the lecture	6.9	79.3
The tutorial exercises using the simulator were set at the appropriate level of difficulty and challenged me	79.3	17.2

Table 2 shows the eight opinion questions included in the first survey. The “Strongly Agree/Agree” and the “Strongly Disagree/Disagree” results are aggregated and presented as percentages.

Table 3 shows the three opinion questions included in the second opinion survey. The results are presented in a similar manner to the first survey results.

The first survey indicates that the students are of the opinion that the simulator helps them understand better the theory and the concepts covered during the lectures. They are also encouraged to explore further and to work in groups using the simulator. On the other hand a small number of students are of the opinion that the simulator is more complicated to use than it should be.

The second survey was conducted after the students used the simulator for a while. This survey indicates that overall, the students were happy using the simulator and regarded it as an effective educational tool.

The simulator is expected to be more effective for students with certain types of learning styles. We then surveyed the participating students for their learning styles. Table 4 shows the results of this survey.

It appears that 83% of the participating students characterized themselves as visual or kinesthetic learners. This is exactly the type of learners the simulator is primarily designed for.

TABLE 3
OPINION SURVEY 2

Question	Strongly Agree + Agree (%)	Strongly Disagree + Disagree (%)
Overall, the simulator has been a useful tool in understanding some of the more difficult concepts	95.2	0
I found the simulator too complicated to understand and use effectively in most simulator-based tutorials	7.1	81
The simulator has been more effective in helping me understand some of the difficult concepts than reading the text books or searching on the Internet	95.2	0

TABLE 4
STUDENT LEARNING STYLES

Learning Style	Number (%)
Visual Learner	31
Auditory Learner	5
Kinaesthetic learner	52
Reader Learner	7
Note: 2 students did not know	

The quasi experimental procedures revealed a slightly different picture and at first sight do not appear to agree with the general sentiments of the opinion surveys. Table 5 shows the results of the pre and post tests carried out on the 2nd year participants as a single group. The results were obtained over the scheduled three practical tutorial sessions. The numbers represent the scores for each question in percentage. The students were given the pre test quiz which was followed by the main tutorial session with the simulator exercises. At the end of the exercises the students completed the post test quiz. It is interesting to note that although the average results do not show significant improvement, there were significant improvements in some of the individual post test questions. On further inspection, these "hot spots" were found to correspond to questions with mainly visual elements.

Evaluating the results of a different aspect of student learning involved a control group and a test group of 1st year students. These results are shown in Table 6. Interestingly, the results indicate that the students using the simulator achieved significantly improved number of attempts and correct solution scores in these particular exercises which were expected to benefit from the visualization features of the simulator.

9 CURRENT STATE OF THE RESEARCH

The simulator project has been partly supported by the funding from Higher Education Academy (HEA) which enabled us to carry out evaluations on the effectiveness of the simulations as a teaching and learning resource. The funding, which is for a period of six months, also aims to support the dissemination of the results and a dedicated web site will be created for this purpose.

TABLE 5
2ND YEAR PRE/POST TEST RESULTS

1. Subject: Investigating Compiler Technology									
Test	Replies	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Avg.
Pre	19	62.5	45.3	18.8	23.4	28.1	15.6	6.3	27.3
Post	19	78.6	44.6	64.3	12.5	35.7	21.4	0	36.4
2. Subject: Understanding Instruction Pipelining									
Pre	16	93.8	75	56.3	87.5	43.8	81.3	37.5	67.9
Post	17	100	82.4	23.5	76.5	64.7	94.1	58.8	71.4
3. Subject: Investigating Cache Technologies									
Pre	18	50	72	100	78	100	78	50	75.4
Post	17	59	76	88	76	94	76	53	74.8

TABLE 6
1ST YEAR CONTROL/TEST RESULTS

Subject: Programming Model 4 - Elements of high-level languages		
Group Type	Average Attempted (%)	Average Correct (%)
Control (paper exercises)	77	59.8
Test (simulator)	93	75.7

Currently the results of our evaluations are being analyzed in more detail and it is our intention to further extend the evaluations in the light of our experiences to date for which we will be seeking further funding.

10 FUTURE WORK

The simulator's current state is maturing and is fairly stable. The extension of the simulator to cover areas of system architecture which are increasingly being included in our modules will undoubtedly further enhance the education of our students. Areas of development for which further funding will be sought are listed below:

- Distributed OS simulation
- Superscalar CPU architectures

- Extended input/output devices

11 CONCLUSIONS

The creation of yet another educational simulator has been fully justified on the grounds that there is a need for a unified and integrated approach to facilitating the teaching and learning of computer architectures and operating systems concepts included in undergraduate computing courses. The research of the existing simulators revealed a fragmented presence of many simulators and none, as far as this author is aware, offers the means of facilitating deep understanding of the concepts as unified and co-operating set of complex technologies.

It is this desire of a unified approach that prompted the author to initiate a new simulator project in the first place. It is hoped that the present system simulator as described in this paper will go some way to closing this gap in educating future computer scientists and professionals.

ACKNOWLEDGMENT

The author wishes to thank Higher Education Academy (HEA), UK, for providing funding in support of the evaluation stage of this work.

REFERENCES

[1] Computing Cirricula 2001 (2001). Computing Science, Final Report, December 15 2001. ACM and IEEE Computer Society joint report, USA.
[2] Computing 2007 (2007). Subject Benchmark Statement. The Quality Assurance Agency for Higher Education, UK, 2007.
[3] The Future of Higher Education (January 2003), UK Government white paper, Dept. of Education and Skills.
[4] Yurcik, W., Wolffe, G.S., and Holliday, M.A. (2001). A survey of simulators used in computer organization/architecture courses. SCSC 2001, Orlando FL, USA, July 2001.
[5] Yehezkel, C., Yurcik, W., Pearson, M. and Armstrong, M. (2002). Three simulator tools for teaching computer architecture: EasyCPU, Little Man Computer, and RTLSim. ACM Journal of Educational Resources in Computing, Vol. 1, No. 4, December 2001, Pages 60-80.
[6] Chan, T.W. (2004). A Software Tool in Java for Teaching CPU Scheduling. JCSC 19, 4 (April 2004).
[7] Than, S. (2007). Use of a simulator and an assembler in teaching input-output processing and interrupt handling. JCSC 22, 4 (April 2007).
[8] Nishita, S. (2004). MKit simulator for introduction of computer architecture. 31st International Symposium on Computer Architecture, June 19, 2004, Munich, Germany.
[9] Maia, L.P. and Pacheco, A.C. (2003). A Simulator Supporting Lectures on Operating Systems. 33rd ASEE/IEEE Frontiers in education Conference, November 5-8, 2003, Boulder,CO.
[10] Robbins, S. and Robbins, K.A. (1999). Empirical exploration in undergraduate operating system. SICCSE'99, 3/99 New Orleans, LA, USA.
[11] Ianov, L. and Mallozi, J.S. (2004). A Hardware/software simulator to unify courses in the computer science curriculum. JCSC 19, 5 (May 2004).
[12] Null, L. and Lobur, J. (2003). MarieSim: The MARIE computer simulator. ACM journal of Educational Resources in Computing, Vol. 3, No. 2, June 2003, Article 1.
[13] Shelburne, B. (2003). Teaching computer organization using PDP-8 simulator. SIGSCE'03, February 19-23 2003, Reno, Nevada, USA.
[14] Styer, E. (1994). On the design and use of a simulator for teaching computer architecture. SIGCSE Bulletin, Vol. 26 No. 3, sept. 1994.
[15] Skrien, D. (2001). CPU Sim 3.1: A tool for simulating computer architectures for computer organization classes. ACM Journal of Educational Resources in Computing, Vol. 1, No. 4, December 2001, Pages 46-59.
[16] Vollmar K. and Sanderson, P. (2006). MARS: An education-oriented MIPS assembly language simulator. SIGCSE'06, March 1-5, 2006, Houston, Texas, USA.
[17] Robbins, S. (2001). Starving philosophers: Experimentation with monitor synchronization. SIGCSE 2001 2/01 Charlotte, NC, USA.
[18] Robbins, S. (2005). An address translation simulator. SIGCSE'05 February 23-27, 2005, St. Louis, Missouri, USA.
[19] Morsiani, M. and Davoli, R. (1999). Learning operating systems structure and implementation through the MPS computer system simulator. SIGCSE'99 3/99 New Orleans, LA, USA.
[20] Burrell, M. (2004). Fundamentals of Computer Architecture. Palgrave Macmillan.
[21] Garrido, J. M. and Schlesinger, R. (2008). Principles of Modern Operating Systems. Jones and Bartlett.
[22] Wolffe, G.S., Yurcik, W., Osborne, H. and Holliday, M.A. (2002). Teaching computer organization/architecture with limited resources using simulators. SIGCSE'02, February 27-March 3, 2002, Covington, Kentucky, USA.
[23] Stallings, W. (2009). Operating Systems Internals and Design Principles. Sixth edition. Pearson.
[24] Null, L. and Lobur, J. (2006). The Essentials of Computer Organization and Architecture. Second edition. Jones and Bartlett Publishers, Inc.
[25] Atkin, B., and Sirer, E.G. (2002). PortOS: An educational operating system for the post-PC environment. Proceedings of the 33rd ACM SIGCSE Technical Symposium on Computer Science Education, pages 116-120, Covington, Kentucky, February 2002.
[26] Hovemeyer, D., Hollingworth, J.K. and Bhattacharjee, B. (2004). Running on the bare metal with GeekOS. SIGCSE'04, March 3-7, 2004, Norfolk, Virginia, USA.

Besim Mustafa Holds BSc (1973) in Electronics from Manchester University and MSc (1985) in Computing from Heriot-Watt University, UK. He also holds a PCert. in HTLS from Lancaster University (2006), UK. He was senior hardware and software design and development engineer at International Computers Ltd. (UK), Unisys and Critical Path. He is currently a Senior Lecturer at Edge Hill University. He was a member of the IEE and a Chartered Engineer (1985 – 2000). He is currently researching visualization and multi-level simulations supporting degree level computer architecture and operating systems courses.