YASS: A system simulator for operating system and computer architecture teaching and learning

Besim Mustafa
Department of Computing, Edge Hill University, Ormskirk, U.K.
For correspondence: mustafab@edgehill.ac.uk

Abstract: A highly interactive, integrated and multi-level simulator has been developed specifically to support both the teachers and the learners of modern computer technologies at undergraduate level. The simulator provides a highly visual and user configurable environment with many pedagogical features aimed at facilitating deep understanding of concepts which are often difficult to grasp by the students. The rational behind the development is explained and the main features of the simulator are described. A brief account of the ways in which the simulator has been used to support undergraduate lectures and tutorials is given. The current state of the research in assessing and evaluating the value of the simulations at undergraduate levels is presented.

Introduction

The teaching of computer architecture which includes key subject areas such as production of executable code, instruction set architectures (ISAs), performance enhancing features and principles of operating systems form the core set of subjects for most computing and computer science undergraduate degree programmes. These topics have been identified both in the ACM/IEEE computing curricula (USA) (Computing Curricula, 2001) and the QAA's subject benchmark in computing (UK) (Computing 2007, 2007) reports. In most cases, the teaching of these topics involve combination of methods such as traditional lectures, individual programming assignments, modification of educational operating systems and simulations. Exactly which combination 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 Edge Hill 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 go on to studying more advanced topics in the second year. The delivery of the programme includes traditional theory via lectures supported by tutorial and practical sessions as well as individual and group coursework assignments. The students studying for computing degrees are recruited from wide educational backgrounds and competencies which may not include any previous computing experience or qualifications.

The tutorial and practical sessions on computer architecture have been supported by investigating aspects of different operating systems, mainly Windows and Linux. However, there remained a requirement for studying those architectural features which are difficult or impossible to access and demonstrate on real systems. With this in mind, it was decided that a software simulator would be developed with features designed to support the computing modules in computer architecture and operating systems.

On the Motivation

The motivation for developing a new educational simulator from scratch is prompted by the need for the following main requirements

An integrated system simulator. In computer architecture, different technologies are interrelated and support each other across clearly defined interfaces. It is these interdependencies and the interplay that the integrated simulator aims to represent. The students prefer to see the "big picture" and to understand how things "hang" together.

Rich pedagogical features. As an educational tool, it is important that a simulator enhances and enriches learning experiences of students and at the same time facilitates deep understanding of the key technological concepts and issues. It should actively encourage experimentation, exploration and investigative problem solving with students working either individually or in groups.

Control and monitoring facilities. In a dynamic simulation environment it is essential to be able to suspend, stop and re-start the simulations on the occurrence of some pre-determined event or state. It may also be necessary to manually make changes to selected system components (e.g. memory, registers, instructions, etc.)

Support at different 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.

Integrated visual displays. Animated, real-time, colour coded, visual displays can provide immediate feedback and impact which can help reveal trends, show state transitions and facilitate comparisons.

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.

Little or 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.

Support for advanced features. The simulator is designed to support advanced architectural features such as multiple processors, instruction level pipelining, compiler optimizations and virtualization.

Easy and intuitive user interface. The users of the simulator should not be faced with a steep learning curve. As the same simulator is expected to be used across different modules and over different years of study, the students soon become familiar with its usage.

Prior Work

Over the years, many simulators of computer architectures have been developed which have been used as valuable educational resources (Yurcik et al, 2001 and Yehezkel et al, 2002) at undergraduate computing courses. These range from simple, abstract, high-level simulators to advanced simulators of commercial CPUs.

Table 1 lists some examples of software simulators developed for educational purposes. The simulators are categorised 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.

Table 1. A survey of some of the educational simulators, past and present.

Name	Simulator		Comments
	OS	CPU	
SchedulerSim (Chan, 2004)	Yes	No	CPU scheduling concept
Sim. + assembler (Than, 2007)	No	Yes	IO processing + interrupt handling
MKit (Nishita, 2004)	No	Yes	Inst. set + data paths + control unit
SOsim (Maia, Pacheco, 2003)	Yes	No	Process + memory management
Sim. (Robbins, Robbins, 1999)	Yes	No	Process scheduling, HTML-based, scripted
Sim. (Ivanov and Mallozi, 2004)	No	Yes	Assembler + inst. set
MarieSim (Null, Lobur, 2004))	No	Yes	Assembler + inst. set + data paths
PDP-8 simulator (Shelburne, 2003)	No	Yes	PDP-8 inst. set + assembler
Simulta (Styer, 1994)	No	Yes	Inst. set + microcode + input/output
CPU Sim (Skrien, 2001)	No	Yes	Inst. set + microcode + assembler
MARS (Vollmar, Sanderson, 2006)	No	Yes	MIPS assembly language simulator
Starving philosophers (Robbins, 2001)	Yes	No	Limited OS: Syncronization + monitors
Address translation (Robbins, 2005)	Yes	No	Limited OS: Virtual memory
MPS (Morsiani, Davoli, 1999)	No	Yes	Inst. set + input/output + MIPS CPU sim.
JASP toolkit (Burrell, 2004)	No	Yes	Inst. set + assembler + high-level lang.
PsimJ sim. (Garrido, Schlesinger, 2008)	Yes	No	Various isolated OS component simulations

There have been surveys of many other software simulators and visualization tools designed to support computer architecture education at universities and colleges (Yurcik et al, 2001 and Wolffe, et al, 2002), 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 (Garrido and Schlesinger, 2008, Stallings, 2009, Burrell, 2004 and Null and Lobur, 2006). 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 which the students are asked to modify and/or extend (Atkin and Sirer, 2002 and Hovemeyer et al, 2004). This approach often requires good programming ability by the students and, although highly realistic, is not always suitable as a teaching and learning resource.

Simulator Design Details

The design and development of the system simulator are based on clearly defined design principles. 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.

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. Image 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.

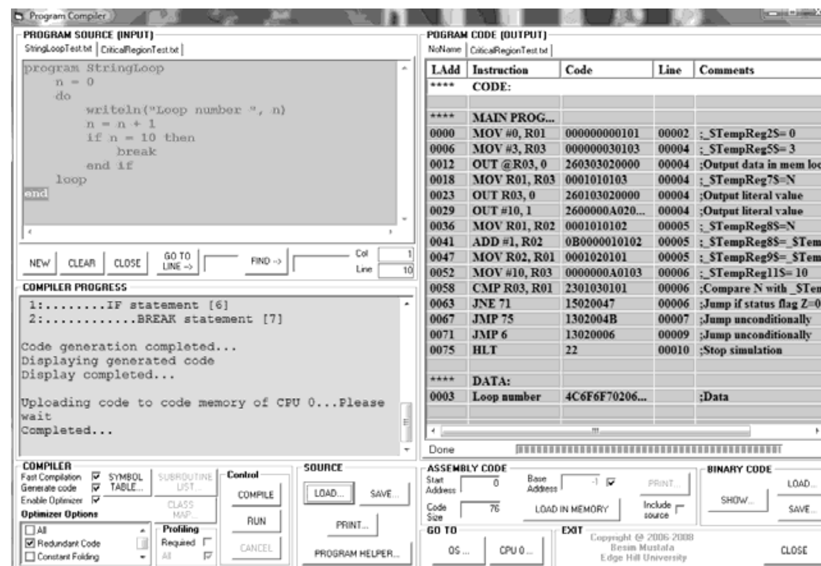

Image 1. The compiler main screen.

The CPU simulator. The CPU is loosely based on RISC architecture with a prominent register file composed of from 8 to 64 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 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.

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. Image 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 processors simulations. Each processor is identical and loading code in one is duplicated in others. 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.

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. Image 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 includes 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 which is used to display text and accept input data.

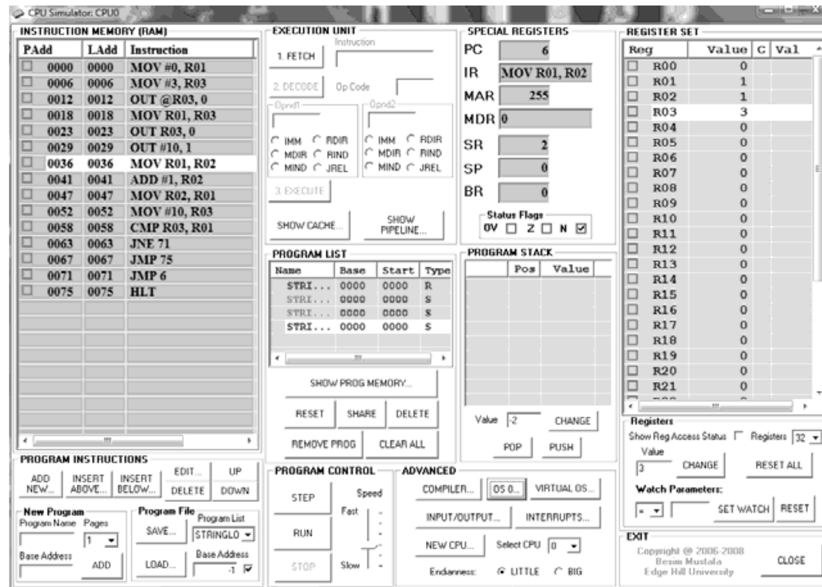

Image 2. The CPU simulator main screen.

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.

Image 3. The OS simulator main screen.

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.

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.

The Virtual Machine (VM)

A separate stand-alone virtual machine has been developed which is able to interpret and execute the compiled byte-code. The VM is a native executable code and is currently implemented on Windows and Linux operating systems. 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 code on different platforms.

Some Notable Features

The system simulator boasts some notable features, not available elsewhere. Below is a summary of some of these features.

Compiler. The compiler incorporates object-oriented (OO) features and can be used to demonstrate inheritance, encapsulation and polymorphism. The students can also observe the way the code is generated for object-oriented programs.

A language construct is available to demonstrate the code generated for exception handling. The programmer can specify areas of code that can be protected as in Java programming language.

The Inter Process Communications (IPC) is an important aspect of modern computer architecture. The simulator's language includes constructs which generate system calls that support IPC.

The compiler's source editor and the view displaying the corresponding code generated are context sensitive. So, as the cursor is moved or placed on a particular line of source, the corresponding code generated is highlighted and vice versa. This makes it easy for the student to observe the code generated corresponding to each line of source statement.

CPU simulator. The compiler favours registers as locations for program variables. However it can be forced to spill these over to the memory by generating the appropriate load/store instructions if the number of registers is set low.

The register file includes a "watch" facility where selected registers can be specified with a value and a condition upon which the simulation will be suspended.

In addition, the registers in the register file can be tagged with visible markers (e.g. images) which indicate the status of registers with respect to the stages of the pipeline for hazard conditions.

OS simulator. The OS simulator implements inbuilt system calls. The system calls use an instruction which causes software interrupt and passes a parameter to the OS indicating the type of the call. As system calls are frequent occurrences, the students need to understand the general mechanisms involved.

One of the interesting features of the OS simulator is its ability to create and run processes automatically for extended periods. The input/output events, resource allocations/de-allocation, deadlocking, process memory pages, page swapping, process scheduling, context switching are all randomly simulated.

Current Research

The simulator project has recently secured funding from Higher Education Academy (HEA) to carry out evaluation 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 this of the results; a dedicated web site will be created for this purpose.

Both qualitative and quantitative data will be gathered and the research will concentrate on devising tutorial and lab exercises which will be attempted with and without the support of the simulations. The exercise results will be assessed and a comparison between the two sets of results will be made. There will also be a survey of student opinions on the use of the simulations in underpinning theory.

Further 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 funding will be sought are listed below:

- Distributed OS simulation
- Superscalar CPU architectures
- Extended input/output devices

Conclusions

The creation of yet another educational simulator has been fully justified on the grounds that there is a need for a unified approach to facilitate the teaching and learning of computer architectures including 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 of unification of the technology.

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.

References

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.

Burrell, M. (2004). Fundamentals of Computer Architecture. Palgrave Macmillan.

Chan, T.W. (2004). A Software Tool in Java for Teaching CPU Scheduling. JCSC 19, 4 (April 2004).

Computing Cirricula 2001 (2001). Computing Science, Final Report, December 15 2001. ACM and IEEE Computer Society joint report, USA.

Computing 2007 (2007). Subject Benchmark Statement. The Quality Assurance Agency for Higher Education, UK, 2007.
Garrido, J. M. and Schlesinger, R. (2008). Principles of Modern Operating Systems. Jones and Bartlett.
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.
Ianov, L. and Mallozi, J.S. (2004). A Hardware/software simulator to unify courses in the computer science curriculum. JCSC 19, 5 (May 2004).
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.
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.
Nishita, S. (2004). MKit simulator for introduction of computer architecture. 31st International Symposium on Computer Architecture, June 19, 2004, Munich, Germany.
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.
Robbins, S. (2005). An address translation simulator. SIGCSE'05 February 23-27, 2005, St. Louis, Missouri, USA.
Robbins, S. and Robbins, K.A. (1999). Empirical exploration in undergraduate operating system. SICCSE'99, 3/99 New Orleans, LA, USA.
Robbins, S. (2001). Starving philosophers: Experimentation with monitor synchronization. SIGCSE 2001 2/01 Charlotte, NC, USA.
Shelburne, B. (2003). Teaching computer organization using PDP-8 simulator. SIGSCE'03, February 19-23 2003, Reno, Nevada, USA.
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.
Stallings, W. (2009). Operating Systems Internals and Design Principles. Sixth edition. Pearson.
Styer, E. (1994). On the design and use of a simulator for teaching computer architecture. SIGCSE Bulletin, Vol. 26 No. 3, sept. 1994.
Than, S. (2007). Use of a simulator and an assembler in teaching input-output processing and interrupt handling. JCSC 22, 4 (April 2007).
Vollmar K. and Sanderson, P. (2006). MARS: An education-oriented MIPS assembly language simulator. SIGCSE'06, March 1-5, 2006, Houston, Texas, USA.
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.
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.
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.