Simulating CPU Pipelining for Computer Architecture Teaching and Learning Support

Besim Mustafa
Business School
Edge Hill University
Ormskirk, UK.
+44 (0) 1695 657640
mustafab@edgehill.ac.uk

ABSTRACT
An important area of modern computer architecture which is difficult to teach and learn is the CPU pipeline technology. This paper describes an educational CPU pipeline simulator which is part of an integrated system simulator. The various functional features of the simulator are explained; the simulator's user interface is introduced; examples of the usage of the simulator by the lecturers and the students are described in some detail; finally, a brief account of the work done in evaluating the simulator is given.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education --- *Computer Science Education;*
C.1.1 [**Computer Systems Organization**]: Processor Architectures --- *Single Data Stream Architectures*.

Keywords
Computer architecture, processor instruction pipeline, visualization, simulation.

1. INTRODUCTION
One of the important aspects of modern computer architectures which the students of computer science degrees often find difficult to grasp and the teachers experience difficulty in explaining during the lectures is the processor (CPU) instruction pipelining technology. Simply put, the processor pipeline is a complex hardware designed to optimize the processing of instructions by dividing each instruction into distinct stages and executing the stages of different instructions in parallel. This technology enables most modern processors to execute each instruction in a single clock cycle thus significantly improving processor performance.

The author has been responsible for designing and delivering two modules on computer architecture at undergraduate degree level for the past five years. One of the modules is taught in the second year and includes advanced architectural features of which processor pipelining is one. In order to support the practical lab sessions he has implemented an integrated system simulator which includes a teaching compiler, a CPU simulator and an operating system simulator [4]. The CPU simulator incorporates a pipeline simulator and is used to study multi-stage pipeline functionality as suggested by respected CS curricula reports [2, 3].

The system simulator has been successfully integrated into several modules including a module on operating systems and has been in use for the past three years.

2. SYSTEM SIMULATOR
The system simulator integrates three important aspects of computer architecture in one educational software package: generation of CPU instructions using a high-level language compiler; the CPU as the instruction processor; the operating system enabling multiprogramming of the CPU instructions. The system simulator is used to demonstrate the interdependencies and the cooperation between these three areas across well-defined interfaces. The students use the simulators to explore these technologies which would otherwise be too difficult or impossible to achieve.

2.1 CPU Simulator
The part of the integrated system simulator relevant to this paper is the CPU simulator which simulates the hardware functionality of a fictitious, but highly realistic, CPU based on RISC type architecture. This simulator incorporates a five-stage pipeline simulator and a hardware cache simulator which are used to support both the introductory and the advanced modules in computer architecture. The CPU simulator executes instructions which are either automatically generated by the integrated compiler from source code or manually created by the students. Both the cache and the instruction pipeline simulators cooperate with the CPU simulator while the instructions are being executed.

3. CPU PIPELINE SIMULATOR
The configurable and visual CPU pipeline simulator is used in both the introductory and the advanced modules. This is made possible by the various configuration options available; the students of a first year introductory computer architecture module can use the pipeline simulator to simply observe and study the different stages of instruction execution (e.g. fetch, decode and execute) with no reference to pipeline hazards or stalls which can be introduced in an advanced architecture module normally studied during the second year.

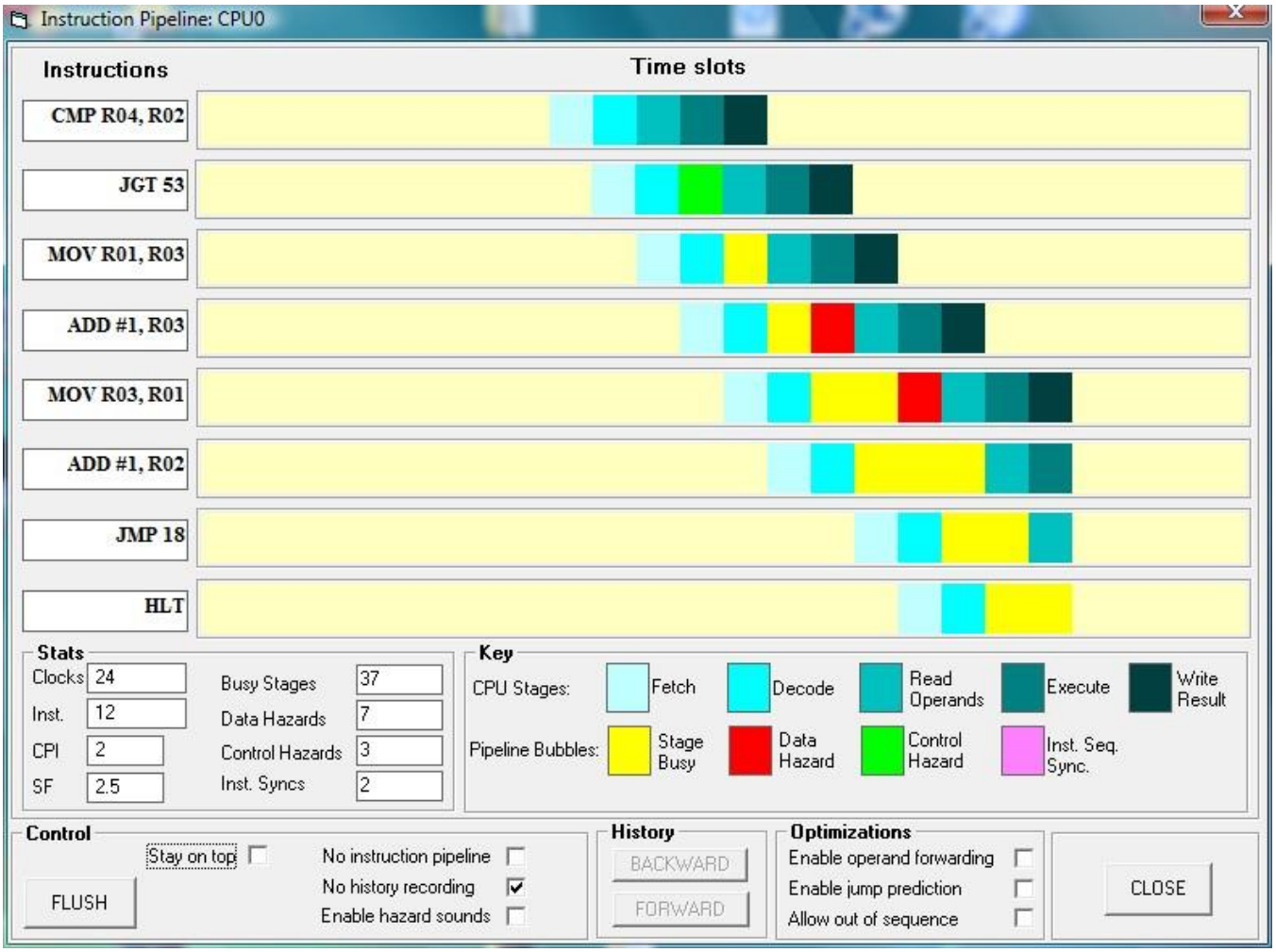


Figure 1. CPU pipeline main user interface.

3.1 Simulator Interface

Figure 1 shows the CPU pipeline simulator user interface. The instructions entering the pipeline are shown on the left and they progress from bottom up in the column of instructions. The different stages of the pipeline are identified by different colours and the hazards are also colour-coded. In Figure 1 these are shown in the frame with the heading "Key". The pipeline simulator maintains a set of relevant statistics as the instructions are processed. These include the total numbers of simulation clocks, the total number of instructions executed, the average clocks per instruction (CPI) and the speed-up factor (SF) as well as the counts of the various pipeline hazards and stalls. The speed-up factor is the ratio of the CPI in the absence of the pipeline to the CPI when the pipeline is available for the same set of instructions. Using the FLUSH button the pipeline can be manually flushed. It is also possible to disable the pipeline (a feature not normally available on real CPU hardware) in order to demonstrate the loss of CPU performance in the absence of a pipeline.

By default, the pipeline hazards are not prevented. This is necessary in order to be able to study them. However, it is possible to prevent different types of hazards by configuring the pipeline accordingly. Figure 2 shows the options available. When set, the first two options switch on those features of the pipeline which eliminate data dependency hazards and control hazards by using the optimization methods operand forwarding and jump prediction. A third option is for configuring the simulator to allow out of sequence execution of instructions.

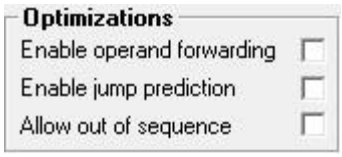


Figure. 2. CPU pipeline optimization options.

3.2 Interpretation of Simulation Data

The integrated system simulator incorporates a compiler used for producing executable code. For example, Figure 3 shows the integrated system simulator's built-in compiler depicting a source code and its assembly level code for demonstration purposes.

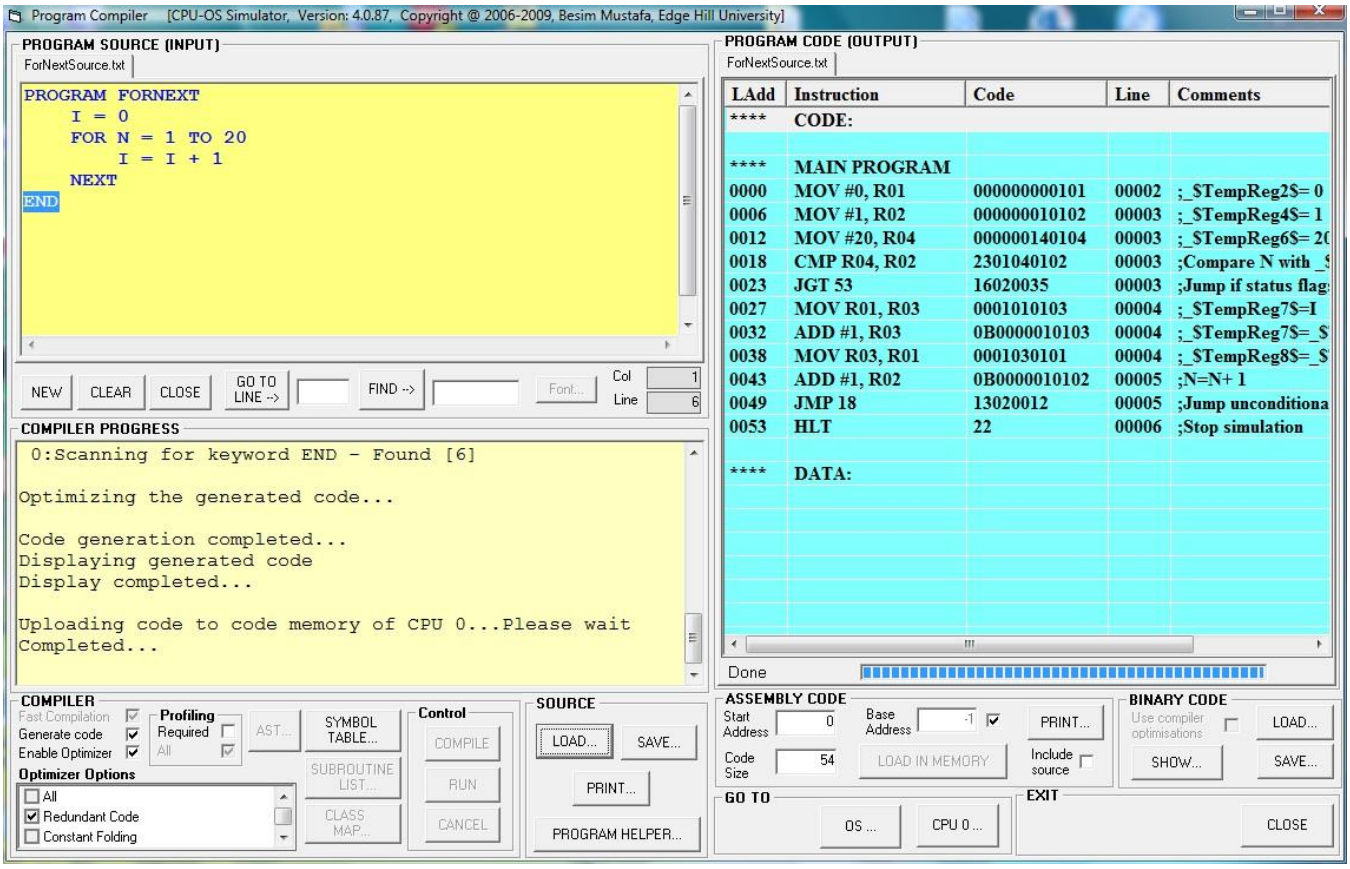


Figure 3. Compiler user interface

The generated code is then executed by the CPU simulator and an instance of which is captured in the pipeline simulator's window. This is shown in Figure 1. Here, consider the top two instructions: CMP R04, R02 (compare instruction with registers 4 and 2 as the operands) and JGT 53 (jump if greater than instruction). It can be seen that the jump instruction is unable to continue as it will not be able to complete before the result of the compare instruction is known. So, after the fetch and decode stages of the jump instruction, the result of the previous instruction is needed but that will not be available until the compare instruction executes and stores back its result during the next pipeline stage. As a result, after the jump instruction's decode stage a control hazard bubble, represented by a green block, is inserted. Similarly, consider the two instructions: MOV R01, R03 and ADD #1, R03. Here, the ADD instruction cannot complete until the MOV instruction writes back the result in register R03. So, the ADD instruction is able to read the R03 operand only after the MOV instruction completes its write back pipeline stage. As a result, the pipeline simulator inserts a red bubble block after the ADD instruction's decode pipeline stage. This represents a data dependency hazard.

An instruction which targets a register for writing the results to, such as a MOV or an ADD instruction, needs to be flagged as "write-pending" until the result is written back. Once a register is in this state, any following instruction wishing to write to the same register will stall until the pending write is completed. The occurrence of this event is captured by the CPU simulator and displayed. Figure 4 shows a section of the CPU register file where the state of register R01 in instruction MOV R03, R01 (shown in Figure 1) before R01 was updated. The small "**w**" letter is displayed against this register as long as it is pending a write. The simulator maintains a "scoreboard" [5] for all the registers which is used to keep track of pending reads and writes.

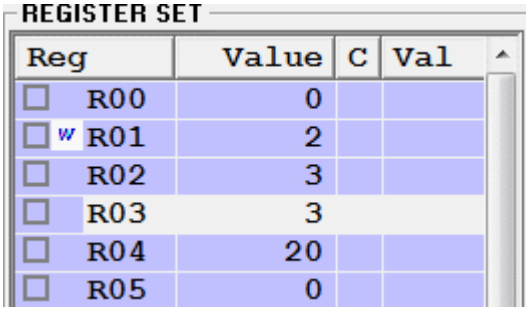


Figure 4. Register "write-pending" flag.

Additionally, the pipeline simulator incorporates instruction execution history feature which can be used to capture instances of code runs. This feature is enabled or disabled as a configuration option and, in addition, provides a "play-back"

facility. This allows scrolling through the captured pipeline stages for further analysis by the students.

4. CASE FOR PIPELINE SIMULATOR

The justification for the integrated system simulator of which the pipeline simulator described in this paper is a part, is the need for a visual simulation environment in which the students are encouraged to appreciate how different parts and layers of a modern computer system fit and work together across well-defined hardware and software interfaces. In this respect, it is unique; majority of educational architecture and operating system simulators function in isolation often simulating a distinct part of a complex system. The pipeline simulator described here is therefore part of an integrated system and clearly demonstrates how it relates to the rest of the system.

There are some other educational pipeline simulators [6, 7] but these tend to be based on particular CPU architectures described in [8, 9] which are unsuitable for our teaching modules and cannot be integrated into our system simulator. Additionally, the integrated compiler is designed to provide support for the system simulations for enhanced pedagogical value.

5. TEACHING, 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.

6. USING THE PIPELINE SIMULATOR

In this section we give some typical examples of tutorials in which the pipeline simulator is used by the students in order to develop a good understanding of the principles involved in modern CPU pipeline technology. The following sections describe five pipeline simulator tutorial exercises designed to help students grasp the essentials of CPU pipelining.

It is important to note that the pipeline simulator is not used in isolation from the rest of system simulator; it works in conjunction with the teaching compiler which generates the actual code and the CPU simulator which executes it.

6.1 Tutorial 1: Pipeline or no Pipeline

In a practical tutorial session the students studying introductory computer architecture module explore the sequence of execution of CPU instructions first in the absence of a pipeline and then using a pipeline and they note and comment on the differences they observe.

The pipeline simulator allows the pipelining functions to be "switched off". This forces the CPU to execute the instructions in sequential order only. In this case a new instruction will not be fetched until after the processing of a previous instruction is completed. The students run the code of a simple loop and observe the five stages of instructions being processed in sequence. At the end of the code, they also note down the CPI and the SF values. Figure 5 shows an instance of such a run. They then "switch on" the pipeline and run the same code, also noting down the new values of CPI and SF. Figure 1 shows an example of this run.

The students observe that without a pipeline the CPI is recorded as 5 and the SF is recorded as 1. They are then expected to conclude that none of the different stages of the instructions are processed in parallel in the absence of a pipeline and that after 5 instructions are completed in 24 clock cycles, the average CPI is 4.8, nearly equal to the number of stages for each instruction, with a baseline SF of 1. They then compare this against the results of a pipelined run. In this case they observe that the CPI is reduced to 2 and the SF is increased to 2.5 meaning that, in this particular case, the pipelined run is 2.5 faster than the non-pipelined run. They can now conclude that this is a significant improvement in CPU performance.

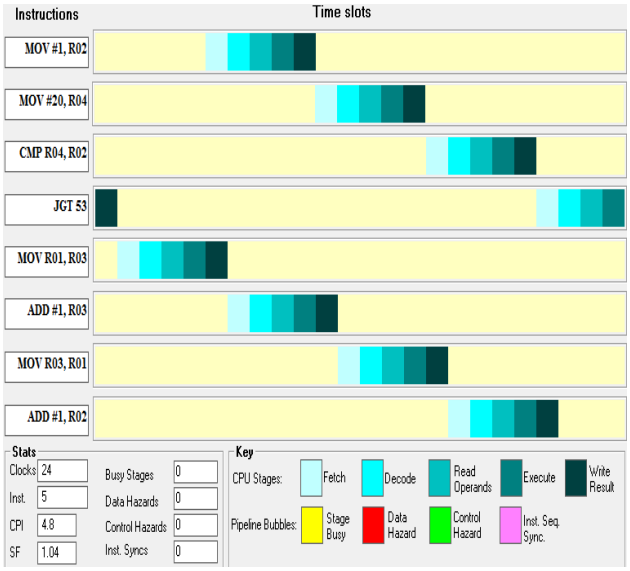


Figure 5. Non-pipelined run.

6.2 Tutorial 2: Pipeline with no Bubbles

Now that it is demonstrated that the CPU pipeline helps improve the CPU performance the students study pipelining technology further. The students are informed that pipelining brings with it certain problems which need to be resolved. The concept of "bubbles" is introduced and the need for this is explored. First, the students configure the simulator to function without bubbles. Next, they are asked to enter and run the following sequence of code

MOV #1, R01

MOV #5, R03

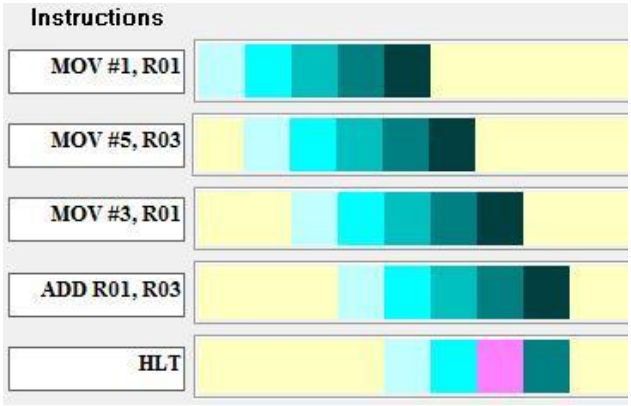
MOV #3, R01

ADD R01, R03

HLT

The students note the content of register R03 at the end of the run. They should observe that the register R03 contains the value 6 at the end of the run. This is different than the expected value of 8. The students are asked to explain this result by observing the sequence in which the different stages of the pipeline are processed for each instruction. As can be seen from Figure 6, the stage at which the operand R01's value of the instruction ADD R01, R03 is obtained before the operand R01's value in instruction MOV #3, R01 is written back. As a result, the ADD instruction picks up the old value of 1 of register R01 thus yielding the result 6 instead of 8.

Later, the students are introduced to a special instruction included in most CPU instruction sets. This instruction has no side effects at all and is usually known as the NOP instruction. The students are then asked to suggest a way to use this instruction to correct the erroneous result in register R03.



They are guided to considering the effect of inserting a NOP instruction between the MOV and the ADD instruction and trying it out using the simulator. Figure 7 shows the effect of

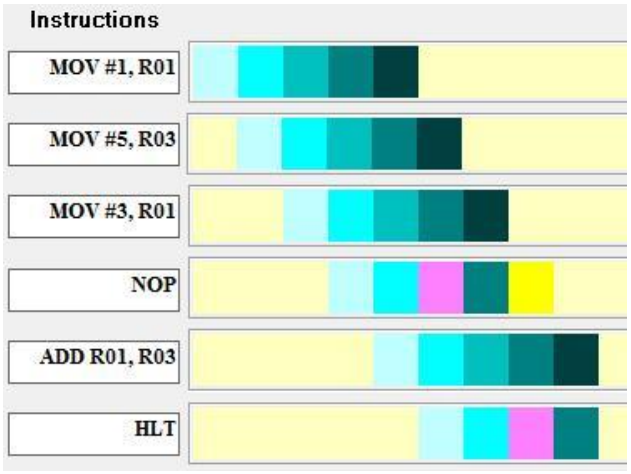
Figure 6. Pipeline with no bubbles.

inserting a NOP instruction on the pipeline.

The NOP instruction introduces a delay during which time the register R01 is updated with a new value before the ADD instruction attempts to fetch the value of R01. This can be clearly seen from the graphical visualization of the pipeline. The students observe that at the end of this run the value of register R03 is set to 8 which is the correct result. The students learn that compilers are normally responsible for inserting NOP instructions, i.e. this is often a software-based solution.

6.3 Tutorial 3: Pipeline with Bubbles

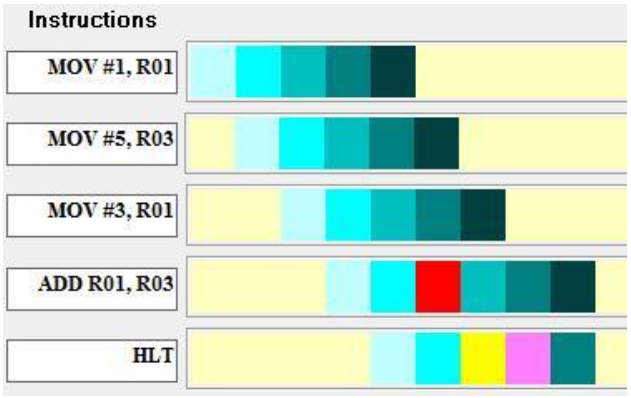
In this exercise, the students are asked to configure the pipeline simulator so that the bubbles are introduced. Bubbles are necessary to make sure the sequence of instructions produce the correct results without NOP instructions inserted which make the code size larger and can reduce performance. In effect bubbles are normally introduced by the hardware.



The students run the same set of instructions as shown in Figure 6 and observe the data hazard bubble (colour-coded red). This time the value in register R03 is the correct value of 8. Figure 8

Figure 7. Using the NOP instruction.

shows the pipeline trace of this run.



In Figure 8 there is an additional red coloured bubble in the ADD instruction soon after the decode stage. This will delay the ADD instruction's operand fetching stage until after the

Figure 8. Pipeline with a Data Hazard bubble.

previous MOV instruction's write-back stage is completed. In the meantime, the pipeline simulator's Data Hazards counter (see Figure 1) will be incremented by one. At the end of this exercise the students are expected to appreciate that pipelines help improve CPU performance but this comes with a price; the pipeline hazards which need to be tackled making the technology more complicated to design. The simulator therefore helps students understand and explain the CPU pipeline by visual means.

6.4 Tutorial 4: Pipeline Optimizations

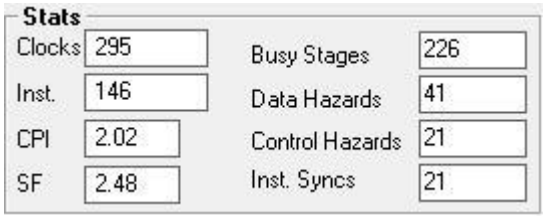
In a more advanced computer architecture module, the students explore the various techniques employed by modern CPU pipelines in improving the performance even when pipeline

hazards are present. The delays introduced by the hazard conditions will effectively have a detrimental effect on the average CPI and the SF values.

The pipeline simulator can be configured to optimize two types of hazards: 1) Data hazards, 2) Control hazards. The data hazards are due to data dependencies between the instructions and the control hazards are due to those introduced by the jump instructions which interrupt the sequential flow of instructions.

The students are normally introduced to methods of dealing with various pipeline hazards during the lectures. The two main methods they learn are the "operand forwarding" which eliminates the data hazards and the "jump prediction" method which eliminates the control hazards due to jump instructions.

During a practical session in advanced computer architecture, the students are first asked to configure the simulator for not eliminating hazards due to data dependencies and jump instructions. They do this by de-selecting the appropriate check boxes (see Figure 2). They are then asked to load and run the simple loop instructions (see Figure 3) and observe the trace of instruction stages and the data hazards counter (see Figure 1). They note their observations at the end of the run. Figure 9 shows the pipeline statistics for this run.

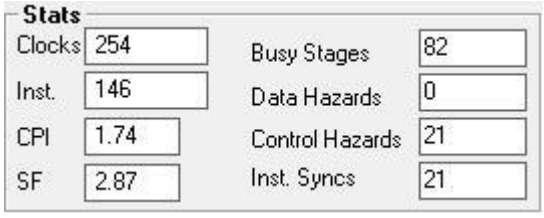


It can be seen from Figure 9 that there have been 41 data hazards and 21 control hazards. Also the students note down the CPI and the SF values for this run.

The students are first asked to configure the simulator to

Figure 9. Statistics with hazards.

eliminate the data hazards and re-run the same code. Figure 10 shows the results for this run. Here, it can be seen that the data hazards have been completely eliminated for this run due to the technique of operand forwarding. Also, as a result, the CPI and the SF values improved thus benefiting the CPU performance.

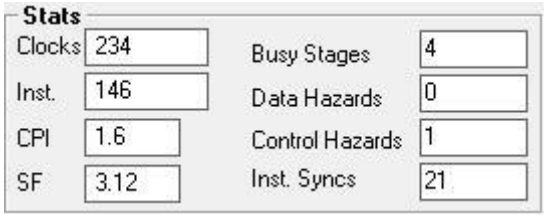


Next, the students are asked to configure the simulator to eliminate the control hazards and re-run the same code one more time. The resulting statistics are shown in Figure 11. The students note that the control hazards are almost completely eliminated as well with only one recorded as opposed to the

Figure 10. Statistics with data hazards eliminated.

previous measurement of 21 (see Figure 10). They now need to explain why one such hazard occurred and not eliminated. They should conclude that the first occurrence is used by the pipeline hardware to make a prediction and was thus unable to eliminate it.

At the end of the last three exercises the students will have noticed a significant improvement in the CPU performance as the hazards are eliminated. If we take the SF as an indicator of performance, there has been an improvement of 26% in the speed-up factor thus emphasizing the importance of the techniques developed to eliminate the pipeline hazards.



6.5 Tutorial 5: Loop Unrolling and Pipeline

The students of advanced architecture module learn about compilers and the ways in which the code generated is

Figure 11. Statistics with control hazards eliminated.

optimized which can also improve the CPU performance.

One such particular optimization is "loop unrolling" which essentially "flattens" a loop by eliminating the jump and compare instructions and reproducing the loop body's instructions multiple times which improves the pipelining of the optimized loop code.

The system simulator, of which the pipeline simulator is a part, incorporates a teaching compiler that can generate loop-optimized code. The students use the compiler to compile the same loop code as in Figure 3 but this time applying the loop unrolling optimization. The students note that the size of the code generated after compiling the loop source without loop unrolling optimization was 54 bytes. Also, when run, the CPU executes a total of 146 instructions (see Figures 9 to 11) of this code. Figure 12 shows part of the optimized codes's listing within the compiler's window. They note that in this case the size of the optimized code produced is 459 bytes which is much larger than the un-optimized code.

The students are asked to run the optimized code and observe the measurements recorded in the CPU pipeline window. Figure 13 shows the results of this run. In this case, the total number of instructions executed is 87 which is 40% less than the un-optimized code. Also, they observe that the CPI is now approaching 1 which approximates to one instruction executing in each clock cycle. The SI is now 4.81 which is an improvement of 94% indicating a very significant improvement in CPU performance.

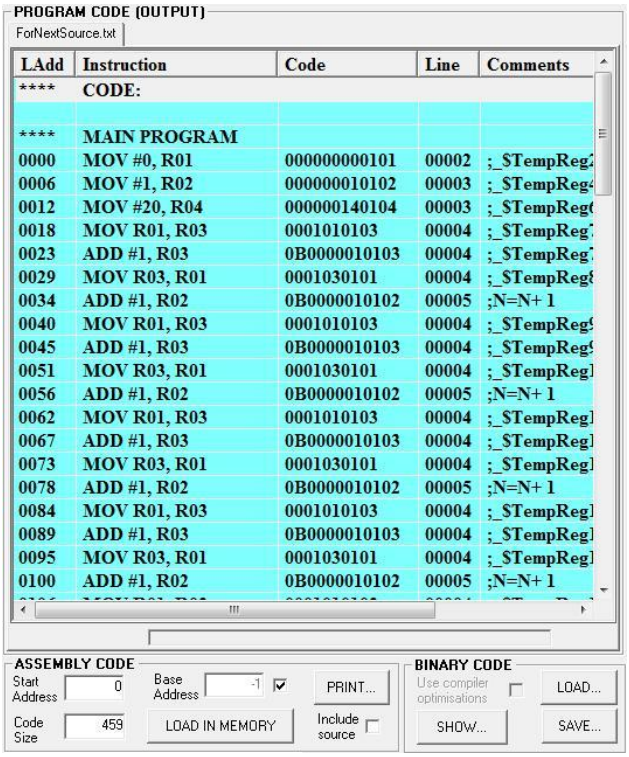


Figure 12. Loop unrolled code listing.

The students are expected to explain the significant drop in the number of loop instructions executed for the same program. They do this by observing that the optimized code does not include any jump and compare instructions. They thus learn that although some code optimizations may create larger executable code, the actual number of executed instructions can be much smaller than in the case of the un-optimized code.

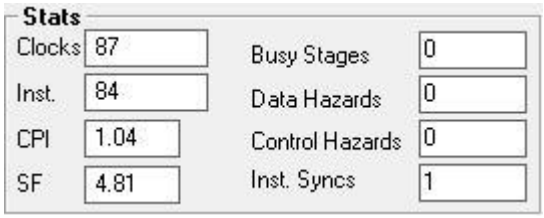


Figure 13. Loop unrolled code statistics.

7. EVALUATING THE SIMULATOR

In order to establish the degree of effectiveness of the simulations implemented in the integrated system simulator a preliminary evaluation has been conducted. The pipeline simulations have been used for the past two years to support teaching and learning both in the introductory and the advanced computer architecture teaching modules of a three-year honours undergraduate degree course. A similar study involving evaluation of a cache simulator has been described in [1].

The evaluation methodology included both qualitative and quantitative methods. The qualitative method was comprised of two opinion surveys of first and second year students. The quantitative method employed quasi-experimental process on non-random samples of student population. The evaluations were based on a relatively small sample of students involving 36 first year and 19 second year honours undergraduates studying for the computing degree. Table 1 shows the results of the first survey. This survey was carried out at the start of the evaluation period and shows that over 79% of the students are of the opinion that the simulations positively benefited them in their studies of the subject of computer architecture. Only a small number thought that the simulator was too complicated to use.

Table 1. Opinion survey 1 results.

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

The results of the second survey are shown in Table 2. This survey was conducted at the end of the evaluation period. Over 95% of the participating students thought that the simulator was a useful and effective educational tool.

Table 2. Opinion survey 2 results.

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 3 presents the results of the quantitative method in evaluating the pipeline simulator as part of the system simulator.

This method involved a small group of second year students studying advanced computer architecture module. The method involved multiple-choice pre-test questions, a set of practical tutorial exercises using the pipeline simulator (i.e. the intervention), and multiple-choice post-test questions. The questions and the exercises were all completed during the same session lasting three hours. There were seven questions in each test. All the students had theoretically covered the topics on which the exercises were based during the previous lectures.

Table 3. Pre and post test results.

Subject: Understanding Instruction Pipelining									
Test	Reply	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Avrg
Pre	16	93.8	75.0	56.3	87.5	43.8	81.3	37.5	67.9
Post	17	100.0	82.4	23.5	76.5	64.7	94.1	58.8	71.4

Table 3 shows the scores obtained by the students before and after the intervention. The scores are recorded as the percentage of the correct answers to each question. The average scores indicate that the post-test scores are marginally better than the pre-test scores. However, some of the individual scores indicate more significant improvement.

Overall, it seems the results of the quantitative evaluation are not strongly supportive of the results of the opinion surveys. Since this is a preliminary and rather a small-scale evaluation, we need to design and conduct a more detailed and larger-scale evaluation and analysis in the near future. One additional comment needs to be added here. The approach taken by the evaluating team of tutors was one of minimal support (akin to problem-based learning) before, during and after the practical sessions. It may be that this is not the best approach for the particular group of students who participated and needs to be re-considered for any future similar study.

We also wanted to survey the participants about their learning styles. The use of an educational teaching and learning resource like simulators is expected to be most effective for students with particular style of learning: mainly visual and hands-on learners. Table 4 shows the results of our survey.

Table 4. Survey of learning styles.

Learning Style	Number (%)
Visual Learner	31
Auditory Learner	5
Kinaesthetic learner	52
Reader Learner	7

Table 4 results show that 83% of the participants regarded themselves as visual and kinesthetic type (i.e. learning by doing) learners. This is a good indication that the simulator is most likely to be effective for those students who prefer learning by visual and hands-on approach of learning support.

8. CONCLUSIONS

This paper presented an educational tool for supporting the undergraduate lectures and the practical tutorial sessions in both the introductory and the advanced computer architecture teaching modules. The CPU pipeline simulator provides a highly interactive, configurable and visual interface which is designed to promote deep learning as well as team working. This tool has been successfully used by the author in his teaching modules on computer systems architecture as part of an integrated set of simulators. The recently conducted opinion surveys appear to support the assumptions that the simulator is both a useful and an effective educational resource. However, more work needs to be done in this area in order to both confirm and consolidate the findings of the preliminary study.

The integrated system simulator and example tutorials are available for download for educational purposes from the following link: www.teach-sim.com.

9. ACKNOWLEDGMENTS

The evaluation part of this work has been supported by funding from the Higher Education Academy (HEA), UK, for which we are very grateful.

10. REFERENCES

[1] Chalk, B. 2002. Evaluation of a Simulator to Support the Teaching of Computer Architecture. 3rd Annual LTSN-ICS Conference, Loughborough University, UK.

[2] Computing Curricula 2001. 2001. Computing Science, Final Report, December 15 2001. ACM and IEEE Computer Society joint report, USA.

[3] Computing 2007. 2007. Subject Benchmark Statement. The Quality Assurance Agency for Higher Education, UK, 2007.

[4] Mustafa, B. 2009. YASS: A System Simulator for Operating System and Computer Architecture Teaching and Learning. FISER'09 Conference, Famagusta, North Cyprus, Mar 22-24.

[5] Tanenbaum, S. A. 2006. Structured Computer Organization. Fifth Edition. Pearson/Prentice Hall, Upper Saddle River, NJ.

[6] Roger, L.U., Marizel, B., Josiel, E. 2004. DARC2: 2nd Generation DLX Architecture Simulator. ISCA2004, Munich, Germany, June 19.

[7] Zhang, Y., Adams, G. B. III. 1997. An Interactive, Visual Simulator for the DLX Pipeline. WCAE3 at HPCA3 , Feb 1997.

[8] Hennessy, J. L., Patterson, D. A. 2003. Computer Architecture: A Quantitative Approach. 3rd edition. Morgan Kaufmann publishers.

[9] Patterson, D. A., Hennessy, J. L. 2005. *Computer Organization and Design*. Third edition, Morgan Kaufmann publishers.