Name:	Iain S. Patterson
Project Title:	Online Teaching Aid for Computer Architecture: DLX Simjava Simulation
Supervisor:	Prof. R.N.Ibbett
Second Marker:	Dr. N.P.Topham
Session Chair:	Dr. Rob Pooley
Date and Time:	8th September 1996, 11:00
Venue:	JCMB, Room 2509

Extended Abstract

1 Principle Aims

The aim of the project was to develop an on-line teaching aid for computer architecture utilising the SIMJAVA simulation package. This is in an attempt to address the difficulty of teaching modern high performance architectures with their increased utilisation of concurrency techniques and instruction level parallelism. The simulation would then be accessible to world wide web users with a suitable JAVA compatible browser.

The project focused upon the Dlx architecture devised and presented by Hennessy and Patterson in their book, Computer Architecture: A Quantitative Approach. This architecture combines concurrent techniques such as pipelining, parallel functional units and a scoreboarding mechanism of control. These features are implemented in a reduced instruction set computer load store architecture. (RISC)

Principally the project hoped to abstract the architectural representation from that of the logical data flow level to a higher level data flow representation. This was then implemented as a working applet with animated simulation and various user defined elements to further enhance the simulation.

2 Introduction

2.1 DLX Architecture

- Load store RISC architecture
- Easily decoded instruction set
- Pipelined instruction execution
- 32 General purpose registers
- 32 Single precision floating point registers
- Word length of 32 bits
- 3 instruction formats

2.2 Java Language

The JAVA language tackles the key ideal of distributed programming capabilities in that it can transmit dynamic objects, i.e. dynamic self executing programs. JAVA benefits from being simple, secure, portable, object-oriented, robust, multi-threaded, architecture neutral and interpreted. For the purpose of this project the key stregth of JAVA was undoubtedly its ability to be incorporated into a working web applet with the potential for animated output.

2.3 Simjava

The SIMJAVA simulation package forms the basis for the program development as used in the project. It is a discrete event simulation tool capable of producing both stand alone simulations and animated applet simulations. These simulations are based around the following concepts.

- Collection of *Sim_entities* with defined behaviour
- Entities communicate by passing *Sim_event* objects
- Simulations can be animated extending *Sim_anim* using user defined icons and passing of events

3 System Design and Implementation Issues

In line with the objective of providing a high level abstraction of the Dlx architecture the following components were selected to be shown in the simulation. See Figure 1

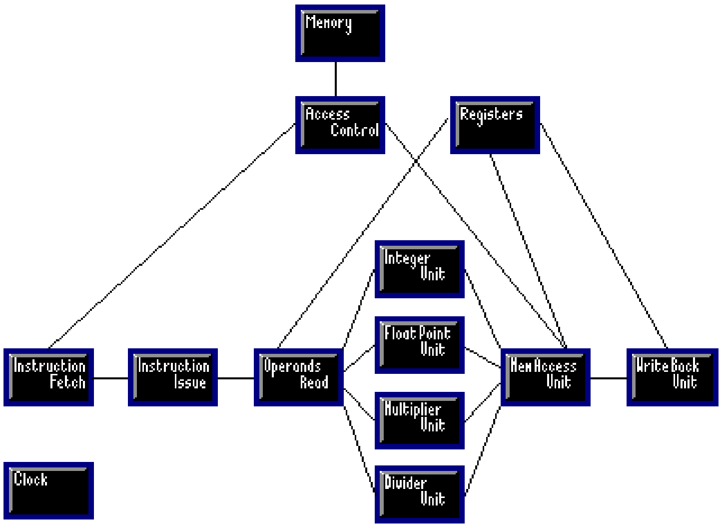

Figure 1: Simulation Entity Diagram

The instruction fetch unit controls the fetching of instructions, and the detection of branch and jump instructions. Once retrieved from memory the instruction is passed onto the instruction issue stage where a number of checks are performed. Structural and write after write (WAW) hazards are detected and relevant updates to the controlling scoreboard tables are made. Once issued the instruction is passed to the read operands stage of the pipeline. Here checks are performed which detect read after write hazards (RAW), and assuming that the instruction encounters no problems, the relevant operands are read and the instruction passses onto the appropriate functional unit.

The functional units are effectively in parallel and their operation is under the control of the scoreboard. There is provision for the option of changing the time necessary for execution of each unit though each has a default setting. In addition to performing calculation on the instruction the functional units must pass on the arithmetic output to the memory access stage without resource conflict. Priorities were assigned to the four units and forwarding of data is dependent upon this. The result then passes into the memory access stage which specifically acts upon load and store memory transfers and simply passes other instructions on to the write back unit. The write back unit then writes the value of the arithmetic operation back into the register file for future use.

The key differences between this layout and much of the documented literature occur in part due to the abstraction of the behaviour. Whilst at the logic level the memory access and write back are both fed from the output of the arithmetic unit, this becomes less feasible for implementation and understanding at the new level of representation. Similarly the inclusion of the instruction issue stage is in addition to the traditional five stage pipeline. However, this provides for clearer representation of the activities of the instruction decode and reading of the operands.

The system contains a novel approach to the representation of the clocked nature of the architecture. Rather than simply describe the individual entity behaviour in standard simulation time, the system incorporates a controlling clocking mechanism which acts to provide the animation with synchronous and clearly defined movement of information. This is a two phase mechanism with execute and forward phases and permits the expansion of the model to incorporate dead-reckoning for the system and its timing mechanism.

An additional feature of the system is the ability to change the memory access time. This then permits the user to view the impact that a slow memory might have on program execution as instruction fetch, load and store instructions request access. It also provides for more realistic simulation and potentially the future inclusion of a faster caching memory system. However, this was outwith the scope of the given projects aims.

An important area of consideration related to the understanding of the key concepts of the model. The student requires obvious visual keys to promote learning and these were centred around the use of text strings of assembly code. Hence the student is able to follow the relevant instruction without the continual need to refer to textual descriptions of the system. This highlights the concepts of structural and control flow hazards as they are encountered in the pipeline.

To further promote learning and visual clarity of the model, it was decided to focus the development of the system around the ability for user input of assembly code. This was tackled by adding a JAVA *TextArea* class and laying this within the SIMJAVA applet. However, this required the modification of the existing SIM_ANIM class and also the introduction of a new layout manager for the animation. In a similar fashion text was used to depict the various registers and their contents over the period of the simulation. These features can be seen in Figure 2 below.

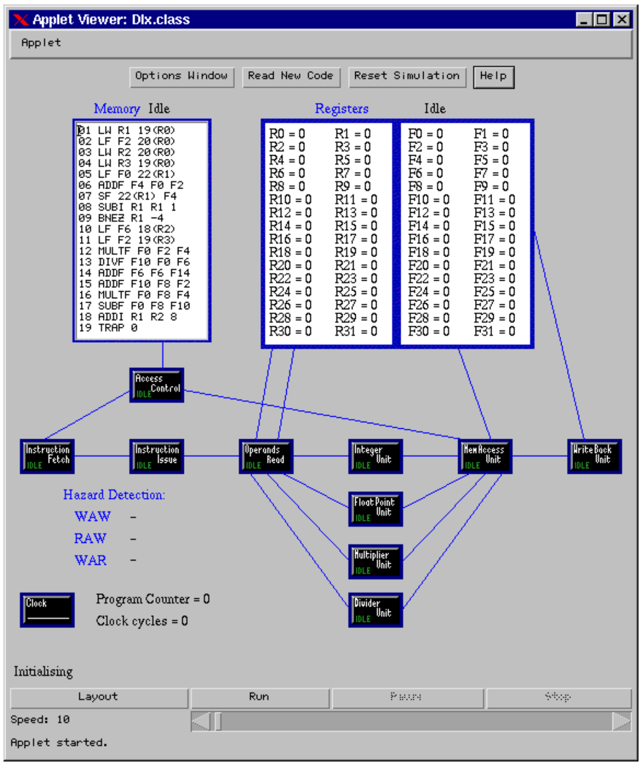

Figure 2: Simulation Web Page

4 Conclusions and Further Work

The user is provided with a clear model of the running of a piece of assembly code. This code, displayed as part of the simulation, also acts as a point of reference for the user to deduce the states of the program over time. The user input to the system provides increased direct involvement with the system and in this way attempts to improve the understanding gained by the user. However, not all desirable properties could be modelled in the available time frame. Extensions to the project would involve adding to the working instruction set, implementing further branch control policies, the impact of a cache memory system and also the ability to hierarchically view the simulation.

The working model incorporated a barrier synchronisation scheme to the simulation which permits the user to view more clearly the actions of the system and more importantly its changing state over time. This two phase two tiered system added to the complexity of the model and at times hampered the implementation of component behaviour. However, the representation provided by the clock control feature adds considerably to the understanding of the entire system.

As the development of the model progressed the difficulties associated with the debugging of a relatively complicated applet became evident. The activities of the threads posed a few problems given their concurrent nature. With the complicated logic being modeled, certain thread patterns at run time provided the system with subtle differences in the interpretation of the byte code. These were difficult to detect and reproduce, making the development slower than was initialy forecast.

The system is novel and provides a unique view of the architectural behaviour of the Dlx processor. While some of the original goals were not met, through pressures of time, the system does meet with the aim of providing a high level abstract model for the processing of assembly code in a web based applet. Indeed with additional improvements in the development tools for JAVA the simulation of such complex systems should become a most powerful mechanism for the teaching of computer architecture.