YAMS - Yet Another MIPS Simulator
Report 1

Adam Brinley Codd Ben Cotter James Milnes
Qian Qiao Sridhar Sarnobat Simon West

October 24, 2003

Contents

Chapter 1

Introduction

1.1 What is MIPS?

MIPS is a processor with a RISC (Reduced Instruction Set Chip) architecture. In the second year of Computing and JMC degrees at Imperial there is a laboratory exercise, as a part of the Compilers course, which involves translating a subset of Modula 2, called DEC(M2), into MIPS assembler code.

Currently, the college uses a MIPS simulator called SPIM to run this code and see if it works. The aim of this project is to write a more user friendly version of the simulator and which is partly customized for the needs of the course.

1.2 Program Outline

Our task involves writing two versions of the simulator. A text-only console version, and a graphical version. As a simulator, it's main task is to correctly simulate a given set of assembler instructions and produce any necessary output.

We have split the problem into several parts. To begin with, an instruction file will be made which will contain information on the MIPS instruction set. From this, we will construct a parser for the assembly language, and Java code to handle the instructions and system calls.

Secondly, using the parser, an assembler/loader will assemble the input file into machine code and load it into our memory manager. Then a processor will be started, which actually simulate the machine code. The final part is integration of the GUI into the three above steps.

Chapter 2

Minimum Specifications

2.1 Automatic Generation of Parser and Handlers

- The instruction file should be an XML file containing the instruction set needed for the second year Compiler lab exercise.
- It will be used to generate a grammar file suitable for use with Antlr. Antlr will use this file to, in turn, generate a parser for the assembler language input files.
- The XML file will also be used to generate an Instruction Handler, which will contain the code necessary to 'execute' each instruction.
- Finally, a System Call Handler will be generated with the same form as the Instruction Handler, but with code for a limited set of basic system calls (as used in the Compiler lab exercise).

2.2 Assembler/Loader

- The Assembler/Loader will run the parser over the input file and build an Abstract Syntax Tree (AST). If any syntax or structural errors are found in the file at this stage, they will be reported to the user and the program will not continue.
- If the AST was generated successfully, then the Assembler/Loader will walk through the AST and convert it to machine code, storing it into a 'virtual memory' structure.

- The Assembler/Loader will have the task of expanding pseudo instructions into a set of 'real' MIPS instructions as it produces the machine code.
- The Assembler/Loader will also be responsible for creating and initialising the memory, register and processor units. Once finished, it will instruct the processor to begin work.

2.3 Processor

- The processor will simulate all the cyclic functions of the MIPS processor, retrieving and executing instructions from memory. It will simulate storage of machine code as well as all the arithmetic functions required to carry out all possible instructions.
- It will be split into the following modules: Cycle Manager, ALU (Arithmetic and Logic Unit), Memory Manager, Register Manager, Instruction Decoder, Statistics Manager

2.3.1 Cycle Manager

The cycle manager will control each cycle of the processor, coordinating the actions of the other modules. Its overall purpose will be to carry out the following objectives:

- Retrieve the next instruction from memory as pointed to by the PC
- Decode instruction
- Execute the current instruction

It should be possible to issue commands to the cycle manager which cause the processor to pause after executing the next instruction, or to advance on instruction at a time, as this functionality will be required by the GUI.

2.3.2 ALU

The ALU unit is a component that isolates common operations that will be required often in coding MIPS instructions' semantics in the XML instruction file. These operations will include arithmetic and logical operations, such as shifting,

rotating, multiplying and sign extending. The operations provided will save coding time and lead to increased readability and consistency of the Java code when developing the XML file.

2.3.3 Memory Manager

The purpose of the memory manager is to provide fast access to all the memory locations available on the MIPS architecture. Both byte and word addressing must be provided to allow for flexible execution of instructions.

The MIPS architecture provides four segments;

- **Reserved** This section is located at the base of memory, starting at 0x00000000 and would normally be used for operating system code.
- **Text segment** Used to store the machine code for the program just assembled, starting above the reserved section.
- **Data segment** Used to store static and variable data used by the program. Begins at 0x10000000 and grows upwards.
- **Stack segment** This is located at the top of memory (0x7FFFFFFF) and grows downwards.

Note that this division will not be visible to the methods accessing the Memory Manager, since they will only use the read/write functions that deal with any address within memory. This separation within the Memory Manager may increase the efficiency of the processor, since it will divide the data stored in memory between the various segment structures and lower the possibility of increased access time when the memory gets full.

2.3.4 Register Manager

The purpose of the Register Manager is to encapsulate the MIPS registers and provide simple access procedures for them. The MIPS CPU has 32 general purpose 32-bit registers, named $0 through $31.

2.3.5 Instruction Decoder

Once an instruction has been retrieved from memory, it can be passed to the instruction decoder. It will work out firstly whether the retrieved data is a regular instruction or a system call and pass it onto the appropriate handler as necessary.

The Instruction and System Call Handlers perform very similar roles, and they have the responsibility of changing the state of the MIPS CPU to simulate the execution of an instruction.

The Handlers take the opcode of the instruction passed to it and perform the appropriate action stored for that instruction. This will, typically, manipulate registers using the Register Manager and memory through the Memory Manager. The action performed has been derived from the original XML file, which has Java code stored against instructions and their opcodes.

'Syscall' instructions are executed by the System Call Handler, all other instructions are executed by the Instruction handler. This is because the 'syscall' instruction has non-instruction set-specific semantics, so depending on the simulator and the values in argument registers, a different operation may occur. The 'syscall' instruction is thus often used as a means of communicating with the simulator's host system, to read input and print output.

2.3.6 Statistics Manager

The statistics manager will be a module devoted to keeping track of the current processor statistics, including individual register use, how much memory is being used by the current assembly program and number of clock cycles executed so far.

These statistics will be displayed at the end of the program run in the text-only version of the program, or dynamically as the program is running in the GUI version.

2.4 GUI

The program should have an easy to use GUI with features such as a constantly updating display of the values stored in memory and a graphical representation of the way instructions are being executed. This is the part of the program with the greatest scope for extension and so a wide range of possibilities will be discussed in the next section.

Chapter 3

Extensions

Since simulators either do or do not simulate correctly, there is little in the way of extensions that can be made to the core processor. However the GUI provides a lot of possibilities for enhancement.

- Support for the full MIPS instruction set, including floating point operations. This would involve making changes to the register manager to support the 32 floating point registers, $f0 to $f31
- Include descriptions of each instruction in the XML file, then an XSLT transformation could be applied to this file and help documents will then be produced.
- Extra statistics, such as keeping count of how many loops the code goes through during its execution, or how many times the code refers to specific instructions.
- Keep track of statistics on a time-based level. For every piece of data stored, there would be a timestamp associated with this value. While the implementation of the Statistics Manager would be significantly more complicated, he potential flexibility of the obtained data would be invaluable.
- In the GUI, produce graphical representations of the data provided by the Statistics Manager, for instance a histogram to represent the register use throughout the execution.
- Also in the GUI, produce charts representing how much of each segment of memory had been used during the execution in order to give the user an idea of how wasteful/efficient the program had been.

Chapter 4

Proposed Division of Work

Since the project divides easily into four sections, we have divided the work along these lines. However, the individual parts cannot easily be worked on concurrently, as, for instance, work on the assembler cannot begin until the parser has been built. Once interfaces have been defined and we have some minimal code working then it will be possible to start work on other sections, and this will be divided as follows:

Qian and Sridhar will work on the parser, James and Adam will work on the Assembler/Loader and Ben and Simon will start work on the processor.

Since the processor is the largest part of the program, once work is finished on the other two parts, these people will then work on other modules within the processor.

Once that is finished, we will all start work on the GUI, which we envisage will be a significant part of the project.

In addition, Adam is the Project Leader and Ben is Secretary.

Chapter 5

Timeline

Please see attached sheet for a breakdown of the project dates.

The first column contains deadlines set by the supervisors. The second column contains our own deadlines for planning, writing code and testing and the final column shows when work should be done on each report.