Universidade de Aveiro

Mestrado Integrado em Engenharia de Computadores e Telemática

Arquitectura de Computadores Avançada

Home group assignment 1
Implementing a forwarding and stall unit in a pipelined architecture

Academic year 2015/2016 — Nuno Lau/José Luís Azevedo

1. Introduction

The program **`MIPS_SystemC`** uses SystemC [3] to simulate the internal architecture of a MIPS processor. The simulated architecture, which includes a 5 stage pipeline, is similar to that shown in Figure 6.30 of the book Computer Organization & Design, H & P [1]. The current implementation does not include forwarding and stalls the pipeline to solve all data dependencies.

In this program, each MIPS component is described as a SystemC module (**`SC_MODULE`**). Some components are specified in terms of their behaviour by using methods that are invoked whenever any signal of its sensitivity list has its value changed (see, for instance, **`alu.h`** and **`alu.cpp`** files that specify the **`ALU`** module). Other components are specified in a hierarchical manner by instantiating other modules, interconnected through signals (see, for example, **`mips.h`** and **`mips.cpp`** files where the main components of the architecture are interconnected).

The source code of **`MIPS_SystemC`** is in the **`MIPS_SystemC.0.6.6.tgz`** archive available on the course web site. This code is the same as the one used in Lesson 3 lab assignment, still containing the error that must be corrected in the first question of that lesson (if you didn't yet solve Lesson 3, solve, at least, the first question of that assignment before starting this work).

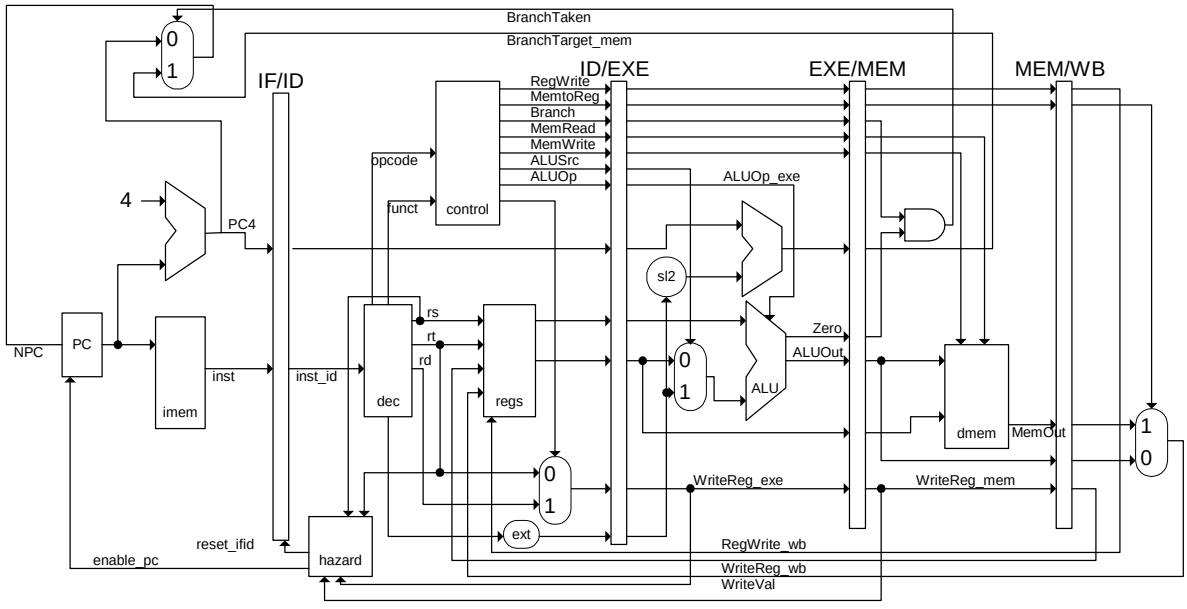

Figure 1

The simulator has a graphical interface, whose development was based on the Qt [4] class library, which allows controlling the assembly program execution, and the visualization, in various forms, of the processor state.

The **Arch** window enables viewing the architecture through a diagram similar to that shown in Figure 1. The values of the ports of some modules are shown on the figure. The **Modules** window allows viewing the state of some input and output ports of the main modules of the architecture. There are also windows for viewing the contents of the instruction memory, data memory and processor registers.

The file **refman.pdf** is the reference manual for the **MIPS_SystemC** simulator that thoroughly describes the functionality of each of the components of the simulated model.

2. Work description

This assignment is divided into the following cumulative tasks:

1. Suppose that it is necessary to increase the clock frequency of the processor, being this increase acceptable in all pipeline stages except in the **ID** stage, due to an unsuitable Register File access time. Therefore, to accomplish that, the **ID** stage could be divided in 2 stages (**ID1** and **ID2**) by inserting a new pipeline register (**ID1**/**ID2**) resulting in a processor with a 6 stage pipeline. With this approach, the output of the Register File is only available during the **ID2** stage, for a read operation that starts in **ID1**.

   Implement this new structure by adding the necessary changes to the **MIPS_SystemC** architecture provided for this assignment[1] (remember to change the hazard unit to fit this new pipeline architecture, so that data stalls are still inserted whenever needed). **Note**: the names of signals and ports of stages **IF**, **EXE**, **MEM** and **WB** should not be changed.

   Submit your source code file as "**ex1_*pn_nm1_nm2*.tgz**"[2].

2. Implement the following:

   2.1. Change the architecture resulting from task 1 so that the conditional jump instruction **beq** can be resolved during the **ID2** stage (instead of **MEM** stage), implementing the *delayed-branch* technique (i.e., the instruction that follows the **beq** is executed whether the branch is taken or not).

   2.2. Add the functionality that allows also the execution of the **bne**, **bgtz**, **blez**, **j** and **jr** instructions (all resolved in **ID2** stage), using the same *delayed-branch* technique.

   2.3. Modify the hazard unit to discard the instruction that wrongly entered the pipeline whenever the branch/jump is taken.

   Submit your source code file as "**ex2_*pn_nm1_nm2*.tgz**".

3. Take into consideration the changes introduced in tasks 1 and 2, and:

   3.1. Identify all types of forwarding that can occur in the architecture, using, wherever possible, a strategy where the **source of a forwarding action is a pipeline register and the destination is a pipeline stage**. Consider that a forwarding situation is

[1] For this and the following tasks, replace files "**regfile.h**" and "**regfile.cpp**" by those provided in "**regfile_home_assignment1.tgz**" (available at **elearning.ua.pt**). The "**regfile.h**" and "**regfile.cpp**" files implement the file register module needed for the 6-stage pipeline and, therefore, **should not be changed**.

[2] "**pn**" is the practical class id (p1, p2, or p3), "**nm1**" and "**nm2**" are to be replaced by the numbers of the group students (e.g. "**ex1_p1_32156_82345.tgz**").

different from another whenever there is any change in the source, in the destination or in the conditions under which the forwarding must be enabled.

3.2. For each type of forwarding identified, provide a code example, as well as the precise logic conditions under which that type of forwarding has to be enabled.

3.3. The result of this task should be clearly identified and explained in the assignment report.

4. Change the hazard detection unit (files **`hazard.h`** and **`hazard.cpp`**), so that stalls are never inserted (keep in mind that the branch/jump instructions should perform normally). After that, implement and test all types of forwarding situations identified in the previous task. For that:

   4.1. Specify in SystemC the multiplexer modules needed to implement the forwarding multiplexers for all required stages.

   4.2. Implement the "**`forward`**" module (add the files **`forward.h`** and **`forward.cpp`**) responsible for controlling the forwarding multiplexers.

   4.3. Add to the **`mips`** module (**`mips.h`** and **`mips.cpp`** files) the required changes to fully implement the forwarding technique.

   4.4. For testing purposes, use, among others, the code examples found in task 3, adapting them so that they do not need to stall the pipeline.

   Submit your source code file as "**`ex4_`*`pn_nm1_nm2`*`.tgz`**".

5. After having implemented and tested all types of forwarding identified in task 3, identify the conditions, and provide examples of instruction sequences that still cause the pipeline to stall. Change the hazard detection module (files **`hazard.h`** and **`hazard.cpp`**) in order to insert pipeline stalls in the identified situations.

   Submit your source code file as "**`ex5_`*`pn_nm1_nm2`*`.tgz`**". Highlight, in the assignment report, the instruction sequences that cause the pipeline to stall.

The work should be done, whenever possible, by a group of 2 students (groups of more than 2 students are not allowed). Each group must deliver:

- The source code of the program **`MIPS_SystemC`** for each of the tasks 1, 2, 4 and 5, in a "**`tgz`**" file with the names provided before. Please do not use "**`zip`**" or "**`rar`**" formats.
- A report, **only at the final due date**, that presents: a) the answers to the questions raised in this document, namely in task 3 and task 5; b) the strategy followed for the resolution of the various implementation tasks using, where convenient, schematic drawings that clarify the implemented solution; c) how the implemented solution was validated (which instruction sequences were used).

3. Important notes

1. DO NOT change directly the "**`Makefile`**" whenever you need to add a file to the project. For that you should 1) edit the "**`MIPS_SystemC.pro`**", adding to the compile list all the files you need and 2) run the "**`qmake MIPS_SystemC.pro`** " or "**`qmake-qt4 MIPS_SystemC.pro`** " command. This command will update the project **`Makefile`**.

2. Before generating the "**`tgz`**" archive file, execute the command "**`make clean`**", so that unnecessary object and executable files are deleted.
3. Generate the "**`tgz`**" file of the entire **`MIPS_SystemC.0.6.6`** directory, e.g.: "**`tar cvzf ex1_p1_32156_82345.tgz MIPS_SystemC.0.6.6/*`**"
4. Be sure that your code compiles without errors before submitting it. Be also sure that the compiled program does not fail execution, due to run-time errors. Code submitted with compilation or run-time errors will not be taken into consideration.
5. **The code submitted to solve a given task should not include any code for solving subsequent tasks. Hence, you should make a proper management of your code.** Hint: use a svn repository at code.ua platform to manage your source code versions.
6. During the development of this assignment you should follow an ethical conduct that prohibits plagiarism, in any form, as well as the participation of external elements in the assignment development. Any initiative that, judged by the teaching team, might be considered as a plagiarism situation will have real consequences on the student(s) evaluation and may lead to disciplinary sanctions.

4. Due dates:

- **November 2, 2015** (tasks 1 and 2)
- **November 16, 2015** (tasks 4, 5 and report)

Visualization changes

If necessary, during the development of the previous tasks, the visualization capabilities of **MIPS_SystemC** can be extended. Hence, to display the value of port "**port**" of the component "**component**" in the **Arch** window add the following lines to the constructor of **MIPSarchCanvas** of file **GUI/MIPSarch.cpp**:

```
portVal=new PortValItem(this,mips1.component->port, "port");
portVal->move(x,y);          // defines the position in the window
portVal->setColor(QColor("blue"));  // defines the color
portValVec.push_back(portVal); // inserts in the displayed list
```

To change the modules to display in the "**Modules**" window analyze the contents of the **GUI/MIPSmods.cpp**.

Bibliography

[1] *Computer Organization and Design: The Hardware/Software Interface, Second Edition*, D. A. Patterson and J. L. Hennessy, Morgan Kaufmann, 1998 (or a more recent edition)

[2] *Computer* Architecture, *A Quantitative Approach, Fifth Edition*, J. L. Hennessy and D. A. Patterson, Morgan Kaufmann, 2011

[3] *www.systemc.org*, SystemC web site

[4] *http://qt-project.org*, Qt development tools web site

Appendix

This appendix presents the meaning of some common errors that may occur when using SystemC, namely, compilation, execution and model specification.

Compilation errors

Using templates incorrectly: the use of templates is common in the construction of a SystemC model; however, there should be some caution in the way they are declared:

```
mux< sc_uint<32> > *mPC; //correct
mux< sc_uint<32>>  *mPC; //wrong (should have a space between >>)
```

Unsing undefined signals, ports or modules: all signals, ports or modules must be defined before being used. For example, an attempt to use port **`diin`** (not defined) of the **`alu`** module can lead to the following error message:

```
mips.cpp:138: no matching function for call to
              alu::diin(sc_signal<sc_dt::sc_uint<32> >&)
```

Type mismatch between a signal and a port: the connection between a signal and a port is only possible if they are of compatible types. Failure to comply with this rule leads to error messages like:

```
mips.cpp:129: no match for call to `(sc_out<sc_dt::sc_uint<32> >)
             (sc_signal<sc_dt::sc_uint<3> >&)
```

Execution errors

During the execution of the model, SystemC runs some checks looking for model consistency. When an error is detected, SystemC writes an error message and terminates the program. The most common error situations are listed below.

No signal connected to a given port: all ports of a module must be connected to a signal. The following message indicates that the third port (in order of declaration) of the **`MIPS.alu`** module is not connected to any signal:

```
Error: (E109) complete binding failed: port not bound:
              port 'MIPS.alu.port_3' (sc_out)
```

More than one signal connected to the same port: only one signal can be connected to a given port; attempts to connect more than one signal at the same port lead to messages like:

```
Error: (E107) bind interface to port failed: maximum reached:
              port 'MIPS.alu.port_3' (sc_out)
```

Specification errors

Sensitivity list incomplete: when changing the functionality of a module described in a behavioral way (**`SC_METHOD`**), especially when new input signals are added, the sensitivity list should always be checked.