Universidade de Aveiro

Mestrado Integrado em Engenharia de Computadores e Telemática

Arquitectura de Computadores Avançada

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

Academic year 2013/2014 | Nuno Lau/José Luís Azevedo

The program **`MIPS_SystemC`** uses SystemC [3] to simulate the internal architecture of a MIPS. 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.0.6.5.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.

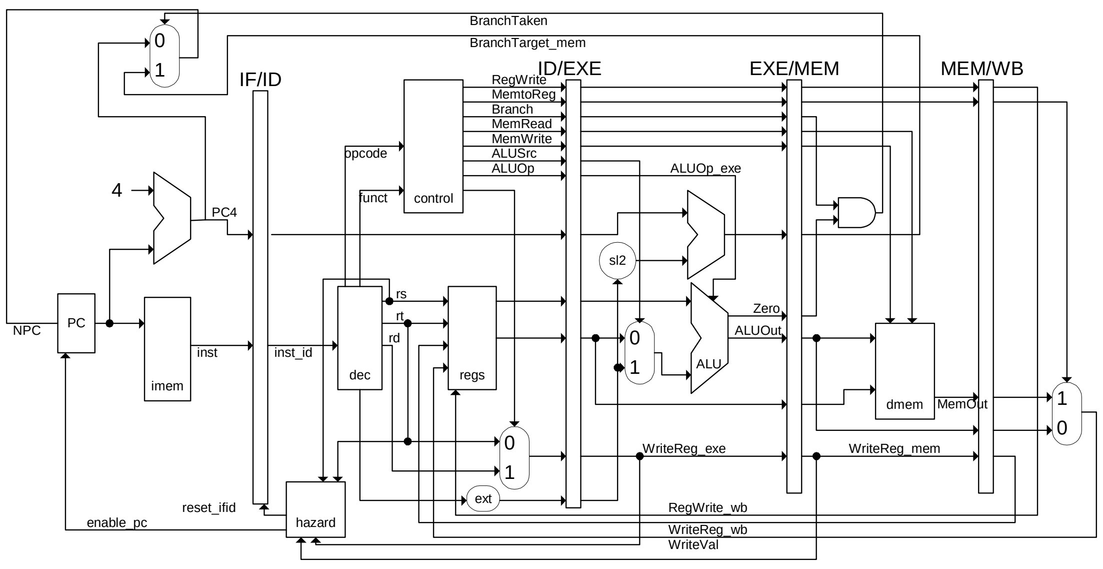

Figure 1

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.

This assignment is divided into the following tasks:

1. Change the **`MIPS_SystemC`** architecture so that conditional jumps (**`beq`** instruction) can be resolved during the **`ID`** stage (instead of **`MEM`** stage).

2. Change the **`MIPS_SystemC`** architecture to support the execution of unconditional jumps, namely instructions **`j`** and **`jr`** (jump indirect on register), both resolved in the **`ID`** pipeline stage.

3. Suppose that it is necessary to increase the clock frequency of the processor, being this increase acceptable in all pipeline stages except in the **`MEM`** stage, due to a high memory access time. Therefore, to accomplish that, the **`MEM`** stage should be divided in 2 phases (**`MEM1`** and **`MEM2`**) by inserting a new pipeline register (**`MEM1`**/**`MEM2`**) resulting in a processor with a 6 stage pipeline.

   Replace the **`dmem.cpp`** and **`dmem.h`** files with the version that is provided in the course site for this exercise. Implement this new structure (datapath with a 6 stage pipeline) by adding the necessary changes to the existing simulator. The names of signals and ports of stages **`IF`**, **`ID`**, **`EXE`**, and **`WB`** should not be changed.

4. Identify all types of forwarding that can occur in the architecture, taking into consideration the changes introduced in tasks 1, 2 and 3, using, wherever possible, a strategy where the **source of a forwarding action is a pipeline register and the destination is a pipeline stage**. For each type of forwarding identified, an example, as well as the precise conditions under which forwarding has to be enabled, should be provided. Consider that a forwarding situation is 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.

5. Implement all types of forwarding identified in the previous task.

   5.1. in order to test the forwarding situations in an easier way, change the provided hazard detection unit (files **`hazard.h`** and **`hazard.cpp`**) supposing that it is never necessary to stall the pipeline. Even if some instructions will not execute correctly, you will be able to create forwarding situations easily;

   5.2. specify in SystemC the "**`forward`**" module responsible for controlling the forwarding multiplexers;

   5.3. specify in SystemC the forwarding multiplexers;

5.4. change the **mips** module (**mips.h** and **mips.cpp** files) so that it makes the forwarding in a correct way (files **reg_if_id.h**, **reg_id_exe.h**, **reg_exe_mem.h** and **reg_mem_wb.h**); if necessary add signals to the pipeline registers and to the **mips** module.

6. Considering that all types of forwarding identified in task 2 were implemented, identify the conditions, and give examples of situations that still causes the pipeline to stall. Change the hazard detection and stall generation module to insert pipeline stalls in the identified situations. Add this module in the **MIPS_SystemC** program, implementing the necessary 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**.

The work should be done, if 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 with the corresponding changes to tasks 1, 2, 3, 5 and 6;
- a report that presents: a) the answers to the questions raised in this document; b) briefly presents the strategy followed for the resolution of the various implementation tasks.

Due date: November, 11, 2013

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 >>)
```

Using 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 have 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 to 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

Incomplete sensitivity list: 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.