Computer Architecture 2012/2013

Assignment 2c

Date: September 25th, 2012

Deadline: October 29th 2012, 23:59

Contents

1 Overview

Reminder: The purpose of assignment series 2 is to implement the MIPS ISA in MGSim, so as to be able to run real MIPS code compiled using GCC and the GNU Binary utilities (assembler+linker). Assignment series 2 will be spread over multiple weeks, and split into individual assignments 2a, 2b, etc.

The goal of assignment 2c is to:

- extend the work started in assignment 2a & 2b
- complete the decode logic;
- complete the execute logic;
- test the results with simple programs.

2 Instructions

- For this assignment, you can work in groups of 2.
- Read this entire document before you start.
- Your must submit a compressed tarball[1], named after your last name and student ID, containing:
  - the files that you have produced during the assigment.
  - a file `report.rst` containing your write ups to open questions using reStructured Text. This must also contain your full name and student ID. Ensure that `report.rst` is valid by using `rst2html`.
- Your submission must be sent by e-mail before the deadline, at the e-mail address given by the assistants. Do not send your submission to the mailing list!

3 Prerequisites

You will need the following:

- the Alpha an MIPS cross-utilities and cross-compilers, and the MGSim simulator compiled for Alpha, as per assignment 1/2a/2b.
- the MGSim source code and development environment, as per assignment 2a/2b.
- a copy of the following files, which should accompany this document:

File	Description
mipsel-cc	Script to compile/assemble/link MIPS code.
minisim.ini	Configuration file for MGsim.
arith.c	Example micro-program.

Note

The files `alpha-cc`, `mipsel-cc` and `minisim.ini` are different from assignment 1.

4 Branches

To effect a branch in the execute stage, the code in `ExecuteInstruction` should look like this:

```
...
COMMIT { m_output.pc = <new PC value, target of branch>; }
...
return PIPE_FLUSH;
```

[1]A compressed tarball is created with `tar -czf xxxx.tgz ....`

In other words, whereas "normal" instructions must terminate `ExecuteInstruction` with `PIPE_CONTINUE`, branches must instead return `PIPE_FLUSH`. Therefore, to implement branches you first need to introduce a new conditional and have different paths to `return` depending on the operation's function code.

From this point you can perform the following:

1) implement unconditional branches: `j`, `jr`, `jal`
2) implement conditional branches: `beq`, `bne`.

> **Note**
>
> `bnez` and `beqz` are simply `bne` and `beq` with one operand set to $0, which is the special zero register.

5 Memory operations

MGSim already implements a memory stage and already defines the appropriate buffers in the execute stage's output latch. However, the execute stage must configure these buffer to "activate" the memory stage and make memory operations actually happen.

There are 5 buffers:

- `size`: the size of the memory operation in bytes (eg. 4 for `lw`, 2 for `lh`, 1 for `lb`);
- `address`: the memory address;
- `sign_extend`: whether to sign-extend the value to the full register width (`true/false`, cf the difference between `lh` and `lhu`);
- `Rc`: for load instructions, the index of the register where loads must place the result (0-31, set with `MAKE_REGADDR` in decode stage as per assignment 2a);
- `Rcv`: for store instructions, the value to store in memory.

The memory stage is activated as soon as `size` is set to a non-zero value. Then to distinguish between loads and stores:

- if `Rcv.m_state` is `RST_FULL`, then the operation is a store;
- if `Rcv.m_state` is `RST_INVALID`, then the operation is a load.

For example, a 4-byte memory load can be implemented as follows in the execute stage:

```
COMMIT {
  m_output.address = Rbv + displacement;
  m_output.size = 4;
  m_output.Rcv.m_state = RST_INVALID;
  // NB: m_output.Rc is already set by decode stage
 }
```

For example, a 2-byte memory store with sign extend can be implemented as follows:

```
COMMIT {
  m_output.address = Rbv + displacement;
  m_output.size = 2;
  m_output.sign_extend = true;
  m_output.Rcv = m_input.Rav; // 1st input register value
 }
```

Using this information, implement the remaining memory operations.

6 Overall testing

Check that your resulting simulation works with the example programs from assignment 1: for each program, report the number of simulation cycles needed to complete the execution with the MIPS ISA.

7 Summary of submission contents

Your final submission archive should contain the following files:

- `report.rst` (your report with explanations);
- `mgsim.patch` (your patch file generated with `git diff origin/mgsim`);
- any micro-programs and testing scripts you have created to support your work.

8 Grading

Assignment 2b and 2c will be graded together. You will be evaluated as follows:

- whether you have implemented unconditional branches (.5pt);
- whether you have implemented conditional branches (.5pt);
- whether you have implemented loads/stores (1pt);
- whether your code is properly structured, indented, documented, either as comments or in the separate report (2pt, overlaps with assigment 2b);
- how you can prove or illustrate that all your instructions work using the test code from assignment 1 (2pt, overlaps with assignment 2b).