Computer Architecture 2012/2013
Assignment 2a

Date: September 25th, 2012
Deadline: October 1st 2012, 23:59

Contents

1 Overview **1**

2 Instructions **2**

3 Prerequisites **2**

4 Set up your development environment **3**
4.1 Set up your shell environment 3
4.2 Set up your copy of the MGSim source code 3

5 Using micro-programs for testing **5**
5.1 Testing "add" on the Alpha platform 5
5.2 Testing "add" on your MIPS platform 6

6 Implementing one MIPS instruction **6**
6.1 Making changes to the MIPS ISA implementation 6
6.2 Testing the MIPS code 7
6.3 Producing a patch file 7
6.4 Describing the architecture 7

7 Implementing another MIPS instruction **8**

8 Summary of submission contents **8**

9 Grading **8**

10 Next steps **9**

1 Overview

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 2a is to:

- set up your development environment for the rest of the assignment;
- learn how to inspect the simulation environment during program execution;
- familiarize yourself with the MGSim core pipeline;
- implement your first MIPS instructions.

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.
- a copy of the following files, which should accompany this document:

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

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

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

4 Set up your development environment

4.1 Set up your shell environment

1. Log in to the Unix shell in your work environment.
2. For the rest of the assignment(s) you will need two directories: an "installation" directory and a "source" directory, both in a private location, for example somewhere in your home directory. We suggest:

   - use `$HOME/opt` (or `~/opt`) as "installation" directory;
   - use `$HOME/src` (or `~/src`) as "source" directory.

   (you are free to select your own, but they must be different from each other)
3. Set up the environment variables `CASRC` and `CAINST` to point to the full path of your directories. Ensure the configuration persists by setting up these environment variables in your shell's configuration file. (Ask the assistants for help on this if you're unsure).
4. Insert `$CAINST/bin` in your `PATH` environment. Again, ensure the modified `PATH` is preserved in your shell's configuration file.
5. Insert `$CAINST/share/man` in your `MANPATH` environment. Again, ensure the modified `MANPATH` is preserved in your shell's configuration file.

4.2 Set up your copy of the MGSim source code

1. Clone the MGSim repository in `$CASRC/mgsim`:

   ```
   cd $CASRC
   git clone https://github.com/knz/mgsim.git
   cd mgsim
   ```
2. Ensure you are working on the "mips" branch:

   ```
   git checkout mips
   ```
3. Run the following, to initialize your working copy:

   ```
   ./bootstrap
   ```

   This generates the `configure` file. You only need to run `bootstrap` the first time you obtain the MGSim source, ie when you use step #1 above.

Note

If you use your own computer, you will need to ensure that Autoconf, Automake, SDL (with development files) and Docutils are installed before running `bootstrap`.

- Ubuntu/Debian: `apt-get install autoconf automake libsdl1.2-dev python-docutils`
- MacPorts: `port install autoconf automake py27-docutils libsdl` (or `libsdl-devel`)

4. Create a build directory, separate from the source directory. For example:

   ```
   cd $CASRC
   mkdir mgsim-build
   ```

5. In the build directory, run the `configure` script:

   ```
   $CASRC/mgsim/configure --target=mipsel --prefix=$CAINST \
         CXXFLAGS="-ggdb3 -O1" \
         --disable-abort-on-trace-failure \
         --disable-verbose-trace-checks
   ```

   This step generates the `Makefile` necessary to drive the build process. This step is only necessary once per build directory.

   Note

   If you use your own computer, you will need to ensure you have libev (http://software.schmorp.de/pkg/libev.html), argp (standard in GNU C Library, otherwise argp-standalone http://www.lysator.liu.se/~nisse/misc/), GNU Make, and Python 2.x (http://www.python.org/) installed, with development files. If possible ensure that SDL is also installed: http://www.libsdl.org/ .

   We customize the work of `configure` as follows:

   - MGSim can support Alpha, SPARC and now MIPS cores. We use the argument `--target=mipsel` to prepare the build for the MIPS core model.
   - `--prefix` selects where the compiled files are installed.
   - `CXXFLAGS"-ggdb3 -O1"` indicates that the simulator itself must be built so that it is easy to debug later.
   - `--disable-abort-on-trace-failure` and `--disable-verbose-trace-checks` disable an advanced checking mechanism in MGSim that is otherwise not relevant for this course.

6. Build with:

   ```
   make
   ```

This compiles the MGSim source code into your build directory and produces the `mgsim` executable and documentation.

7. Install with:

   ```
   make install
   ```

   This installs the MGSim executable file, reference configuration and documentation into `$CAINST`.

8. Check that your installation was successful:

   ```
   mipsel-mgsim --version
   man mipsel-mgsimdoc
   ```

5 Using micro-programs for testing

The execution of a regular C program starts with its function `main`, which is run as if it was "called by the environment". The C language is said to be "hosted": there is some *operating software* running on the computer around the C program, in charge of loading the C program into memory, setting up its standard library, and calling the `main` function.

This must be true even in a reduced environment like an architecture simulator. Even without a C library or process management, the `main` function must be "called" from somewhere, and will eventually "return" to that somewhere upon completion.

This is the role of the assembly source called `minicrt` in assignment 1: The minicrt initializes a stack memory, then calls `main`. When `main` returns control to the minicrt, it then terminates the simulation.

For assignment 2, we wish to start in a "simplified" environment where only very few MIPS instructions are used. This means that we do not want to use the minicrt, where there are at least 3-4 different kinds of instructions required before control is transferred to `main`.

To achieve this, we will use *micro-programs* that do not require a minicrt, but also do not use C's idea of starting execution with `main`. Instead, our micro-programs define a single function called `_start`: MGSim will start execution at this location in memory directly.

5.1 Testing "add" on the Alpha platform

1. Look at the source code of `add-alpha.s`. Predict the value of the registers after each instruction has executed.

2. compile `add-alpha.s` to the executable `add-alpha.bin`, as per the techniques from assignment 1. Notice that `minicrt-alpha.s` is not needed here.

3. Use `objdump` to obtain the address of the entry point. Then run MGSim for Alpha (from assignment 1) in interactive mode with the following command-line parameters:

   ```
   .../mtalpha-mgsim -c minisim.ini \
      add-alpha.bin -i -R2 42 -R3 24 -R4 69 -R5 51 -R6 99
   ```

(You can use your own values for the various `-R` parameters)

Then set up a break point at the entry point, then start execution.

4. At the prompt, run the following command:

   ```
   read cpu0.registers
   ```

   This prints the contents of the register file. Check that the values conform to the `-R` parameters you have specified above.

   Note

   You can only inspect the registers after the entry point is reached by a breakpoint. If you attempt to inspect the registers at simulation cycle 0, they will appear empty.

5. Run the following command:

   ```
   trace pipe
   ```

   This enables reporting the events in the pipeline.

6. Run the following command six times, one after another:

   ```
   step
   ```

   Explain what you see at each step in your own words (in the separate report).

7. Run `step` further. How many times can you run it? Why?

8. Observe the contents of the register file at the end of the execution. Is it compatible with your predictions at step #1? Why?

5.2 Testing “add” on your MIPS platform

1. Look at the file `add-mips.s`. Predict the register values after each instruction.

2. Perform steps #2-#8 from the previous question, substituting “mips” for “alpha” where necessary. How many steps can you run?

6 Implementing one MIPS instruction

6.1 Making changes to the MIPS ISA implementation

1. Open the file `$CASRC/mgsim/arch/proc/ISA.mips.cpp` in your favorite source code editor.

2. Add the following code in `DecodeInstruction`:

```
RegIndex Ra    = (instr >> 21) & 0x1f;
RegIndex Rb    = (instr >> 16) & 0x1f;
RegIndex Rc    = (instr >> 11) & 0x1f;

COMMIT {
m_output.Ra = MAKE_REGADDR(RT_INTEGER, Ra);
m_output.Rb = MAKE_REGADDR(RT_INTEGER, Rb);
m_output.Rc = MAKE_REGADDR(RT_INTEGER, Rc);
}
```

Explain in your report what the values 11, 16, 21 and 0x1f mean and where they come from.

3. Add the following code in `ExecuteInstruction`:

```
uint32_t Rav = m_input.Rav.m_integer.get(m_input.Rav.m_size);
uint32_t Rbv = m_input.Rbv.m_integer.get(m_input.Rbv.m_size);

COMMIT{
m_output.Rcv.m_state = RST_FULL;
m_output.Rcv.m_integer = Rav + Rbv;
}
```

4. Run `make` and `make install` again in your build directory.

Note
Any C++ code in `DecodeInstruction` or `ExecuteInstruction` that *modifies* the output latch should be enclosed between `COMMIT{` and `}`.

6.2 Testing the MIPS code

1. Test the `add` program as per Testing "add" on your MIPS platform above, using the new simulator with your changes.

2. Compare the results with your expectations. In particular, explain how many times you can "step" through the execution and how this differs from the Alpha platform.

6.3 Producing a patch file

Run the command `git diff origin/mips` in your MGSim source tree. Save the output to `mips-add.patch`.

6.4 Describing the architecture

In your report, list the latch buffers that you have used and their width in bits. Make a simple diagram that illustrates which components are involved so far in your processor model.

7 Implementing another MIPS instruction

1. Copy `add-alpha.s` and `add-mips.s` to `sub-alpha.s` and `sub-mips.s`. Modify the latter two, to mix `sub` instructions with `add`, so that the resulting program uses these two types of instructions.
2. Extend your implementation to support `subu` next to `addu`. For this, use the buffer `function` already present in the decode-read latch (see `ISA.mips.h`):
   - in `DecodeInstruction` use `m_output.function = ....`
   - in `ExecuteInstruction` use `if (m_input.function ...)`
3. Test your changes. Copy-paste in your report the part of the output from MGSim that proves that your modification works.
4. Produce another patch for these changes to `mips-sub.patch`.
5. In your report, list the additional buffers that you have used and their width in bits. Make a simple diagram that illustrates which components are involved so far in your processor model.

8 Summary of submission contents

Your final submission archive should contain the following files:

```
report.rst
add-alpha.bin
add-mips.bin
sub-alpha.s
sub-mips.s
sub-alpha.bin
sub-mips.bin
mips-add.patch
mips-sub.patch
```

9 Grading

You will be evaluated as follows:

- whether you have investigated + explained the behavior of "add" on the Alpha platform (10% of grade);
- whether you have investigated + explained the behavior of "add" on the MIPS platform prior to changes (10% of grade);
- whether you have implemented "add" properly for MIPS, explained the decoding constants, produced the appropriate patch file and described the latches properly (30% of grade);
- whether you have implemented "add" and "sub" properly for MIPS side-by-side, produced the appropriate illustration of behavior and patch file, and described the latches and components properly (50% of grade).

10 Next steps

If you have remaining time available, you can move forward and start implementing the additional ALU instructions that you have identified in assignment 1, question 7. Use `arith.c` as a test program (you can modify it as needed). This will be graded later in assignment 2b.