Computer Architecture 2012/2013
Assignment 1

Date: September 4th, 2012
Deadline: September 24th 2012, 23:59

1 Instructions

- For this assignments, you must work individually. However, you are free to ask questions on the mailing list.
- Your must be submit a compressed tarball[1], named after your last name and student ID, containing:
  - your annotated and optimized assembly source files;
  - one file `report.rst` containing your write ups to open questions using reStructured Text. This must also contain your full name and student ID.
- 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!

2 Prerequisites

You will need the following:

- the Alpha an MIPS cross-utilities and cross-compilers; this should be prepared for you before the assignment starts.
- the MGSim simulator compiled for Alpha;
- 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.
minicrt-alpha.s	Boot code for the Alpha environment.
minicrt-mips.s	Boot codeq for the MIPS environment.
minisim.ini	Configuration file for MGsim.
hello.c	Example program.

... continued on next page

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

File	Description
helloworld.c	Example program.
fibo.c	Example program.
comp.c	Example program.
sumdemo.c	Example program.
roman.c	Ancillary file for fibo.c, comp.c and sumdemo.c.
sum.c	Ancillary file for sumdemo.c.

3 Getting acquainted with the environment

1. Open the program `hello.c` in a text editor. Analyze what it does. Picture in your mind its expected output.
2. Compile the program down to assembly:

   ```
   ./alpha-cc -S hello.c -o hello.s
   ```

3. Assemble the code:

   ```
   ./alpha-cc -c hello.s -o hello.o
   ```

4. Assemble the minicrt:

   ```
   ./alpha-cc -c minicrt-alpha.s -o minicrt-alpha.o
   ```

5. Link both objects to form an executable file

   ```
   ./alpha-cc hello.o minicrt-alpha.o -o hello.bin
   ```

> **Note**
> You can simplify steps 2-5 as follows:
>
> ```
> ./alpha-cc hello.c minicrt-alpha.s -o hello.bin
> ```

6. Run this program:

   ```
   mgsim -c minisim.ini hello.bin
   ```

   Check that the program indeed prints the expected output.

7. Repeat steps #1-#6 using theother programs `helloworld.c`, `fibo.c`, `comp.c` and `sumdemo.c`. Note that some files must be linked with others (e.g. `sumdemo` with `sum` and `roman`).

4 Annotating code and understanding optimizations

1. Generate the unoptimized Alpha assembly code for `comp.c`:

   ```
   ./alpha-cc -O0 -S comp.c
   ```

2. Rename the assembly file to `comp-0-alpha.s`. Add code comments to this file that explain how the code works. Explain how the different registers are used and describe the structure of the assembly program. Relate this structure to the original C code. Your comments should explain the workings of the program and not the (trivial) meaning of the instructions themselves. (eg we know that `lda $1,104($31)` loads the value 104 into register $1, so don't tell us)

3. Compile the same source file again using `-O1` to `comp-1-alpha.s`. Compare `comp-1-alpha.s` to `comp-0-alpha.s`. Add code comments to this file that relate the code back to the C source. Explain in your own words (separately) what the compiler did to optimize.

4. Compile again using `-O2` to `comp-2-alpha.s`. Compare `comp-2-alpha.s` to `comp-1-alpha.s`. Again, related the assembly to the original program, then explain in your own words the optimizations.

5. Calculate by hand, for each assembly source:

   a. the number of static instructions in the code.

   b. the number of dynamic instructions executed at run-time.

   c. the number of memory accesses performed during execution.

   d. assume that the execution time is proportional to the dynamic instruction count. Estimate the speedup at each optimization stage.

   Note: exclude the execution of `print_roman` in your analysis.

6. MGSim can report the number of instructions executed and the number of load and store instructions performed at the end of the execution of a program. Use MGSim to verify your estimations for #5b, #5c, #5d.

> **Tip**
>
> The instructions in `print_roman` will "pollute" the results reported by MGSim. Determine a way to exclude this overhead from the verification.

> **Note**
>
> Place the annotated files `comp-0-alpha.s`, `comp-1-alpha.s`, `comp-2-alpha.s` in your submission archive. Place your separate answers to #3, #4 and #5 in your `report.rst`.

5 Manual optimization of assembly

1. Generate the unoptimized MIPS assembly code for `comp.c`:

   ```
   ./mipsel-cc -O0 -S comp.c
   ```

2. Again, rename the assembly file to `comp-0-mips.s`. Add code comments to this file that explain how the code works. Explain how the different registers are used and describe the structure of the assembly program. Relate this structure to the original C code.

3. Create an approximate correspondance table between the MIPS and Alpha instruction sets, for the instructions used in this program (ie. not for the complete MIPS & Alpha ISA).

4. Copy `comp-0-mips.s` to `comp-opt-mips.s`. Optimize the code in `comp-opt-mips.s` by hand and explain your optimizations. **Do not use the MIPS compiler's optimization flags!** However, you can inspire yourself from the Alpha techniques you learned from the compiler optimizations in the previous section. Ensure that your optimized assembly code is recognized by the assembler, by compiling it to binary code.

   Of course the output of the program should be identical; as well as the contents of arrays A and B at the end of the execution.

   Any idea that may contribute to a performance increase may be used (everything about the structure of the program may be changed), provided that you explain in detail what you did. In other words: give thorough comments on what you did!

5. Calculate by hand the static and dynamic instruction counts, and the number of memory accesses, for both the unoptimized version and after your optimizations.

> **Note**
>
> Place the annotated file `comp-0-mips.s` and optimized code `comp-opt-mips.s` in your submission archive. Place your separate answers to #3 and #5 in your `report.rst`.

> **Tip**
>
> You cannot yet test (run) your optimized MIPS code in MGSim. However, preserve your optimized assembly source! In the next assignments, you will extend MGSim with the MIPS ISA; you will then use this week's code to test your future simulator.

6 Bonus exercise: relocations

1. Compile the file `helloworld.s` to both Alpha and MIPS assembly source (`hw-alpha.s` and `hw-mips.s`) with optimizations enabled (`-O2`).

2. Compile the file `helloworld.c` also to final Alpha and MIPS executables (`hw-alpha.bin` and `hw-mips.bin`) with the same optimization level.

3. Using the commands `alpha-linux-gnu-objdump -d` and `mipsel-linux-gnu-objdump -d`, disassemble the resulting executables to `hw-alpharev.s` and `hw-mipsrev.s`.

4. Compare the reverse assembly code to the code generated at step #1. Observe in particular how the address of the string is loaded into a register; note how the assembler has transformed the code to use an additional computation based on a register called GP.

5. Do additional research (Internet, binutils documentation, etc...) using the keyword "relocation"; then explain in your own words how this mechanism works. You can use the assembly source from `comp.c` as additional example to illustrate your explanation.

> **Note**
> Place your explanation in your `report.rst`.

7 Preparing for the next assignments

1. Compile to MIPS binary code for all the test programs provided.

2. Using the command `mipsel-linux-gnu-objdump -d`, inventarize all the different MIPS instruction names that are effectively used by the test programs.

> **Note**
> Place the files `hello-mips.bin`, `helloworld-mips.bin`, `fibo-mips.bin`, `comp-mips.bin` and `sumdemo-mips.bin` in your submission archive. Place your inventary for #2 in your `report.rst`.

> **Tip**
> This list of instructions defines the minimum set of ISA instructions your future assignments will need to emulate. In other words, as soon as your own MIPS simulator supports these instructions, you will be able to run these programs.

8 Summary of submission contents

Your final submission archive should contain the following files:

```
report.rst
comp-0-alpha.s
comp-1-alpha.s
comp-2-alpha.s
comp-0-mips.s
comp-opt-mips.s
hello-mips.bin
helloworld-mips.bin
fibo-mips.bin
comp-mips.bin
sumdemo-mips.bin
```