Assignment 1: Optimization of Assembly

September 2006

In this assignment you will get acquainted with *asssembly language*. Using assembly you have direct access to the instruction set of the processor.

In this assignment, you will optimize a small assembly program by hand.

The following C program is given:

```
int A[20], B[20], C, i;

main()
{
  for(i=1; i<=19; i++)
    { B[i] = i; }
  C = 42;
  for(i=1; i<=19; i++)
    { A[i] = B[i-1] + C; }

  printf("%i %i %i", A[19],B[19],i);
}
```

The unoptimized assembly version of this program can be found in the file `myprog.s` (see `www.liacs.nl/ca/myprog.s`).

1. Add code comments to the unoptimized version (`mypros.s`) 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 structure.

   Your comments should explain the workings of the program and not the (trivial) meaning of the instructions themselves (e.g. We know that lw $2, i loads the contents of variable i into register 2, so don't tell us.)

2. The assembly code in `myprog.s` can be made (much) faster. Optimize this code by hand and explain your optimizations. Any idea that contributes to a performance increase of the program may be used (everything including 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!!! The most important factor for grading is the performance increase (so the less dynamic instructions, the better).

3. Correctness of your code can be checked using the MIPS simulator `xspim`. First simulate the original program and check your own program next. The output of both programs should be equal. Not only the output should be equivalent, the contents of all the variables (arrays A, B and the variables C and i) must be the same too at the end of program execution.

4. Calculate by hand, for the original and the optimized code:

   (a) The number of static instructions in the code.

   (b) The number of dynamic instructions that is executed during program execution.

   (c) The number of data accesses (memory references) done during program execution.

   (d) Assume that the execution time of a program is proportional with the dynamic instruction count. Calculate the speedup of your optimized program.
   NOTES:
   - Do not count the instructions that take care of executing the printf statement

Delivery

Submit (electronically and on paper) a report in which the following can be found:

- Listings of the unoptimized and optimized code (including comments)
- README file with well ordered results(hard to read answers are wrong by definition) of the calculations done.

Technical matters

- MIPS assembly:
  A description of the MIPS instruction set can be found in chapter 5 of the "MIPS Instruction Set Manual". This is a pdf file, which can be found on any local PC at LIACS at the location:
  `/home/csca/public_doc/MIPSPro_Ass._Lang_Vol2.pdf`
  Other information on the MIPS instruction set can be found on the CA website.

- XSPIM
  A manual for xspim can be found at:
  `http://www.cs.wisc.edu/~larus/SPIM_manual/spim-manual.html`
  To make xspim work you need to copy a special file in your working directory. If your working directory is `~/student/ca/opg1`, this file can be copied as follows:
  ```
  cp ~coco/edu/spim/linux/trap.handler ~/student/ca/opg1.
  cp ~coco/edu/spim/linux/trap.handler ~/student/ca/opg1.
  ```
  You should also make a symbolic link to xspim. Depending on if your work on a Solaris or a Linux system, you should execute the following:
  ```
  ln -s ~coco/edu/spim/solaris/xspim ~/placeinpath/xspim
  ln -s ~coco/edu/spim/linux/xspim ~/placeinpath/xspim
  ```
  placeinpath can be chosen any way you want. Just remember to put this path in your PATH environment variable.

- Delivery
  Please read `http://www.liacs.nl/CA` for instructions how to submit your work.

Definitions

- The number of *static instructions* is the number of instructions in the source code.

- The number of *dynamic instructions* is the number of instructions that has been executed when the program has finished.