THE UNIVERSITY OF WAIKATO
Department of Computer Science

COMP311-08B — Computer Architecture
Assignment 2 – MIPS/WRAMP Optimisation

Lecturer:	Matthew Luckie
email:	mluckie@cs.waikato.ac.nz
Due:	8 Sept 2008
Worth:	5%

In this assignment you are to compile and optimise the C-program quick.c. The source code is the same as quick.c that you examined in assignment one – except it has more entries in the array to sort. Please make sure your quick.c file is up to date.

You have the choice of either doing this assignment for the WRAMP boards you used in COMP200, or for a MIPS-based machine which you access with the ssh application. Instructions for WRAMP and for MIPS are given separately below.

You also have the choice of either doing this assignment by yourself, or as a pair with someone else. If you work in a pair, please submit one assignment with both of your names in it.

1. Generate the assembler code generated for the program and annotate it. The annotation should classify each instruction as being in one of the following classes:
   (a) Procedure call execution (including parameter set-up, call, setup and tear-down of stack frame, access of parameters within procedure and procedure return)
   (b) Loop control
   (c) Real execution
   (d) Other (keep this category as small as possible)
2. Calculate the number of instructions in each of these categories actually EXECUTED. This will require you to work out how many times each loop is iterated and so how many times each instruction is executed. By far the best way is to measure the number of times each loop/procedure is executed - a theoretical calculation is very hard to get right. Report these results as a number of instructions (they should add up to the total!) and as a fraction of the total.
3. Devise and conduct experiments to determine the number of clock cycles required to execute each of the following instructions on your chosen processor:
   (a) add and subtract instructions.

   (b) multiply instructions.
   (c) shift instructions
   (d) branch instructions
   (e) load and store instructions
   (f) jal instructions

4. Rewrite the assembler code with the aim to minimise the total number of clock cycles required to execute the program. Suggestions for optimisations are:

   (a) remove redundant instructions;
   (b) optimise register usage
   (c) replace instructions with more efficient instructions that achieve the same result;
   (d) avoid passing array indices as variables;
   (e) `inline` (some of) the function calls;
   (f) anything else you can think of;

5. Run the optimised program and record the new execution time (in clock cycles) and total number of instructions executed. Calculate the CPI for the processor running the new program. Is the CPI calculated for the optimised program the same as the unoptimised program? If not why not?

   A prize of a chocolate fish will be given to the program that executes (correctly) in the least number of clock cycles.

Please submit the annotated assembly listings, and your modified program as well as evidence that it actually works correctly plus the measured number of clock cycles, instructions executed and calculated CPIs for each run.

1 Instructions for WRAMP

When you first login to the lab, you will need to ensure the WRAMP programs are accessible on your machine. If you type wcc and see this:

```
cms-r1-17:[101]~>wcc
-bash: wcc: command not found
```

please type

```
gedit ~/.profile
```

and add

```
export CS200=y
```

to this file. Logout. When you login again, you should be able to use wcc.

To record the instruction count and cycle count, you will need to augment the assembly language program to use the special purpose registers `$ccount` (clock cycle count) and `$icount` (instruction count).

```
main:
     movgs $icount, $0
     movgs $ccount, $0

     ... Code Segment to be timed ...

     movsg $1, $ccount
     movsg $2, $icount
```

note: The execution of these instructions will cause these registers to be incremented and you will need to take this into account when reporting your results.

The lab machines have compilers, assemblers, and linkers available for the WRAMP processor. To obtain the assembly listing for step 1, use

```
wcc -S quick.c
```

This command will create a file `quick.s` To assemble and then link, use these commands:

```
wasm quick.s
wlink -o quick.srec quick.o
```

You can then upload quick.srec to the WRAMP board using remote. Full details of the remote command, can be found in the REX and WRAMP manual found in Moodle. You can also use the following routines:

```
void putch(char);
void putstr(char *);
int  readnum(void);
void writenum(int);
int  readswitches(void);
void writessd(int);
void exit(void);
void delay(void);
```

```
wlink -o quick.srec quick.o /home/200/ex2/lib_ex2.o
```

2 Instructions for MIPS

Your MIPS machine is a Cobalt Qube 2 running NetBSD 4.0. You can login to the machine with

```
ssh <user>@qube.cs.waikato.ac.nz
```

where <user> is your Waikato username. You will be prompted for your password. It is the same password you use to login to the labs. If you can't login, please send me an email so this can be resolved.

To obtain the assembly listing for step 1, use

```
gcc -S quick.c
```

This command will create a file `quick.s`. You can produce a running program with

```
gcc -o quick quick.s
```

The MIPS cpu has a cycle counter but not an instruction counter. To obtain the value of the cycle counter, please paste the following code into your quick.c file and use gcc -S as above.

```
static __inline uint32_t mips_cpcc(void)
{
        uint32_t rv;
        __asm__ __volatile("mfc0 %0,$9;" : "=r" (rv));
        return (rv);
}
```

You can obtain before and after cycle counter measurements with:

```
uint32_t before, after;
before = mips_cpcc();
<code to be timed>
after = mips_cpcc();
printf(‘‘before: %d, after %d\n’’, before, after);
```

note: you will have to put

```
#include <stdint.h>
#include <stdio.h>
```

at the top of your quick.c file for your code to compile. The qube does not have any text editors installed – apart from vi, which you won't want to use unless you are familiar with it. Edit your source code on a local machine, and then upload the file with:

```
scp <file> <user>@qube.cs.waikato.ac.nz:assignment2/
```

To copy a file back from the qube to your local machine to edit, use:

```
scp <user>@qube.cs.waikato.ac.nz:assignment2/<file> .
```