Modern Computer Architectures (ET4074) / Embedded Computer Architecture (IN4340)

MIPS Assembly Programming Assignment

Introduction

This assignment requires writing two MIPS64 assembly programs. Your programs will be executed on the WinMips64 simulator, available at:
http://www.computing.dcu.ie/~mike/winmips64.html (or google WinMips64).

Preparations

- Install the WinMips64 simulator.
- Under Configure > Architecture, change data bus to 11 (means 2k memory)
- Read the documentation and study the example programs.
- Play with simple examples to become familiar with the simulator, its settings and MIPS 64 assembly language.

The Assignment

Translate the C code given in appendix A to MIPS64 assembly and simulate it with WinMIPS64. You should implement and submit two versions of the code:

1. One that is plain and simple. In this version no optimizations on the code are required or desired. We refer to this version as the SIMPLE version.
2. One that is fast and utilizes the compiler tricks (handbook Chapter 2.3; appendix G): loop unrolling (!), algebraic simplification, common sub-expression elimination, code scheduling, etc. We refer to this version as the OPTIMIZED version.

Requirements

To declare, allocate and initialize data, your program must use the following labels:

```
            .data
TAP:    .word   8
```

This will make it easy to modify *TAP*. Your program should work (produce correct results) for *TAP* initialized to arbitrary values up to *64*.

Note that in the SIMPLE version, the function *void fir_filter(...)* must be implemented as a separate procedure, because non-optimizing compilers do not perform function inlining. Make sure you follow the MIPS register conventions for function calls: assuming that the caller does not know what caller-saved registers will be used within the function, you must save and restore all of them in the SIMPLE version. In the OPTIMIZED version you may avoid the overhead of saving unused registers.

Both programs should print the results to the simulator terminal. The correctness will mostly be tested based on this output.

Your program should execute with the default WinMips64 simulator configuration, except the requirements below.

- The SIMPLE version should work when the delay slot is disabled and forwarding enabled (see WinMips64's *Configure* menu).
- The OPTIMIZED version should work when the delay slot is enabled and forwarding is enabled.
- You should use the standard instruction latencies as defined under *Configure*/*Architecture*.

Grading

The maximum grade is 10, and the work will be graded according to the following criteria:

1. Correctness (4.0 points): Both programs should generate the right answers, and only these, for all test cases. Think that your program is part of a larger system, and if it does not match the exact expected answer, the functionality of the entire system would be jeopardized. If the SIMPLE version does not print the correct results to the terminal, the OPTIMIZED version will not be checked! For the optimized version tiny rounding errors because of reordering of floating point instructions are allowed.
2. Clarity (2.0 points): Both programs should be clear, commented.
3. Optimization (4.0 points): You should perform at least 4 different types of optimizations. The grade will depend on the number of optimization types you use and the actual performance improvement achieved by your OPTIMIZED version.

Submission

The homework should be submitted via the BB, no later than **October 10, 2012.** No points will be granted in case of overdue submission.

The submission archive should consist of two plain text files with your programs (*.s files that run in WinMips64), and a one-page document in PDF or plain text format that describes the optimizations you have performed. The description should also contain a table with the number of cycles it took to execute each run, the number of instructions, and the computed CPI, as follows:

TAP	cycles	instructions	CPI
4			
8			
16			
32			

Note that if your report is larger than 1 page, we will only read the first page, and grade it based on that.

Appendix A

```
#include <stdio.h>

//
// This implements an n-tap FIR filter that takes an array of doubles x[] and calculates
// a weighted sum using the coefficients in b[] for each output element in y[].
//

void fir_filter(double y[], double b[], double x[], int n, int tap)
{
    int i, j;
    for(i = tap - 1; i < n; i = i + 1)
    {
        y[i] = b[0];
        for(j = 1; j < tap; j++)
            y[i] += b[j]*x[i-j];
    }
}

//
// The main method contains several big arrays. Two of them are initialized with many values
// in *.s files you do this like:
// .ARRAY_X .double 0.3,45.2,11.2, etc.

#define N 64
#define MAX_TAP 32
#define TAP 8

int main (){

    double y[N];
    double b[MAX_TAP] = {
3.558363, -0.542859, -0.928322, -0.993428, 4.643953, -5.909149, 1.722120, -1.854375,
-1.967672, -2.170269, 0.673272, 0.182548, -4.812593, 0.818270, 0.889598, 0.322937,
-0.430576, -4.259421, -0.983597, 4.285341, -2.333795, 1.856496, -4.711222, -0.517585,
-8.376492, -1.440117, -6.255069, -0.000203, 0.881922, 6.955272, -0.209273, 5.219892 };
    double x[N] = {
0.286381, 0.310398, 0.732308, 0.301956, 0.053523, 0.431617, 0.999498, 0.801469,
0.639967, 0.293192, 0.404720, 0.823173, 0.695269, 0.744480, 0.676903, 0.968651,
0.524716, 0.843609, 0.562026, 0.297492, 0.384924, 0.379665, 0.087916, 0.535140,
0.224908, 0.136965, 0.186164, 0.811894, 0.010466, 0.754323, 0.227373, 0.296848,
0.064720, 0.959681, 0.598804, 0.118243, 0.391298, 0.598302, 0.919712, 0.031265,
0.891494, 0.324432, 0.854438, 0.586763, 0.068912, 0.531341, 0.555414, 0.593628,
0.374950, 0.117441, 0.891120, 0.759874, 0.497106, 0.979036, 0.295014, 0.722014,
0.116001, 0.481178, 0.533908, 0.126467, 0.235500, 0.761281, 0.423315, 0.300220 };

    int i;

    fir_filter(y,b,x,N,TAP);

    printf("Result data:\n");
    for(i=0; i < N; i++)
        printf("%f %f\n", (double)i, y[i]);
    printf("end\n");

    return 0;
}
```