CSE 30321 – Computer Architecture I – Fall 2010
Final Project Description and Deadlines
Assigned: November 18, 2010 **– Due:** (see this handout for deadlines)
This assignment should be done in groups of 3 or 4.

Introduction

For this project, you are asked to apply what you have learned over the course of this semester in a capstone design project. More specifically, you first will be asked to write assembly code for a set of benchmarks that might run on the pipelined MIPS datapath that was developed and discussed during the second half of the semester. You will initially assume that the benchmarks will be run on a single core, pipelined MIPS processor. After quantifying baseline performance, you will then be asked to optimize benchmark performance assuming a single threaded, 4-core MIPS machine. This may involve augmenting the datapath to support new instructions, re-thinking your code to take advantage of the ability to run it in parallel, etc.

Benchmarks

Overview:
You will need to quantify the performance of 3 benchmarks specified below.

Benchmark 1 – Fibonacci:
- Write MIPS code that will add the first N elements of the Fibonacci sequence.
- You should assume that an argument N would be passed into a function that would perform the calculation of the Fibonacci numbers and return the resulting sum.
- The function does not have to be recursive!

Benchmark 2 – The SAXPY Loop:
- The SAXPY loop is a common routine in linear algebra and a common fixture in benchmark suites. Quite simply, the routine is:

```
for (i=0; i<N; i++)
    q[i] = A * x[i] + y[i]
```

- Arrays q, x, and y will all be stored in memory; A is a scalar
  - Their starting addresses are specified in $s0, $s1, and $s2 respectively.
- You should assume that data to be processed is stored in on-chip memory (and could thus be accessed by all cores).
- You can assume that N would be given as an argument to a function that will execute the above pseudo code.
  - Note: for this – and all – benchmarks, you can have the above function call others if desired

Benchmark 3 – Sorting:
- Write MIPS code that implements a sorting algorithm.
- For this benchmark, your sorting function can implement any sorting algorithm that you may have learned about in your Data Structures course – e.g. perhaps Quicksort, Bubble sort, Insertion sort, Shell sort, Heapsort, Bucket sort, Radix sort, Distribution sort, Shuffle sort, etc.
  - You **cannot** choose to implement Mergesort – although you can use it in conjunction with a new algorithm
- You should assume that data to be sorted is stored in on-chip memory (and could thus be accessed by all cores).
  - The address of the first element in the array to be sorted will be stored in $s0
  - The number of elements in the list to be sorted will be stored in $s1
  - For simplicity, you may always assume that loads take just 1 CC – i.e. there are no cache misses to contend with for *any* of the benchmarks.

Performance Baseline

Part 1:

- Write the MIPS code for each of the above benchmarks and test it with XSPIM.
    - To simplify testing:
        - For benchmark 1, compute the sum of the first 6 Fibonacci numbers (starting with 0)
        - For benchmark 2, assume that *i* is equal to 5
            - The array *x* should contain the values: {2, 4, 6, 8, 10}
            - The array *y* should contain the values: {1, 2, 3, 4, 5}
            - The scalar A should be: 20
        - For benchmark 3, sort the list: {7, 10, 9, 3, 17, 2, 24, 18}

Part 2:

- After you have written and tested all of the above benchmarks for a single core, pipelined MIPS processor, determine the total number of clock cycles required for all 3 benchmarks to run sequentially.
- You should assume that you will need to:
    - Calculate the sum of the first 100 Fibonacci numbers (ignore any register overflow)
    - Run the SAXPY loop with *i* = 20,000
    - Sort an array of 5,000 elements
- Other notes:
    - Because the datapath is pipelined, you should account for necessary stall cycles because of data hazards – so be sure to think about register allocation when you are writing your initial MIPS code
    - You should assume full forwarding, that data can be read / written in the same CC, etc.
    - To avoid overly complex overhead, assume branches are always predicted correctly.

Design Enhancements

After calculating a baseline execution time for the three benchmarks, you need to think about ways to improve overall performance when they are run on a machine with four cores instead of one. While obviously performance can (and should) come from parallelism, there are other ways to improve performance too. To help you to get started, you might consider some of the following:

- Increase the pipeline depth
    - (You would need to revisit and account for stall cycles caused by data dependencies.)
    - Also, your assumptions must be realistic. Thus, (unless you move the data memory reference hardware) if a lw currently gets data from memory at the end of the 4th CC, and you split the EX stage and M stage into 2 different clock cycles, the lw should get its data at the end of the 6th cycle.
- Smartly parallelize the benchmarks among different cores
    - If one of your benchmarks takes a longer amount of time than the other 2, it may make sense to try and parallelize it among N of the 4 cores, and run the other benchmarks sequentially.
    - Note that while you will not be required to write MIPS assembly code to parallelize benchmarks, you will have to consider the instruction overhead that will be required to parallelize execution (see notes in the "Fixed Overheads" section below).
- Add new instructions to make your benchmarks run faster
    - Note that if you choose to do this, you should generate a schematic of an augmented datapath; the baseline datapath should be that for the MIPS 5-stage pipeline.
    - Also, you should include a table in your final report that details what new hardware was added, comment on how frequently that hardware would be used, etc. (Adding excessive amounts of infrequently used hardware to improve the performance of just 3 benchmarks is not a smart design decision – and will be graded as such.)
    - While you may not be able to test a new instruction in XSPIM, you should discuss why it would help, include an excerpt of code in your report showing how your program will still be correct, etc.
- More advanced architecture techniques – like register renaming.
    - See me if you are interested in pursuing this route.
    - (You might leverage the SimpleScalar cross compiler if you are interested in this option.)

Note that combinations of the above are also reasonable – and may be influenced by decisions like what sorting algorithm you develop.

Project Guidance

In "real life" an enhancement to a processor often does not make *all* code or jobs run faster. In some cases, an enhancement may only help with just a few types of applications. That said, an enhancement should not make the performance of the other tasks significantly *worse* either. When you suggest your design improvements, you should keep the above in mind (i.e. your enhancements should probably help at least 2 of the benchmarks while not adversely hurting the third; of course if your new design improves all 3, that's great too!)

To give you some sense of proper scope, consider the following hypothetical project: After writing your baseline benchmarks, you determine that the sorting benchmark is a bottleneck (i.e. it takes much longer than all others). You might describe how you will parallelize your sort code. (And perhaps why you chose this approach as opposed to parallelizing all benchmarks.) To determine *how* to best parallelize your sorting algorithm, you investigate the impact of 2-core and 4-core parallelization (taking into account the fixed communication overhead). You also notice your SAXPY code could benefit from a "load and increment" instruction. To add this instruction, you describe the changes to the datapath and control, and explain how the performance gains justify hardware additions.

Note that I *will* take into account the complexity of your algorithm when determining your final grade. This does not mean (for example) that you have to implement the most difficult sorting algorithm possible. However, if you chose to write MIPS code for a parallelized Quicksort and spend less time augmenting the datapath hardware, this extra effort will be taken into account. If you think an algorithm you chose to implement may not perform as well because of the number of elements that must be sorted, you may comment on when (i.e. for what problem size) your code would be advantageous. Similarly, you may choose to implement a simpler sorting algorithm and spend more time considering parallelization tradeoffs among the different benchmarks.

Fixed Overheads

To ensure some commonality among designs, you must assume the following:

- The latencies for the logic in each pipe stage are as follows:
  - **IF**: 500 ps; **ID**: 350 ps; **EX**: 425 ps; **M**: 475 ps; **WB**: 375 ps
  - Thus, the initial clock rate for this machine is 1 GHz
- Every time a message / piece of data is sent from one core to another, there is a 20 CC overhead
- To launch part of a program on a new core, there is a 50 CC overhead.

Design Bonuses

While you are essentially free to choose what enhancements you make to your design, do note that extra credit will be provided for the following:

- The design that offers the lowest execution time for all 3 benchmarks (with justification)
- The best overall report
- The most "clever" design and/or algorithm

Deadlines and Deliverables

This is a team-oriented effort. Each design team may consist of 3-4 students. Though all team members need to participate in the entire design process, it is suggested that different team members take a lead role for different sub-tasks.

Major deadlines are summarized below:

- Nov. 18th: Project assigned
- Nov. 24th: Project proposal due via email (plain text preferred)
- Dec. 16th: Final report due at 3 pm.

The following grade distribution will be used:

- Proposal: 10%
- Final report: 90%
- Group evaluation: my discretion

Project Proposal

Each team should submit a formal project proposal (1-2 paragraphs in length) by November 24th. The proposal should include a discussion of your approach and initial thoughts for a proposed solution. I will provide feedback on these proposals to ensure that your group is not doing too much or too little. Please discuss your proposal with me if desired.

Final Report

The final report is an important part of the project. The report must adhere to the following guidelines:

- Each team is allowed 4 pages for their final report – including figures and tables.
- While smaller fonts can be used in figures and tables, the font size used for the report text cannot be smaller than 11 point. Table font size cannot be below 9 point.
- The margins on each page must be 1 inch.

Each team should turn in one report. The report is due on December 16th by 3 pm. While you may choose to report other information as you see fit (i.e. to explain your design decisions), the report must:

- (Briefly) explain your rationale for creating your baseline benchmarks
- Explain your rationale for your design enhancements
- Quantify how your new benchmarks outperform the old
- Discuss how design / software enhancements led to performance gains

When preparing the final report, think about the reader. Assume the reader is your manager whose knowledge about computer architecture is about the level of an average student in this class. Use descriptive section titles to help with readability. Include figures and tables wherever necessary. Use formal English language (e.g., no slang, no contractions). Also, make sure that correct technical terms are used. *Proofread the report before turning it in.*

Also, turn in your MIPS code (that can run in XSPIM) that demonstrates that the core of all 3 benchmarks works correctly. Please include a README file so that we can easily determine any initializations, etc. that might be required so that we can test. In the report, list the name of the dropbox where the code was deposited.

Group Evaluation: Each team member should turn in a group evaluation sheet individually.