CSE 30321 – Computer Architecture I – Fall 2011
Final Project Description and Deadlines

Assigned: November 10, 2011 **– 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 write and test the MIPS assembly code for the 3 benchmarks described below:

Benchmark 1 – Sorting:

- Write MIPS assembly code that implements a sorting algorithm.
  - You can implement any sorting algorithm that you chose (and that you may have learned about during 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 sorting algorithm.

Benchmark 2 – Searching:

- Write MIPS assembly code that implements a binary search.
- For this benchmark your code can be either iterative or recursive

Benchmark 3 – Matrix Multiplication:

- Write MIPS assembly code to multiply two matrices together.
  - (matrix multiplies are common in scientific computing, graphics transformations, etc.)
- For this benchmark, you can store the data associated with each matrix in whatever way you best see fit. As one example…
  - The array might be stored in the "row major" form – i.e., all the elements are stored in successive memory locations in the following order:

    a[0][0], ..., a[0][m-1],
    a[1][0], ..., a[1][m-1],
    ...
    a[n-1][0], ..., a[n-1][m-1]

    (Thus, a[0][0] is in the first memory location, a[0][1] is in the second, etc.)

  - Alternatively, you may choose to store data in a different way too if there is a better fit for the MIPS code that you write.
    - The only requirement is that you should be able to multiply 2, arbitrarily sized, M x N matrices.

Functionality Testing

- Write the MIPS code for each of the above benchmarks and test it with XSPIM.

- To simplify testing, you should assume the following:
  - For **Benchmark 1** (**sorting**), you should sort the list of numbers shown below:

    {11, 18, 2, 100, 97, 210, 5, 42, 17, 125, 126, 33, 17, 11, 209, 50}
  - For **Benchmark 2** (**searching**), you should "find" the number 24 in the following list of numbers:

    { 0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60,
    64, 68, 72, 76, 80, 84, 88, 92, 96, 100, 104, 108, 112, 116, 120, 124,
    128, 132, 136, 140, 144, 148, 152, 156, 160, 164, 168, 172, 176, 180, 184, 188,
    192, 196, 200, 204, 208, 212, 216, 220, 224, 228, 232, 236, 240, 244, 248, 252,
    }
  - For **Benchmark 3** (**matrix multiplication**) multiply matrices A and B together:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

    (In your report, be sure to specify how data is written so that we can interpret the results of your code correctly.)

Performance Baseline

After you have written and tested all of the above benchmarks with XSPIM, you will need to determine the total number of clock cycles required for all 3 benchmarks to run sequentially on 1 core. The baseline performance numbers should reflect the assumptions described below:

- For **Benchmark 1** (**sorting**), you should estimate the amount of time required to sort an 8000-element list.
  - Note that for this benchmark – and the other 2 as well – XSPIM simulations are not required. Just estimate the additional run time based on the test case.
  - Additionally, if appropriate, you should report an average sort time and a worst-case sort time for your code.
- For **Benchmark 2** (**searching**), you should estimate the amount of time required to find the 50th, 500th, and 5000th elements in an 8000-element list. Report the average time.
- For **Benchmark 3** (**matrix multiplication**), you should estimate the amount of time required to multiply 2, 250 x 250 matrices together.
- Other notes:
  - Your performance projections should assume a datapath with a 5-stage pipeline (i.e. just like the pipelined datapath discussed in class).
    - The only stall cycles you need to consider are for a lw instruction that is followed by a dependent instruction
    - To avoid overly complex overhead, assume branches are always predicted correctly, full-forwarding, that virtual addresses always result in a TLB hit, that instruction and data cache references result in L1 hits, etc.
  - Don't forget to account for the time to fill and drain the pipeline.
- *Your report should briefly summarize how your baseline data was obtained.*

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).
        - Note that Lab 05 might offer some insight as to how to attack the matrix multiply benchmark.
- 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.)
    - 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 choose to implement.

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 aforementioned overheads to be discussed below). You also notice your matrix multiply 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:

- Thus, the initial clock rate for this machine is 1 GHz

- You may assume that all of the data for a given benchmark is contained in a shared, on-chip, L2 cache. However, if a benchmark is parallelized, data will need to be copied to the respective L1 cache of a given core. As such, you should assume that (on average) 5 CCs would be required to move a data word (i.e. an element of an array to be sorted) to/from a given core's L1 cache. *Thus, this represents overhead associated with parallelization.*
    - Don't forget that upon completion, data must be moved *back* to the L2 cache – thus the 5 CC overhead must be accounted for twice.

- Similarly, to launch part of a program on a new core, there is a 200 CC overhead per instantiation – in addition to that required to move data from L2 to L1.

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. 10th: Project assigned
- Nov. 17th: Project proposal due via email (plain text preferred)
- Dec. 8th: Final report due; can be turned in by Dec. 15th at 3 pm with no late penalty

The following grade distribution will be used:

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

Project Proposal

Each team should submit a project proposal by November 17th. The proposal should include a discussion of your approach and initial thoughts for a proposed solution. If appropriate / needed, 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.

Note that for your proposal, a short, 1 paragraph description OR 4-5 bullet list is sufficient. Please CC all team members on your email – and include the names of all team members in the email.

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 15th 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.

CSE 30321 – Computer Architecture I – Fall 2011
Final Project Rating Sheet

Your Name:

Please write the names of all of your team members, INCLUDING YOURSELF, and rate the degree to which each member fulfilled his/her responsibilities in completing the assignments. Sign your name at the bottom. The possible ratings are as follows:

Excellent:
- Consistently went above and beyond the basic team responsibilities – tutored teammates

Very good:
- Consistently did what he/she was supposed to do, very well prepared and cooperative

Satisfactory:
- Usually did what he/she was supposed to do, acceptably prepared and cooperative.

Ordinary:
- Often did what he/she was supposed to do, minimally prepared and cooperative

Marginal:
- Sometimes failed to show up or complete assignments, rarely prepared

Deficient:
- Often failed to show up or complete assignments, rarely prepared

Unsatisfactory:
- Consistently failed to show up or complete assignments, unprepared

Superficial:
- Practically no participation

No show:
- No participation at all

These ratings should reflect each individual's level of participation and effort and sense of responsibility, not his or her academic ability.

Group Member Names:	**Rating:**
_____	_____
_____	_____
_____	_____
_____	_____

Signature: