# ECSE 425 Project Specification, Revision 1

Fall 2011

http://www.info425.ece.mcgill.ca/project.html

This semester, all students will work in pairs to complete a final architecture project in which they:

1. Propose modifications to the MIPS64 architecture, as implemented by the EduMIPS64 simulator,
2. Implement these changes in the simulator,
3. Evaluate the effect of these changes on performance,
4. Present their modifications and experimental results to the class, and
5. Submit a final report for review by the instructor.

The objective of this course is to familiarize students with architectural simulation, and reinforce concepts discussed in the class with hands-on implementation, debugging, and experimentation.

## Resources

The EduMIPS64 simulator and related resources can be found at:
http://www.edumips.org.
The core files needed for the project will also be available from course project web site:
http://www.info425.ece.mcgill.ca/project.html.

## Grading

In total, the final project is worth 30% of the semester grade. A single grade will be given for each group; students should therefore choose their partners wisely. The project grade is broken down into several components, discussed below:

| | | |
|---|---|---|
| • Project Proposal | 5% | due October 31 |
| • Progress Report | 10% | due November 16 |
| • Final Presentation | 25% | in-class, December 5 and 6 |
| • Final Report | 60% | due December 6 |

*Project Proposal*

For the project proposal, students will prepare a paragraph, submitted by email to the

instructor, indicating (a) the members of their team, (b) their preferred date for the final presentation (either December 5th or 6th), and (c) the investigation the team will conduct. The proposed investigation may be taken from the list provided below, or may be suggested based on the interests of the students. The proposed investigation must be approved by the instructor, and the grade for the final report will be determined in part by the extent to which students achieve their stated goals. Significant overlap among groups will not be allowed, and projects will be approved on a first-come, first-served basis. Choice of presentation dates will also be given on a first-come, first-served basis.

To receive full credit, proposals must be received no later than 10:35 AM on October 31; proposals are to be sent to *brett dot meyer at mcgill dot ca*.

*Progress Report*

For the progress report, students will prepare a paragraph, submitted by email to the instructor, describing (a) the work done so far, (b) the work remaining, and (c) any problems that have been encountered. The progress report is a natural time for students to raise any issues that may prevent them from completing the work they originally proposed.

To receive full credit, Progress Reports must be received no later than 10:35 AM on November 16; reports are to be sent to *brett dot meyer at mcgill dot ca.*

*Final Presentation*

Each group will prepare and give an oral presentation during class on either December 5th or 6th. The use of PowerPoint or equivalent is encouraged. The presentation should summarize the final report. It should be no longer than 12 minutes and must:

- Describe the proposed modifications (*e.g.*, improved branch prediction),
- Give the motivation behind the proposed modifications (*e.g.*, to further reduce control hazards),
- Give an overview of the implementation (*e.g.*, extending the Instruction object class),
- Detail the experimental setup used to evaluate the modifications (*e.g.*, what benchmarks and why),
- Briefly describe the results (*e.g.*, if things sped up, and by how much), and
- Briefly discuss things that didn't work as expected, and if possible, why.

*Final Report*

Each group will prepare a final report no longer than five pages (US Letter, 1" margins, 12 pt font, single spaced). The report should include the same information as the presentation, but provide greater detail. For example, the document might be organized with the following headings:

1. Introduction—briefly, what was done, why, and whether or not it was successful;
2. Approach—briefly, the modifications that were made to EduMIPS64, and why;
3. Implementation—how the architectural changes were translated into changes to the EduMIPS64 source code;
4. Experimental Setup—how the architectural changes were evaluated;
5. Results—the results of the evaluation; and,
6. Post-mortem—what went well, what went wrong, and other final thoughts.

Deviation from the above outline will not be penalized, provided the relevant information is captured. The instructor recommends writing the report first, and subsequently summarizing it for the presentation, rather than the other way around.

To receive full credit, Final Reports must be received no later than 11:59 PM on December 6th; reports can be handed in during final presentations, or submitted by email to *brett dot meyer at mcgill dot ca.*

Each group must also submit the sources they modified, together with any benchmarks they developed for the purpose of evaluating their modifications. A description of the sorts of tests that should be applied to verify the modifications should also be included. These files should be packaged and sent as a single archive to *brett dot meyer at mcgill dot ca* no later than 11:59 PM on December 6th.

**Suggested Projects**

All projects must, at a minimum, (a) modify the baseline simulator, and (b) evaluate the performance effect of the modification. Furthermore, (c) modifications may not break the architecture: programs must yield the same results before and after modification.

To satisfy (b), groups should develop or select their own set of benchmarks (examples are available), and provide justification for why the chosen mix of benchmarks is appropriate given the particular modification under test. The baseline simulator may be used to verify correct program behavior, therefore satisfying (c).

Projects that fail to satisfy (a)-(c) above will be penalized substantially. Beyond these requirements, there are many possible modifications to pursue; examples are listed below; feel free to propose your own.

*Pipeline Mechanics*

By default, EduMIPS64 performs no forwarding. Explore the effect on performance of forwarding, by comparing performance when (a) there is no forwarding, and either (b) partial forwarding, or (c) full forwarding.

By default, EduMIPS64 allows registers to be read and written in the same cycle, and assumes the processor can read from instruction memory and data memory at the same time. Explore the effect of structural hazards:

- by preventing simultaneous reading and writing of the register file, or
- by using a single memory for instruction and data access.

EduMIPS64 does not support precise exceptions. Explore the effect they have on performance by requiring that floating point operations not overlap in execution.

*Branch Prediction*

By default, EduMIPS64 predicts that branches are not taken. Explore the effect on performance of:

- flushing the pipeline when branch instructions are encountered,
- predicting taken, or
- dynamic prediction techniques, such as
    - an n-bit local predictor,
    - a (m, n) correlating predictor,
    - a tournament predictor, or
    - a branch target buffer.

Alternatively, in the context of any of the above, explore the effect on performance of when branch resolution is completed:

- What is the penalty if EduMIPS64 waits until EX to resolve branches?
- What would the benefit be if branches could be resolved during IF?

*Caching*

By default, EduMIPS64 assumes single-cycle memory access latency. However, EduMIPS64 implements an interface for the Dinero cache simulator. Use Dinero to explore the effect on performance of:

- direct mapped caching,
- set associative caching,
- multi-level caching, or
- shared/split instruction and data caches at various levels.

Additionally, in the context of any of the above, explore the effect on performance of the delay required to access particular levels of cache, the size of caches, etc.

*Other Topics*

Ambitious students might consider more ambitious projects, such as implementing all or part of the logic required to support out-of-order execution, *e.g.*, a re-order buffer. Other topics will be considered on a case-by-case basis. *Note*: students' grades will be determined in part by the extent to which they achieve their stated goals.

**Additional Notes**

EduMIPS64 includes a GUI for visualizing the progress instructions make through the pipeline. Groups need not modify the GUI to reflect the changes they've made, though students may find the GUI simplifies debugging.