Optional Final - In lieu of the in-class final
378 Fall Quarter
Due December 17th - NO LATE DAYS (Early morning of the 18th is OK)

WORK INDIVIDUALLY

For the true masochists among you this is the *optional* replacement for the final exam. You can do this project instead of showing up and taking the in-class final. This design project in SMOK is optional because it is consistently more difficult than the examination. I offer this as an option not because it is easy but because, frankly, you will learn more from this than taking a final. That being said, a word of caution:

(a) This project is more difficult than I would assign in a graduate course.

(b) Do not do this project to the detriment of studying for your other courses! This project is INSANE. Please pace yourself and gauge whether or not you should just take the in-class final.

(c) Be careful of the due date. It is specifically setup to be the night of the 17th (NO EXTENSIONS), in case you do not get it to work you can show up and take the final. You will NOT GET YOUR PROJECT GRADED PRIOR TO THE FINAL. It will take Tim & Dmitriy a LONG time to grade this project.

(d) If you take the final and do this project your grade on the "final" will be the maximum of this project or the in-class final (not additive).

You MUST work individually on this project. This project is in lieu of a final worth 35% of your grade, and as such it should be your own work. Just like we do not proctor exams (we just show up to answer questions), we expect you to do your own work. Feel free to discuss the project with myself and the TAs though.

NOTE: It is impossible for me to cleanly specify all of the details about how to construct this assignment. The next page will list the highlights of the things you must support. You will probably find a ton of things you have to adapt along the way because I didn't specify them completely or correctly. Minor adaptations you can do on your own, any major ones you should run by Tim / Dmitriy or myself. As with HW5, please include a readme.txt with any changes you make to the specifications. Be careful about the changes you make -- if you make substantive changes we may have to judge you haven't done the project. So when in doubt run your changes by us.

I want to stress again: you probably have more than one class and as such I definitely do not want you to hose those classes because you where taking this insane optional final. Please use caution while driving with SMOK.

That being said, your mission, should you choose to accept it, is to design a working in-order dispatch / out-of-order execute / out-of-order completion pipelined processor with split L1 caches. Here are the details:

- The processor should fetch and dispatch up to two instructions per cycle (in order).

- It should have 4 functional units: a load/store unit, a branch resolution unit, and two integer ALUs.

- Each of the load-store and integer functional units should have 4 reservation stations above it. The integer functional units should be able to execute any ready instruction from any reservation station above it. The Load/Store unit should execute from the reservations stations in order so that total load/store ordering is maintained (unless you want to get psycho and make a unit that speculates on load/store instructions).

- The branch resolution unit should have 1 reservation station above it. The branch unit should not use the completion buses but rather interface directly with the fetch unit.

- It should be able to retire (complete) 2 load-store or integer instructions per clock cycle from any of those three functional units. Favor the load/store unit and make some policy for choosing between the integer units.

- Its data cache should be write-back, 2-way set associative, have a line size of 16 bytes (4 words), and a set size of 32 lines.

- The instruction cache should be the same (2-way set associative, 16 byte line size, 32 lines) but not support writes (hence no dirty bits). Since your fetch unit is fetching two instructions per clock cycle you should have 2 READ ports on your instruction cache. This is a non-trivial thing to do, but is nonetheless possible.

- The processor should support all of the instructions from HW5.

- You should write a function "mergesort" that implements the mergesort sorting algorithm. Make an array of 128 words and place it somewhere in memory. Sort this array with your mergesort code. Your mergesort() function should be recursive! Ask me if you need to know how mergesort works and i'll point you to some references.

- The basic pipeline will be: FETCH / RENAME-DISPATCH / EXEC / COMPLETE for integer operations. For Load/Store instructions the pipeline will be FETCH / RENAME-DISPATCH / EXEC / MEM / COMPLETE.

- You should not stall the whole out of order execution and fetch core due to a cache miss in the data cache. Instead you should simply stall the load / store unit. You do NOT have to implement a non-blocking data cache. You can have your cache handle only 1 I/O operation at a time.

- You should not stall the whole out of order execution and load/store core due to a cache miss in the instruction cache. Instead you should simply stall the fetch unit.

- Use the DRAM timing information from HW6 and the delay timer method for accessing a *unified* DRAM memory for both instructions (starting at address 0) and data (starting at 512). Since your DRAM is unified you should make a single arbitration unit that selects requests for reading / writing from the instruction and the data cache. Do NOT split the DRAM into an instruction and data DRAM.

- Construct your own register file from individual registers. This register file should support the reading of 4 items and writing of 2 items per clock cycle. Use lots of muxes, etc. Do not use the SMOK register file since you need more I/O ports.

- You can check out and read "Computer Architecture A Quantitative Approach" by Hennessy and Patterson. Chapter 4 of that book will be helpful for this assignment. I will make photocopies of chapter 4 and you can pick them up from me if you want to attempt this project.