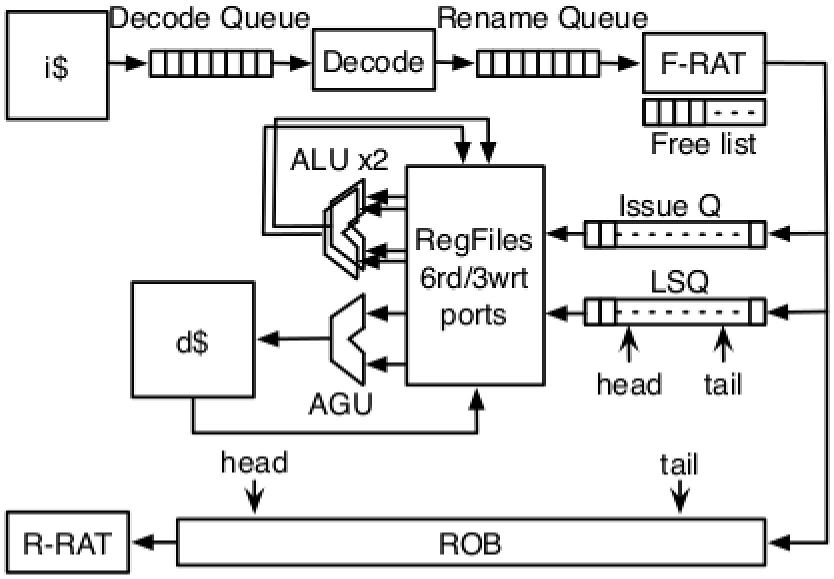
ECE 401: Instructions for Project 3

1. Description

In this project you are going to get familiar with designing an out-of-order superscalar processor. An illustrative example of the desired architecture is shown below.



Fetch/Decode/Rename /Issue/Commit width	2/2/2/2+1/2	Issue Queue	16-entry, Compacting
Branch Prediction	Always not-taken	Load/Store Queue	16-entry FIFO, Wakeup & Select
Decode Queue	8-entry, FIFO	Register File	64 entries, 6 Reads, 3 Writes
Rename Queue	8-entry, FIFO	Functional Units	2 Integer ALUs, 1 AGU
Renaming	Dual-RAT, No Check-pointing	ROB	64 Entries

2. Requirements

In this project you are asked to complete the following tasks.

- Compile and run the provided code for all the test programs and immediately report if you see any problem.
- The desired parameters for the out-of-order scalar processor are summarized in the table above.

- On a branch misprediction, wait for the branch to become ROB head, then copy R-RAT to F-RAT (No checkpointing mechanism is required).

- A separate issue queue is used for load and stores, which is in-order queue with wakeup and select mechanism.

- Start with the provided in-order scalar processor and implement the desired out-of-order scalar. Then implement dual issue for the desired out-of-order superscalar processor.

- First level instruction and data cache sizes are the same as last project.

- To test your design run the test programs provided for this project. See the list of the test programs in the table below.

Application	**Total Instructions**	**Application**	**Total Instructions**
noio	1997	hanoi	201315
fact12	110522	class	96915
file	95220	ical	216217
matrix	136979	sort	104656
hello	95443	fib18	305487

3. Bonus

There are up to 100 bonus points available on this project for implementing optimizations to the required structures. This includes, but is not limited to,

- More accurate branch prediction
- Check-pointing
- Cache optimizations such as critical word first and early restart. If doing these optimizations make the following assumptions about the memory system, 2 cycle access latency with 1 word provided per cycle after that.
- Non-blocking/lockup free caches. Start with single hit under miss, then single miss under miss and finally multiple miss under miss.

There are other optimizations that can be made, in general anything we have talked about in class that applies to an out-of-order processor is fair game. Bonus will be provided proportional to the difficulty and quality of implementation. If you have any questions about the applicability of an optimization ask the TA.

4. Sources

The provided source files can be found on blackboard. They can be extracted with the following command.

- tar -xzvf project3.tar.gz

Test files and compile script are the same as projects 1 and 2. They are also available on blackboard but do not need to be re-downloaded.

5. Compile and Simulate

After copying the sources into your home directory you can compile and run the simulation using the following commands.

- ./compile
- ./VMIPS *test_program*

Replace *test_program* by your desired test program for verification.

5. Hand—In

- The verilog and cpp files (compressed using tar –cvf sim_src.tar sim_src).
- Project report, including discussion of implementation, IPC report table, and a description of any bonus features implemented (only in PDF format).