Out of Order Roadmap

Weekly Road Map

Week 1: Finish implementation of baseline ERR processor
Week 2: Design parameterizable, pipelined cache, start/finish superscalar design
Week 3: Add remaining desired advanced features (BTB, RAS, Load/Store queues with forwarding)

CP1

Progress Report:

- Block Diagram : Logan, Josh, Lorenzo
- Queue: Logan, Josh, Lorenzo
- Fetch and magic memory w/ queue: Logan, Josh, Lorenzo
- Roadmap + Progress Report : Logan, Josh, Lorenzo

Functionalities Implemented:

- Parameterized Queue : Signals status of queue (full or empty) and outputs dequeue data
- Fetching with magic memory and using the queue to hold the instructions

Testing Strategies:

- Created a TB to randomly create instruction data that will be enqueued when able to and dequeued when full
- Tested specific queue scenarios such as simple queue and dequeue, dequeue when empty/full, enqueue when empty/full, and continuous enqueue and dequeue 100 times.

CP2 Plan Ahead

Functionalities To Complete: (need to assign who does what still)

- Integrate Multiplier into CPU as Functional Unit - Logan
- ROB - Logan
- Common Data Bus (CDB) - Josh
- Reg File - Josh
- Free List- Josh
- Front End (Fetch, Decode, Rename, Dispatch) - Lorenzo
- Reservation Station + Wakeup - Lorenzo

L0 register cache - fetch / memory functional unit
Speculative Stores - in the memory

CP2

Progress Report:

- Reservation Stations: Josh and Lorenzo
- ROB: Josh, Lorenzo
- CDB: Josh, Lorenzo
- Roadmap + Progress Report : Logan, Josh, Lorenzo
- FreeList: Logan
- Rats: Logan
- Functional Unit: Josh, Logan
- Physical Regfile: Logan

Functionalities Implemented:

- Out of order execution for ALU and Mult instructions

Testing Strategies:

- Test every module and track cycles to make sure its everything is happening when we want it
- Connect RVFI
- Test Individual instructions both IMM and REG to make sure writeback from CDB works
- Check step by step/ cycle by cycle to make sure everything is happening when it needs to

CP3 Plan Ahead

Functionalities To Complete: (need to assign who does what still)

- LSQs(Logan Josh)
- Control Logic(Lorenzo)
- Integrate Cache(Logan)
- Instruction Cache(Josh)
- Branch Predictors(Lorenzo)
- Wallace and Shift subtract divider (Logan)

CP3

Progress Report:

- Fixing mul: Logan
- Fixing branching: josh
- Fixing cache: Lorenzo i

Functionalities Implemented:

- Out of order execution for ALU and Mult instructions
- Load/Stores implemented out of order wrt with other instructions
- Branch resolution

Testing Strategies:

- Test every module and track cycles to make sure its everything is happening when we want it
- Make TB
- Use Spike
- Unit Test

CP3 Plan Ahead

- Branch Predictor: Lorenzo
- Speculative loads: josh
- Wallace and Shift divider: Logan
- Super scalar: everyone

Functionalities To Complete: (need to assign who does what still)

- Advanced features

Functional Description of Modules

Fetch:

Purpose: Maintain a program counter and fetch instructions from an instruction cache. Will stall the fetching pipeline while memory responds, especially with cache misses. Retrieving instructions from memory may take more than one cycle and in order to keep the flow of fetching instructions, we can have a register that will store an enqueue signal and the current PC + Valid Signal. Then, when we get a response from memory, we will write data into the front end of the register and flop the register. Additionally, while we are waiting for imem to respond, we can stop incrementing the PC.

Once, we have flopped the register forward, the IQ receives the enqueue signal, and data is written into this queue. It will take one cycle for this queue to be updated. Now it is the job of the decode stage to send a signal to dequeue and extract information from the instruction.

It is also important that the fetch register will not flop while the queue is full. We will also not increment the PC and turn off the read mask while the queue is full.

Data Flow:

- Cycle 0: Processor updates PC, assigns it to the instruction memory address, and sets instruction memory mask. It will also load an enqueue signal to the front end of a register.
- Cycle 1-10*(Dependent on cache hit/miss): Memory responds and then PC can incremented to start fetching the next instruction. The register is flopped and the data from memory will be enqueued.

- Cycle 2: Queue may send a full signal at this point, but there will be an instruction in flight from memory, this means that we need to prevent the register from flopping forward, and to stop fetching from memory until the buffer isn't full. Data will now be available on the queue and can be dequeued by the decoding stage, given that the empty signal is off.