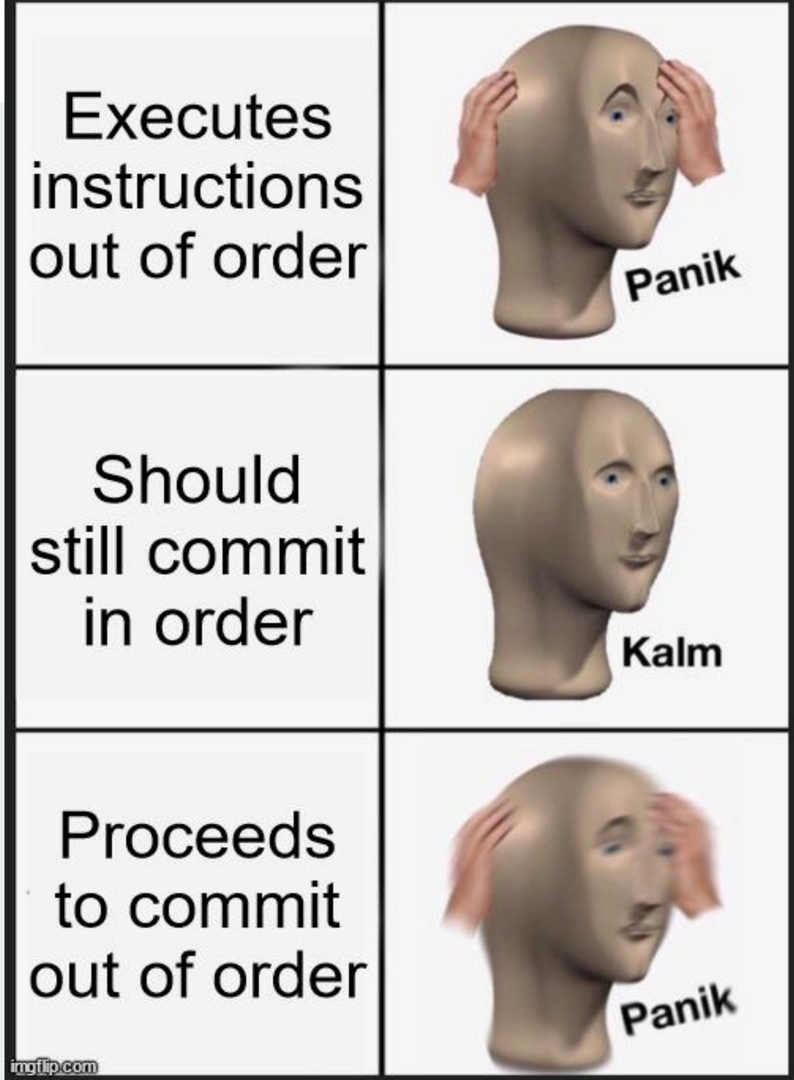
mp_ooo Lab 2

Presented by: people


Executes instructions out of order
Panik
Should still commit in order
Kalm
Proceeds to commit out of order
Panik
imgflip.com

Topics

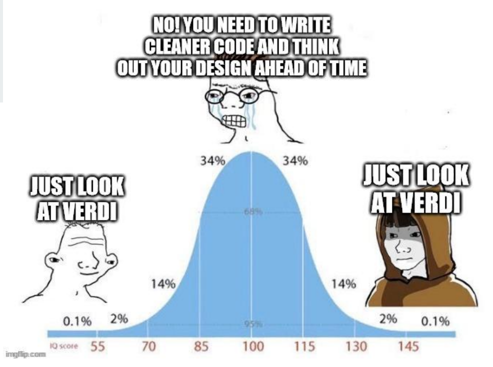
- Vibe check
  - How are you doing?
- Checkpoint 2 & tips to start
  - What are you doing?
- OoO processor structures
  - Do you know what's in your processor?
    - Common structures - ROB, reservation stations, CDB
    - ERR structures - RAT/RRF, free list (and backup)
- Multiplier integration
  - How do you hook it up?
- Frequency analysis
  - How do you make your processor go fast?

Checkpoint 2 Requirements

Requirements:

- Your design must support out-of-order execution of the arithmetic operations (you must implement ROB, RS, CDB, etc.)
  - Correctly runs `dependency_test.s`
  - Instructions writeback to ROB out of order when running `ooo_test.s`
- Integrate the provided multiplier into your processor as a functional unit
  - If a single multiply instruction from `ooo_test.s` commits correctly, you get full points
- Progress report + Roadmap
- Schedule a meeting with your mentor TA to demonstrate the above, you will need RVFI integrated to do so
  - RVFI is connected in the same way as mp_pipeline via `hvl/rvfi_reference.json`

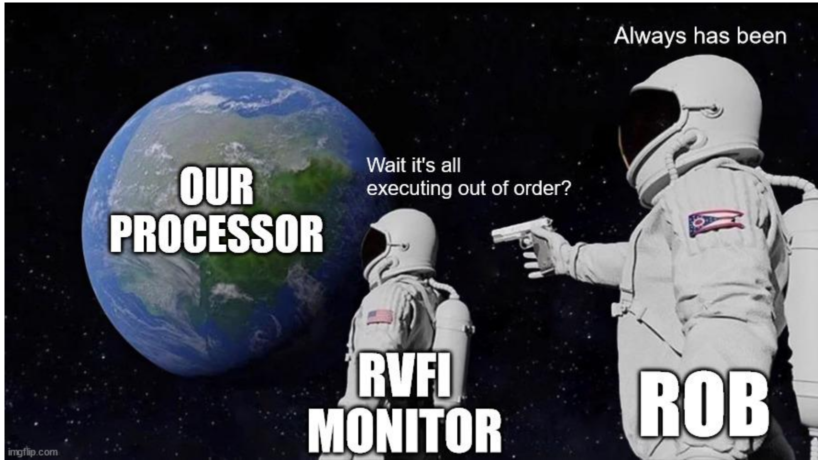
General Tips



- Implementing RVFI
  - Similar to mp_pipeline
  - Recommend making RVFI struct and populating + passing it with corresponding instruction
- Using structs & $bits() / $clog2()
  - Structs are convenient way to group together signals for a module
  - $bits() is a white-listed SV command that outputs the number of bits the input takes up
- Modular testing and flag-setting
  - Unit test modules!!! Write separate testbenches and test as you go
  - Use flags to turn on and off parts for testing
- Parametrization
  - Make as many modules parameterizable as possible
  - Good for finding optimal parameters like ROB size later on
  - Especially good if you plan to implement superscalar in the future
- Use types.sv!!!!

Common Out-of-Order Structures: The ROB

- Stands for Re-Order Buffer
- Circular buffer (these tend to show up a lot!)
- What does it mean if ROB is full? Empty?
- What is stored in a ROB entry?
    - Result ready / valid signal
    - ERR - destination register mapping
    - Tomasulo - destination register & computed value
    - RVFI data
    - Anything else?



Common Out-of-Order Structures: Reservation Stations

- Allows us to hold many in-flight instructions
- Can have small ones per functional unit (FU) or one large global one
  - Advantages? Disadvantages?
- What should we hold in it?
  - ROB index
  - Instruction information (opcode, destination register, etc.)
  - Tomasulo - holds register values
    - Read before issue architecture
  - ERR - holds physical register numbers
    - Read after issue architecture
  - RVFI data
  - Anything else?

Common Out-of-Order Structures: Functional Units

- Analog to the “execute” stage of an in-order pipeline
- Gets data from reservation station, sends data to an arbiter/on the CDB (next slide)
- Recommend at least two main functional units for CP2
    - ALU
    - Multiplier (will talk about this in a few slides)
    - Don’t need memory and comparison/branch units yet, but good to start thinking about
- Starting out
    - One reservation station, one functional unit. Just get an instruction through!
    - Add more and arbitrate between them

Common Out-of-Order Structures: The CDB(s)

- Delivers data from the FUs to the ROB (and other places that may need it!)
  - What else needs CDB data in Tomasulo? What about in ERR?
- Generally only one 'common' data bus for all FUs
  - What if multiple FUs finish at the same time?
  - Technically can have multiple buses…
  - Otherwise - arbitrate!
- What should we hold in it?
  - Valid/ready signal
  - ROB index
  - Output data
  - Destination register (arch. or phys.)
  - Tomasulo - Reservation station index
  - RVFI data

ERR-Specific Structures

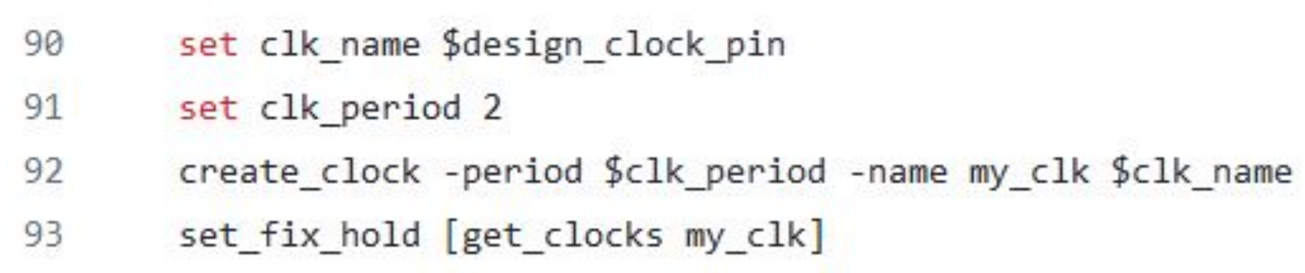
- RAT/RRF
  - Register Alias Table & Register Retirement File
  - RAT - speculative; RRF - precise state
  - Holds mappings for all architectural registers (32 in total)
  - Handling R0 instructions…
- (Backup) Free list
  - Split into speculative and architectural like with RAT/RRF
    - Can get clever and use only one with proper state rollback
  - Circular buffer (remember when we said it's common?) - like a ticket system
  - Contains a list of all available physical registers
  - Overflow/underflow?

Multiplier Integration

- Multiplier is provided on Campuswire and in the release repo
- **DO NOT MODIFY** for CP2
- Details about spec:
    - 3 multiplication modes, select using the `mul_type` input
    - Reset start flag after a multiplication is done (can't hold start high to issue multiple ops)
    - For signed * unsigned multiplication, A is signed port and B is unsigned port
- Need to modify decoder to support multiply instructions

Frequency Analysis

- Needed for all future CPs + for your report
- Frequency
    - Go to `synth/synthesis.tcl`
    - Decrement clock period (in ns) until synthesis fails
    - Can start low and adjust based on how bad your slack is



```
set clk_name $design_clock_pin
set clk_period 2
create_clock -period $clk_period -name my_clk $clk_name
set_fix_hold [get_clocks my_clk]
```

Thanks for coming!
Questions?