
# 3-Way Superscalar R10K Processor

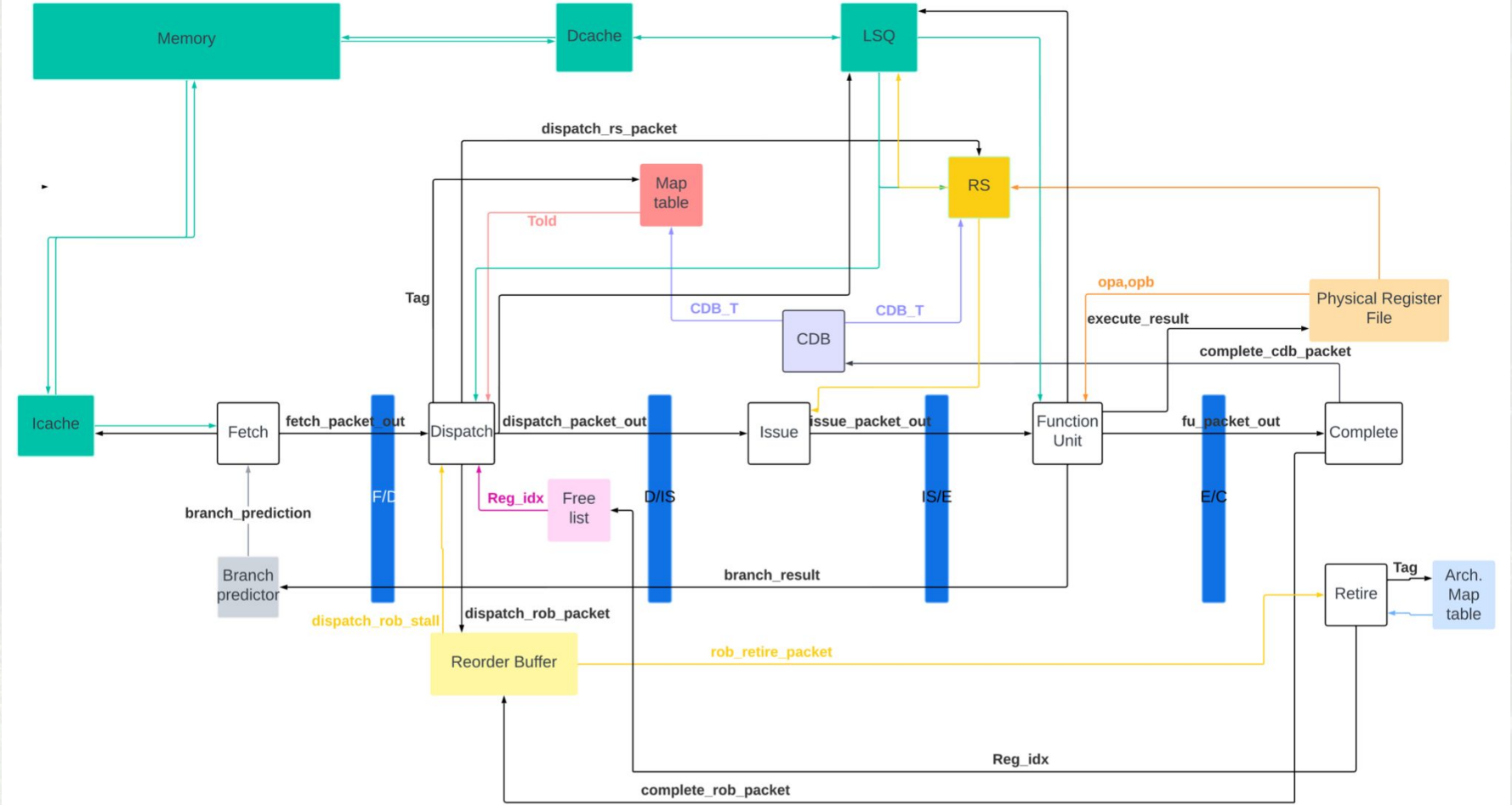
Group 9: Yan-Ru Jhou, Hsin-Ling Lu, Daniel Yu,
Hsiang-Yang Fan, Eli Muter

# Headline

- Introduction
- Design Overview
- Advanced Features
- Verification Strategy
- Final Results
- Teamwork

# Introduction

- Built the Out-of-Order R10K uarch processor based on VeriSimpleV RISC-V
- Already implemented advanced features like
  - 3-way superscalar execution to increase level of instruction level parallelism.
- **Tried** to implement advanced features including
  - Support not only 3-way but arbitrary N ways


Memory
Dcache
LSQ
dispatch_rs_packet
Map table
RS
Told
Tag
CDB_T
CDB
CDB_T
opa,opb
Physical Register File
execute_result
complete_cdb_packet
Icache
Fetch
fetch_packet_out
Dispatch
dispatch_packet_out
Issue
issue_packet_out
Function Unit
fu_packet_out
Complete
F/D
D/IS
IS/E
E/C
Reg_idx
Free list
branch_prediction
Branch predictor
branch_result
Retire
Tag
Arch. Map table
dispatch_rob_stall
dispatch_rob_packet
Reorder Buffer
rob_retire_packet
Reg_idx
complete_rob_packet

# Design Overview

| Reservation Station(RS) | Reorder Buffer(ROB) | Physical Registers(PR) | Functional Units(FU) |
|---|---|---|---|
| 16 entries | 32 entries | 64 registers | 3 ALUs, 2 Mults, 1 Branch, 2 Load |

# Design Overview

| I cache | Store Queue(SQ) | D cache |
| --- | --- | --- |
| Direct mapped<br>32 lines, 8 byte per line | 8 entries<br>with load forwarding | Direct mapped<br>32 entries/lines, each 8 block |

# Advanced Features

- Completed
  - 3-Way superscalar
- Under development
  - N-Way arbitrary superscalar
    - Use macro SUPERSCALAR_WAYS to abstract N-way superscalar execution, avoiding hard coding 3-way superscalar
    - Currently all modules except the FU, the LSQ, and the cache support N-Way arbitrary superscalar

# Verification Strategy

- Unit Testing
  - Verified each individual modules with comprehensive testbenches
- For Integration Testing
  - Progressive Verification Strategy
  - First began testing fetch, fetch <-> dispatch, fetch <-> dispatch <-> issue, ..., and finally pipeline_testbench including all modules

# Auto Testing

- We wrote 2 scripts to auto test our pipeline's output, "testSim.sh" and "testSyn.sh"
  - Compare the pipeline's output to that of the P3 pipeline for each public test case, as well as some we wrote ourselves
  - Require an exact match for the writeback.out
  - Require a match for lines in program.out beginning with "@@@"
    - This checks the final memory state as well as the stop condition of the pipeline (wfi halt, illegal instruction, etc.)

# Final Results

- Public Testcases
  - Passed halt and mult_no_lsq
- Overall CPI
  - Halt: 7
  - mult_no_lsq: 2.54

# Teamwork (Equal Contribution)

- Yan-Ru Jhou*: Reservation Station, Decoder, Precise State, Pipeline and testbench.
- Daniel Yu*: Map Table, Physical Register File, Issue Stage, Function Unit
- Hsin-Ling Lu*: Freelist, Complete Stage, Store Queue
- Hsiang-Yang Fan*: Fetch Stage, Retire Stage, Architecture Map Table, Icache, Dcache
- Eli Muter*: Dispatch, ROB , verification, advanced features

# Q&A

Content

# Thank you for listening

Content