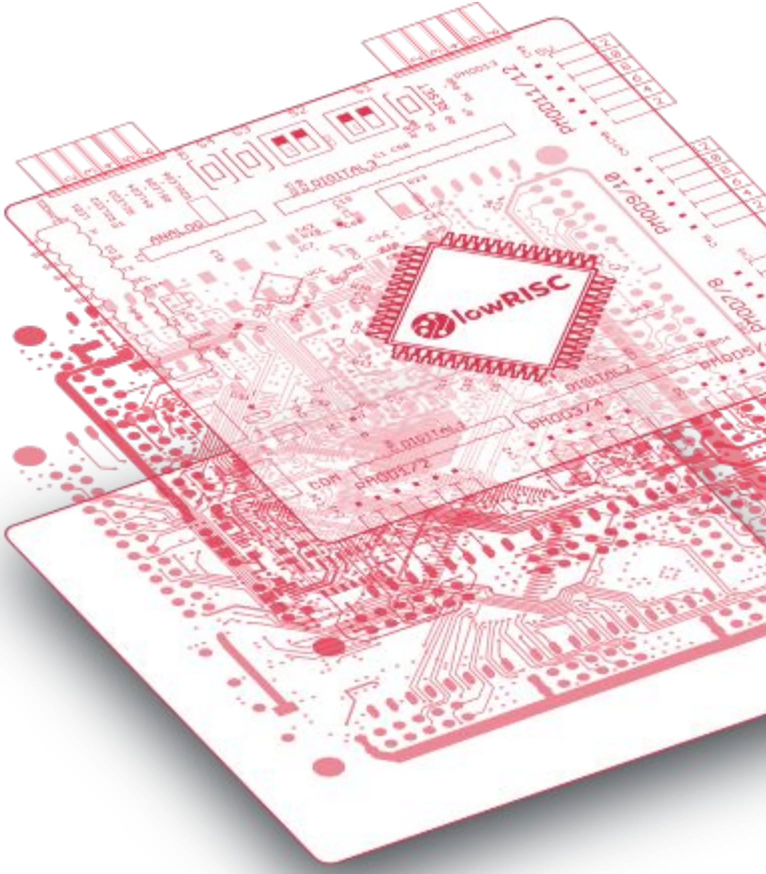
Improving Ibex Performance

lowRISC

Greg Chadwick
RISC-V Devroom FOSDEM 1st February 2020



Ibex

- Microcontroller class CPU with two stage pipeline
- 32-bit RISC-V IMC/EMC with M-Mode, U-Mode and PMP
- Written in SystemVerilog
- Initially developed as Zero-riscy as part of the PULP platform by ETH Zurich
- Now developed by lowRISC, a not for profit company building open source silicon through collaborative engineering
- Used by the recently announced OpenTitan, an open source silicon root of trust

Improving Performance

- Aim to reduce total cycles to execute Coremark and Embench
- Need to be careful about optimising for the benchmark only
- Analysis of execution provides a useful guide for what to improve
- Must consider how applicable improvements will be to code that isn't benchmarks
- Planned improvements will be configurable options
    - Choose a smaller/simpler Ibex or a faster one

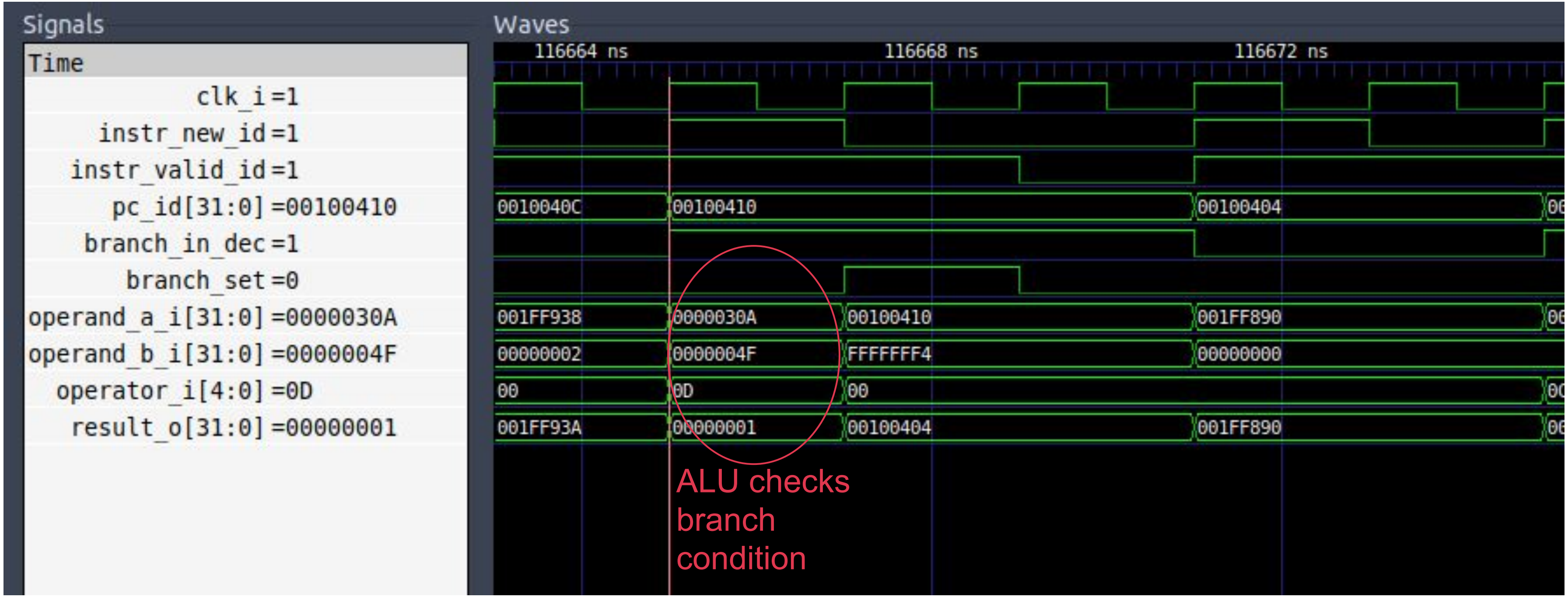
Trial System

- Simulate Ibex with Verilator
- Dual ported memory containing code and data
- Single cycle memory access latency
- Reasonable analogue of a best case ‘real’ system

Analysis Techniques (1)

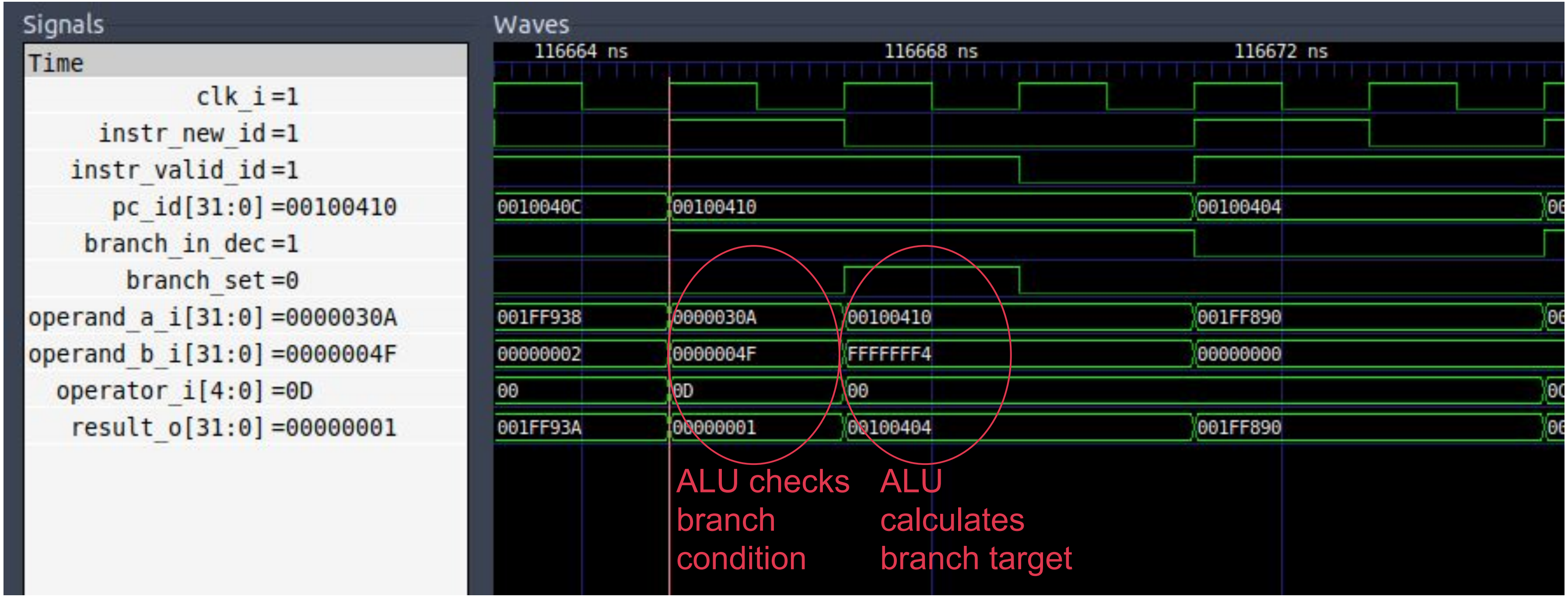
- Run the benchmark
- Trace the simulation
- Examine trace in GTKWave
  - Look at signals indicating top-level stall
  - Choose a few points to examine why stall is occurring
- No quantitative analysis but quick and easy way to survey what kinds of things are slowing down execution

Trace in GTKWave, Branch Stall



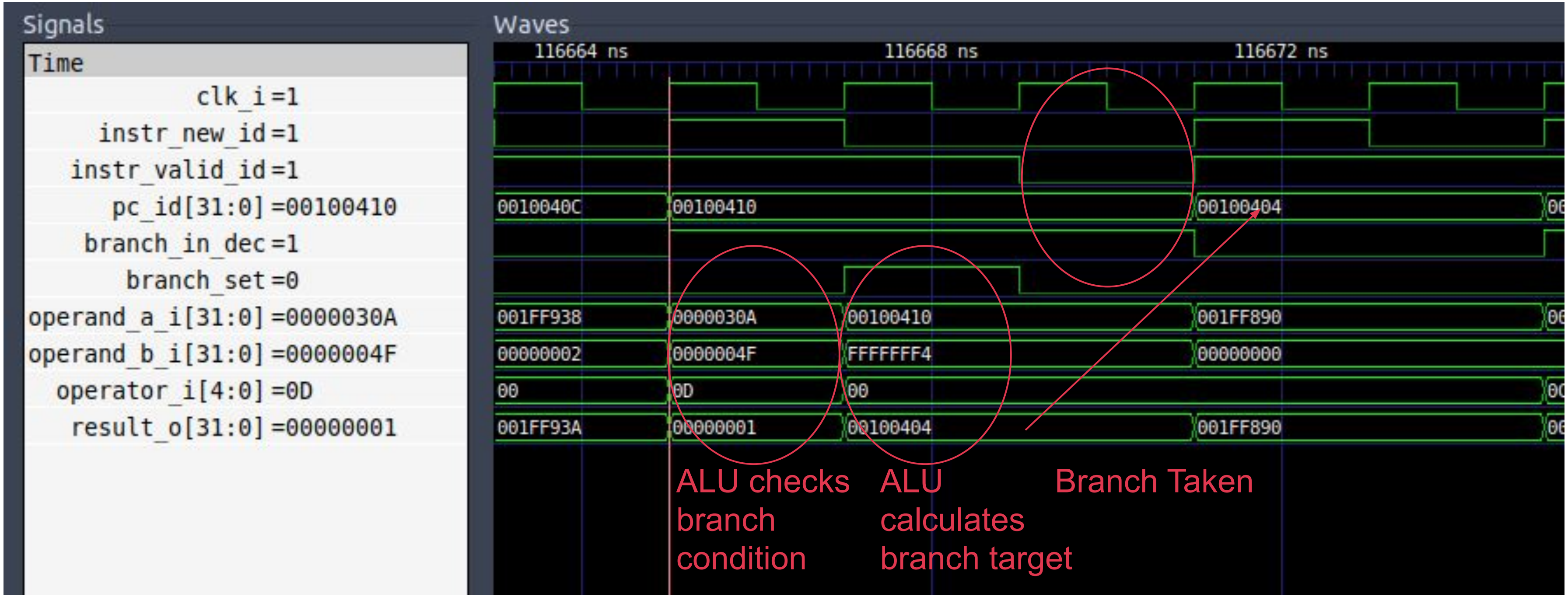
bne t2,s5,100404

Trace in GTKWave, Branch Stall



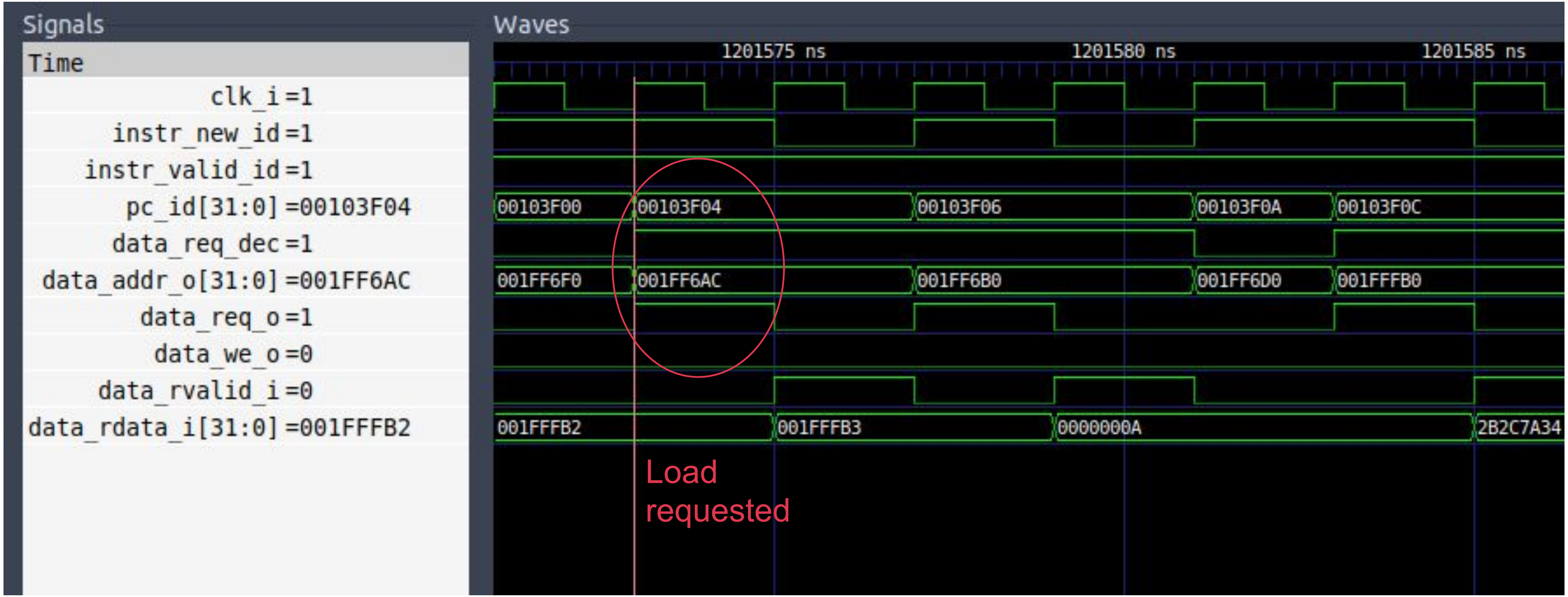
bne t2,s5,100404

Trace in GTKWave, Branch Stall



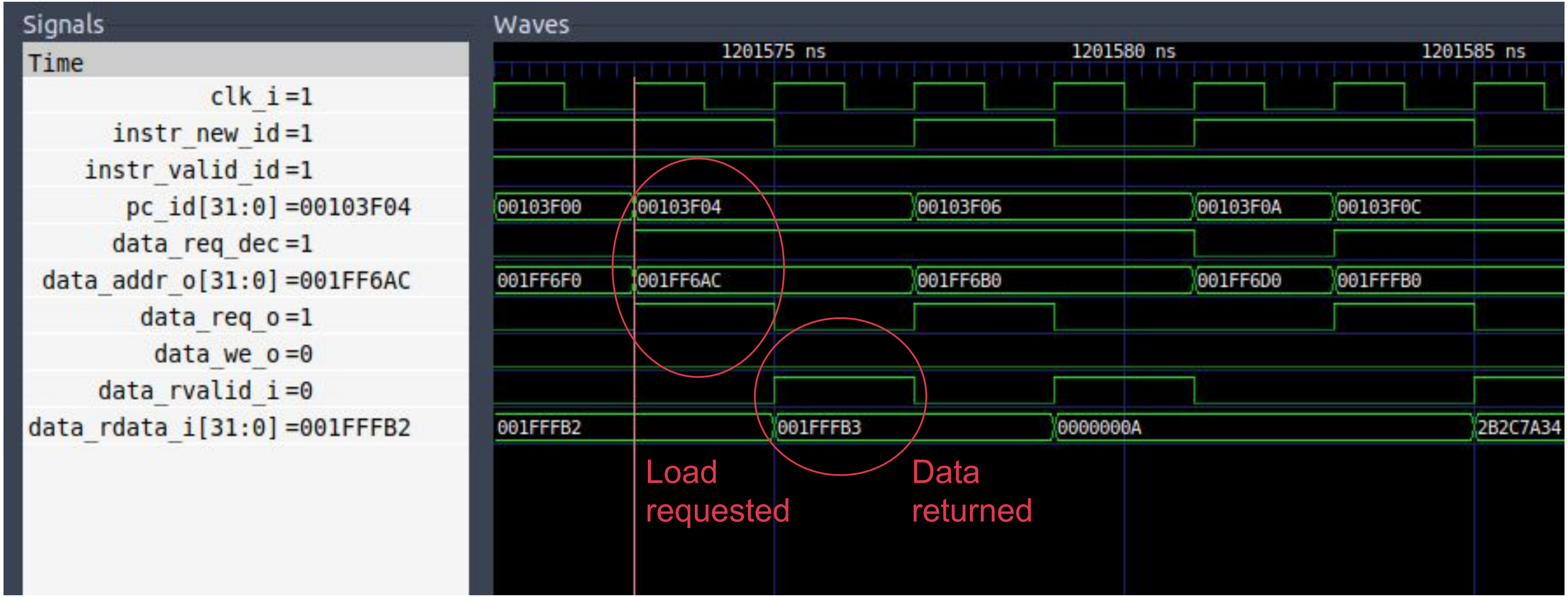
bne t2,s5,100404

Trace in GTKWave, Load Stall



lw t3,12(sp)

Trace in GTKWave, Load Stall

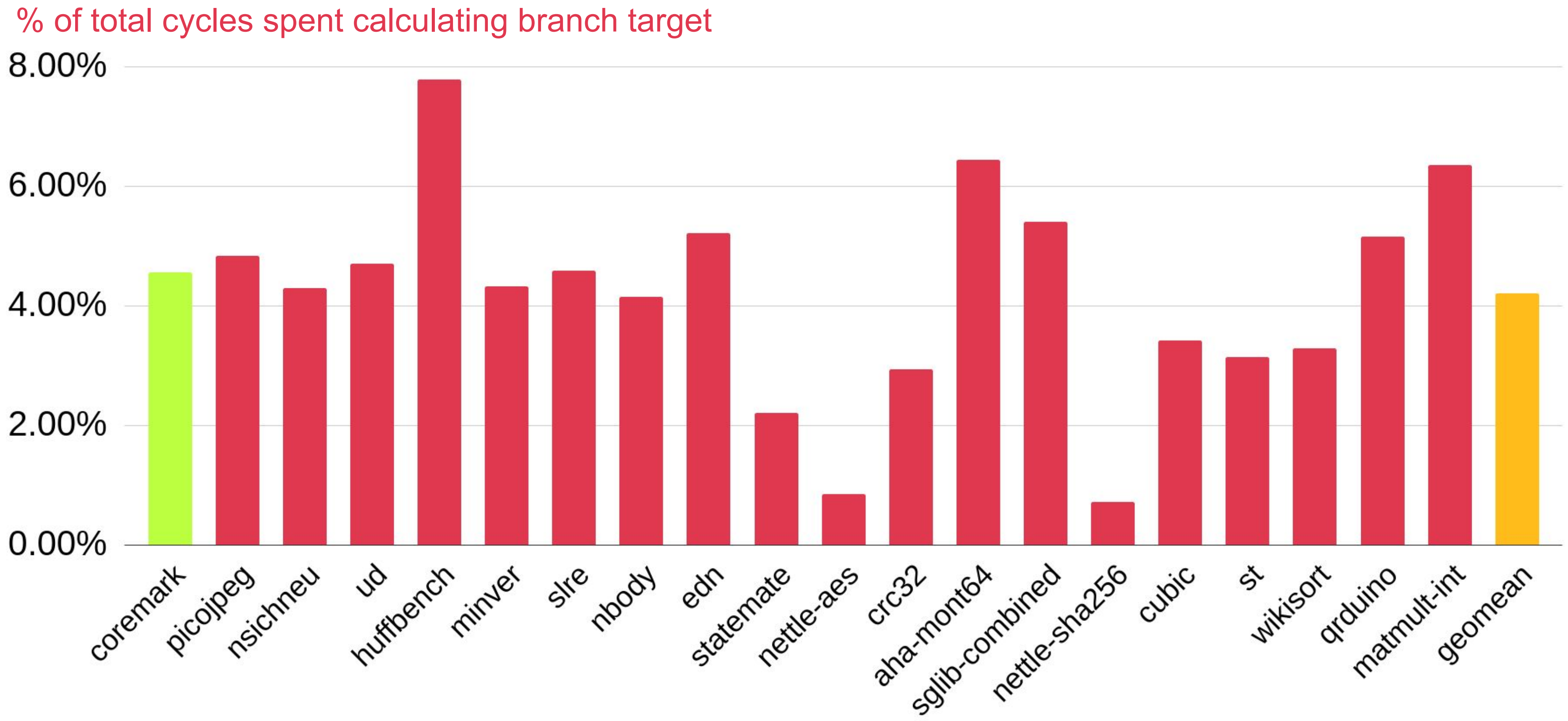


lw t3,12(sp)

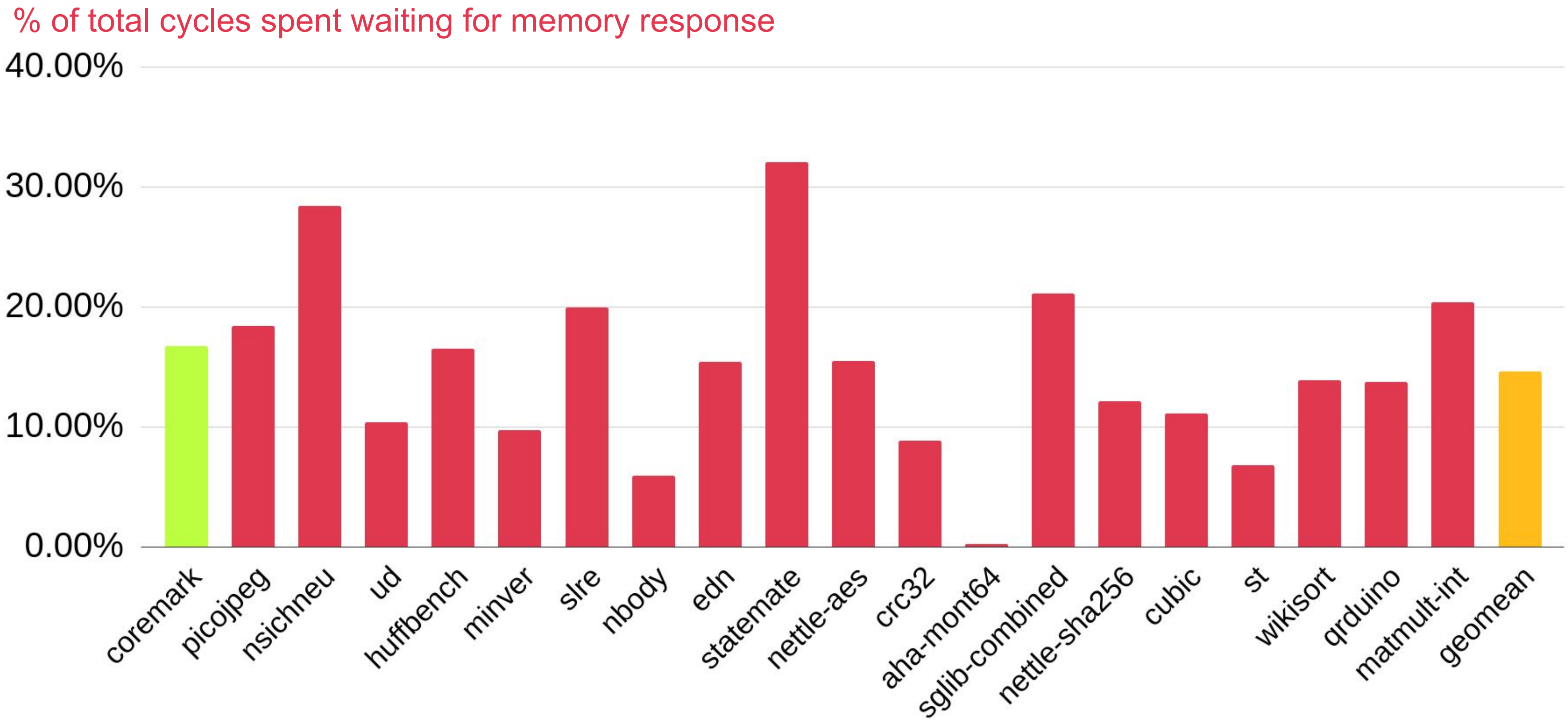
Analysis Techniques (2)

- Log performance counters after benchmark run
- Use previous survey to decide on interesting things to count
- Examine with spreadsheet to produce quantitative data on effect stall conditions from informal survey have on performance

Branch Stall %

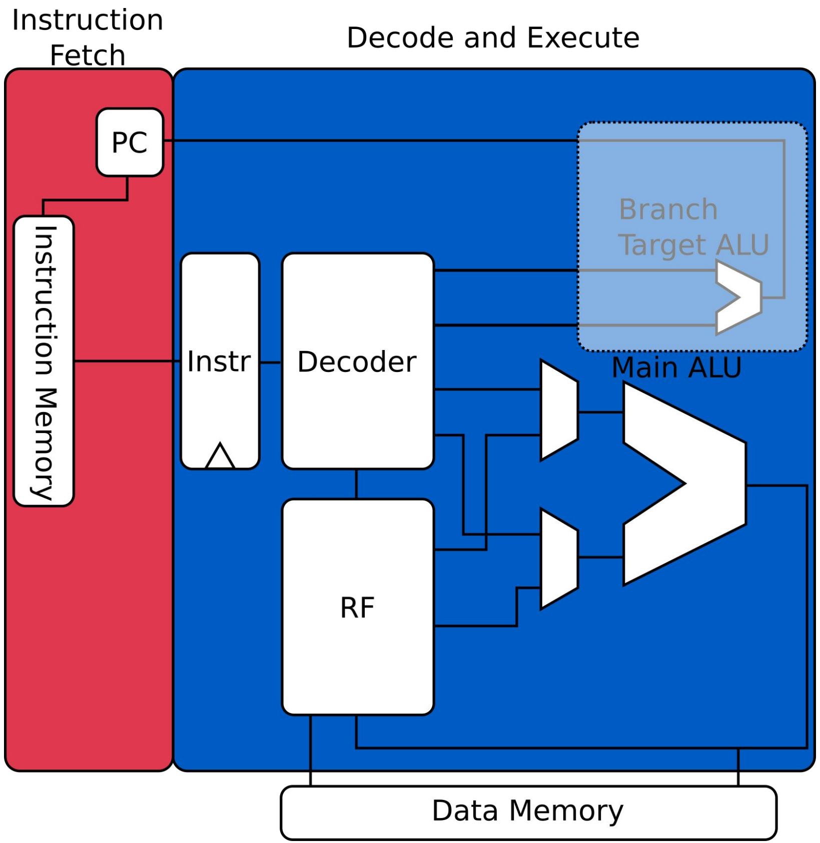


Memory Stall %



Branch target ALU

- Add second ALU to calculate branch targets
- Compute branch target and branch condition in parallel
- Minor area increase for ~4% performance gain



Implementation Trials

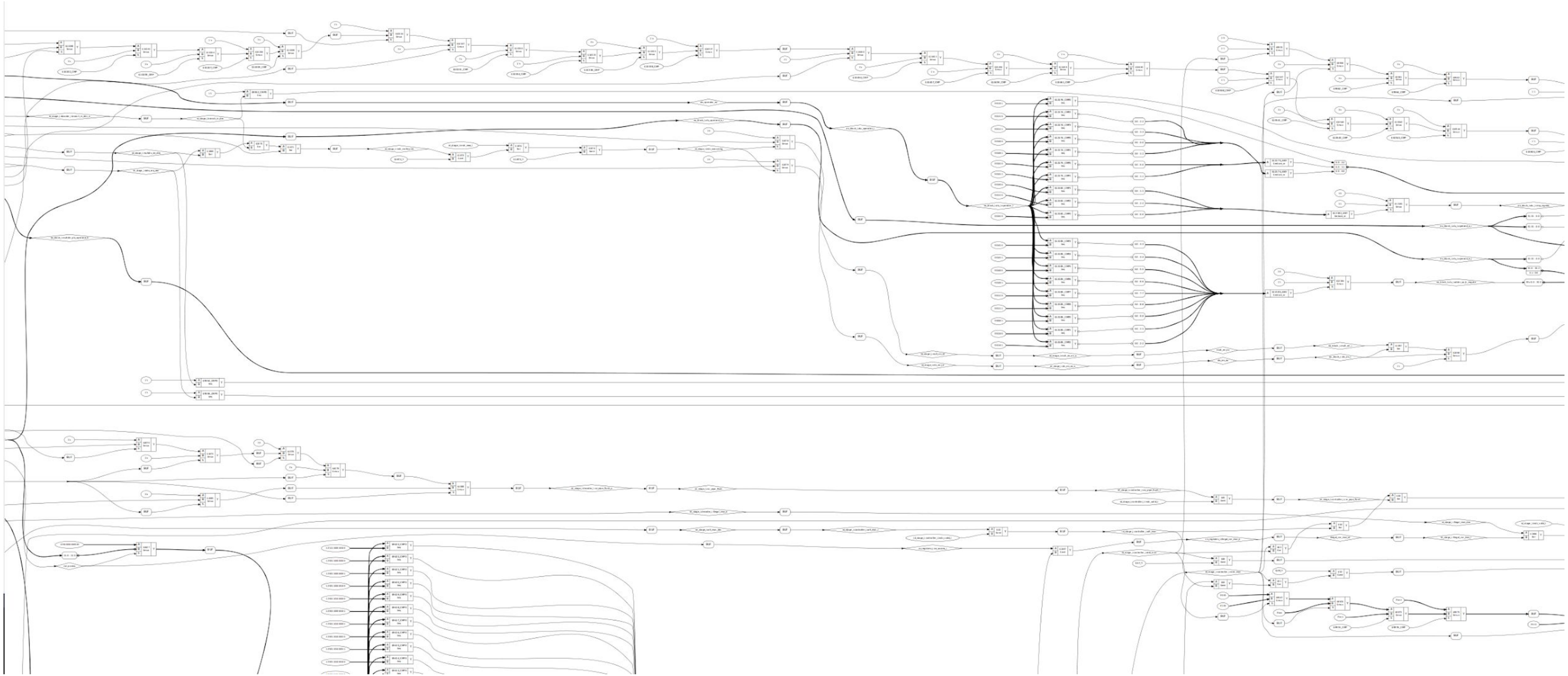
- Need to check impact of change on frequency and area
- Built experimental synthesis flow using Yosys with Timing Analysis via OpenSTA
- Using the nangate 45nm library available from the OpenROAD repository
- Better numbers likely achievable with commercial tools and library
    - Flow used to see relative changes and areas of timing pressure

Branch Target ALU Implementation Results

	Base	Branch Target ALU	% change
Coremark/MHz	2.40	2.51	+4.5 %
Area	27,345 μm^2	27,666 μm^2	+1.2 %
Fmax	269 MHz	234 MHz	-13.0 %
Coremark	645.6	587.3	-9.0 %

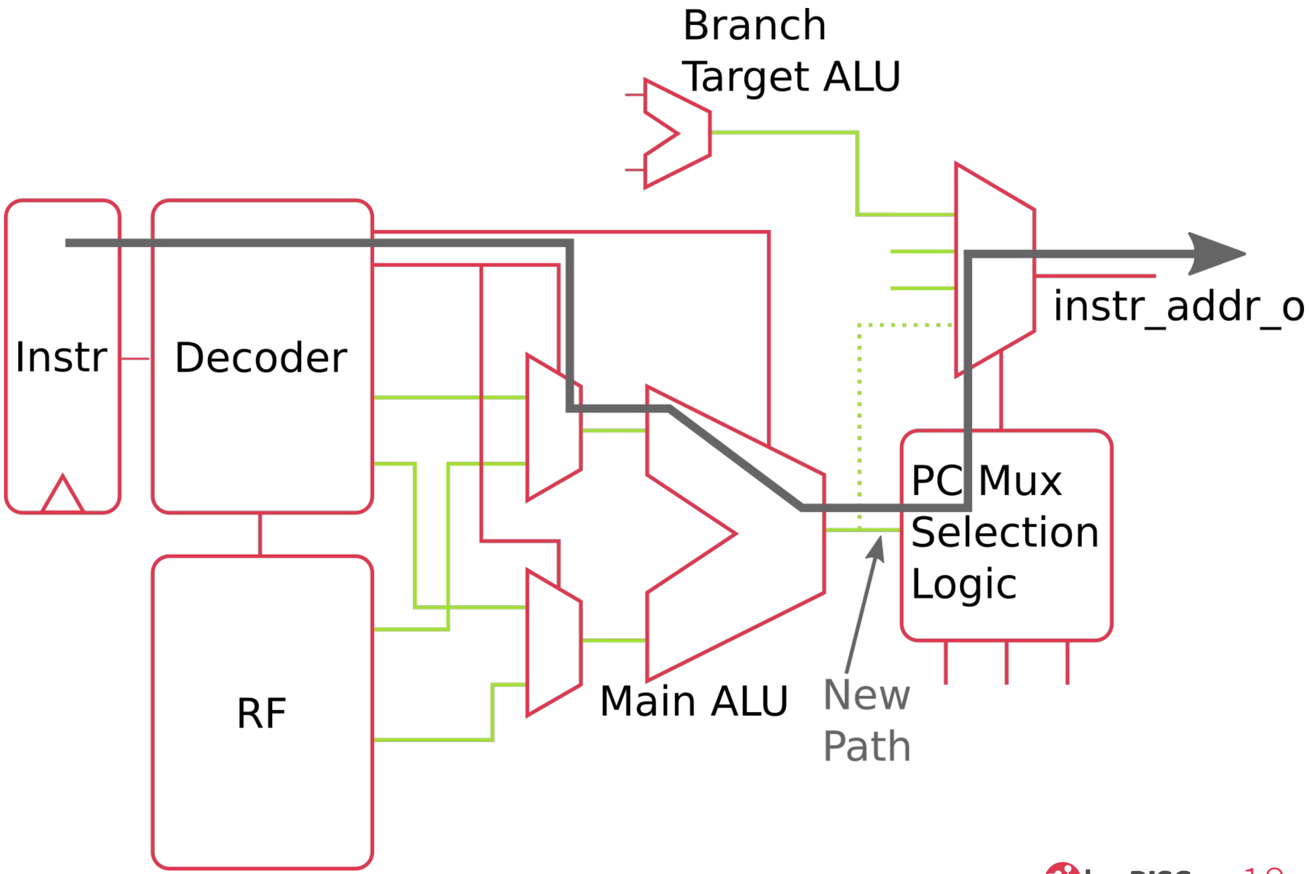
- Adding in branch target ALU reduced maximum frequency
- Overall worse performance at Fmax (but better per MHz)
- What can we do about it?

Can you spot the problem (1) ?



Can you spot the problem (2) ?

- Previously the branch decision was stored in a flop after being computed by the main ALU
- Now it's being fed straight in the PC Mux select
- Main ALU result used to feed into PC selection mux (as it computed the target), which was the worst path
- It now goes via extra logic into the select
- So worst path has got longer



How Do We Fix It?

- Need main ALU result earlier
- Key issue is selects for ALU operand mux, provided by the decoder
- Decoder complex blob of logic, so outputs not as early as we like
- Make the ALU operand mux select outputs earlier from the decoder and we can solve the problem

Instruction Flop Fan-Out

- Instruction flop in ID/EX has a large fan-out
  - Meaning it feeds its data to many different gates
- Requires buffering to ensure it can drive everything it connects to
- Reduce required buffering by duplicating it
- Split decode to decide ALU operand select and operation from duplicated register
- Decode all other control from other register

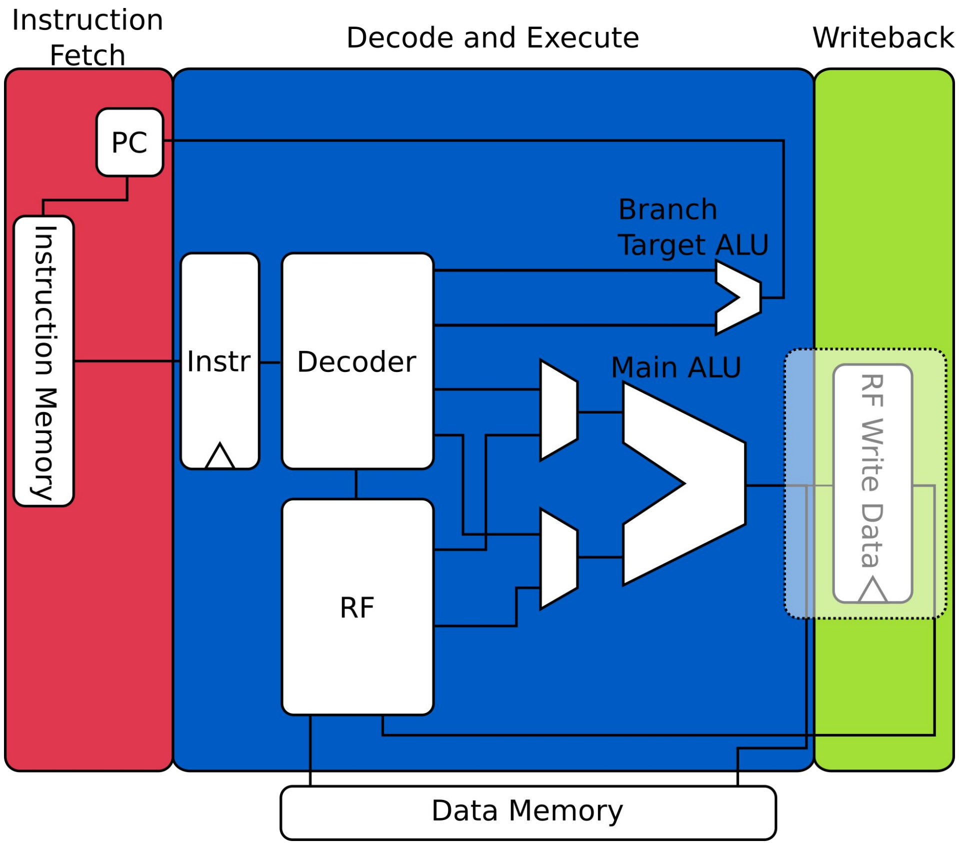
Improved Branch Target ALU Implementation

	Base	Branch Target ALU	% change
Coremark/MHz	2.40	2.51	+4.5 %
Area	27,345 μm^2	27,579 μm^2	+0.9 %
Fmax	269 MHz	250 MHz	-7.6 %
Coremark	645.6	627.5	-2.8 %

- Slightly better area due to reduced buffering
- Still haven't restored Fmax
  - Yosys/ABC doesn't take IO timing constraints into account
  - So doesn't optimise worst path properly
  - May not want to run at Fmax anyway

Writeback Stage

- Add a third pipeline stage, writeback which holds the value to be written to the register file
- Load data from memory writes direct to the register file
- Drops a stall cycle for loads & stores as response only needed the cycle after ID/EX
- Greatly Simplified Diagram!
    - Significant new stalling and hazard logic needed



Writeback Implementation

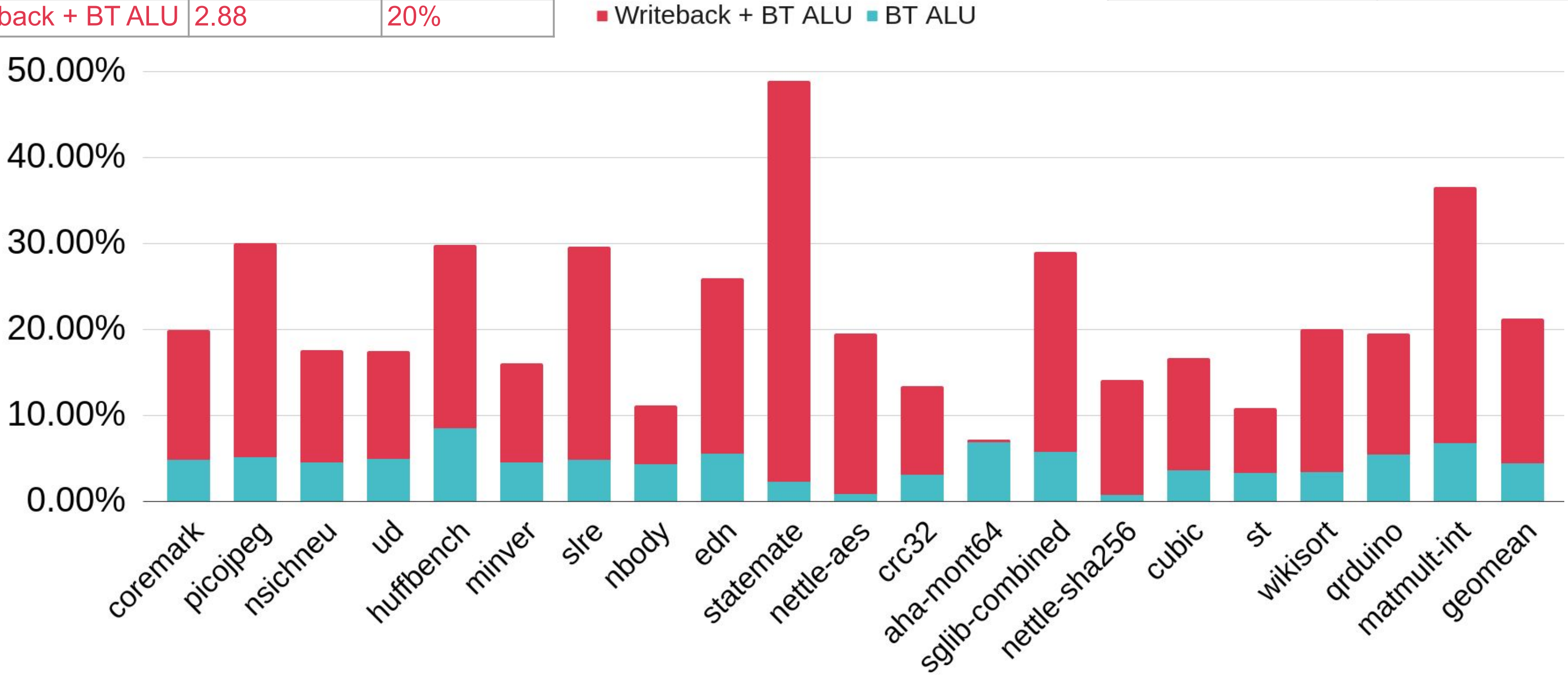
	Base	Writeback + BT ALU	% change
Coremark/MHz	2.40	2.88	+20.0 %
Area	27345 μm^2	29212 μm^2	+6.8 %
Fmax	269 MHz	253 MHz	-6.3 %
Coremark	645.60	728.64	+12.9 %

- Notable area cost
  - Outweighed by performance gains
- Little change in Fmax from BT ALU implementation
  - Worst case path from BT ALU change still dominates

Overall Speedup

	Coremark/MHz	Speedup
Base	2.40	-
BT ALU	2.51	4.5%
Writeback + BT ALU	2.88	20%

	Geomean Speedup
BT ALU	4.42%
Writeback + BT ALU	21.3%



Find Out More

- Check out the Ibex repository www.github.com/lowRISC/ibex
- Third pipeline stage + benchmarking infrastructure not yet in main repository
  - See my ‘ibex_fosdem’ branch at www.github.com/GregAC/ibex to take a look
- See the lowRISC website at www.lowrisc.org
  - Now recruiting!
- My email: gac@lowrisc.org