Runahead Processor

Finale Doshi
Ravi Palakodety

Outline

- ***Motivation***
- High Level Description
- Microarchitecture
- Results
- Conclusions

Where We Left Off…

- Lab 3 – Building a 4-stage pipelined SMIPS processor
- Critical Path – **Load-α**
  - Fetch → Decode → Execute → **ReadDataCache** → Writeback
- Data Cache Miss?
  - Stall until data returns from Main Memory

A Baaad Example

- **Ld-α, Ld-β, Ld-γ, …**
- If latency = 100 cycles from main memory to cache, then:
  - Initiate **Ld-α** request
  - Stall for 100 cycles
  - Initiate **Ld-β** request
  - Stall for 100 cycles
  - And so on…

Key Insight

- "Runahead" to see whether there are memory accesses in the near future
- With an instruction sequence **Ld-α, Ld-β**
  - Initiate memory request for **Ld-α**
  - Continue execution
  - Initiate memory request for **Ld-β**

Outline

- Motivation
- ***High Level Description***
- Microarchitecture
- Results
- Conclusions

DataCache Miss Occurs...

- Backup the register file
- Keep running instructions
- Use **INV** as the result of any ops that:
  - Are DataCache misses
  - Depend on calculations involving DataCache misses

Data Returns..

- Cache is updated from MainMem:
- Restore the register file
- Rerun the original "offending" instruction

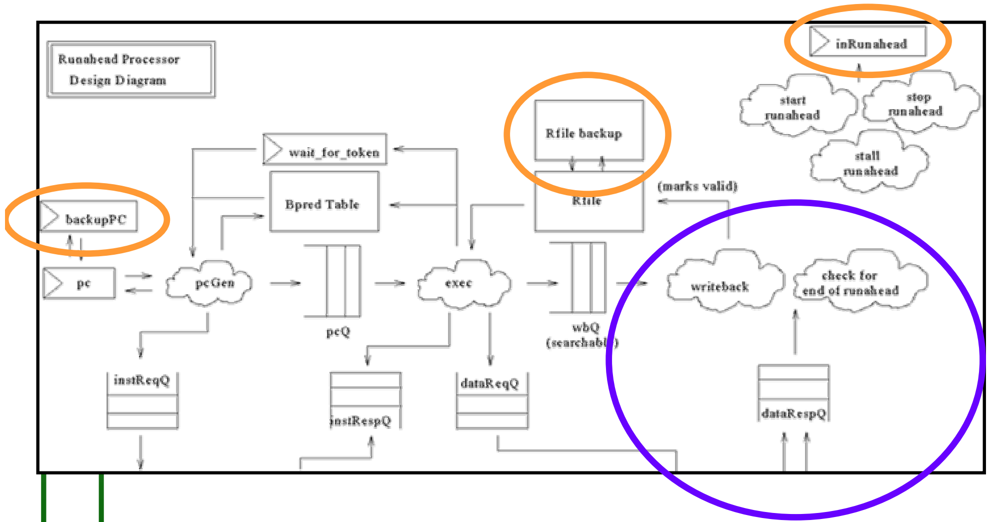
Follow the Rules

- Do NOT
  - Update the DataCache while in Runahead mode
  - Initiate Memory Requests that depend on **INV** addresses
  - Branch when predicate depends on **INV** data
  - Initiate Memory Requests that cause collisions in DataCache

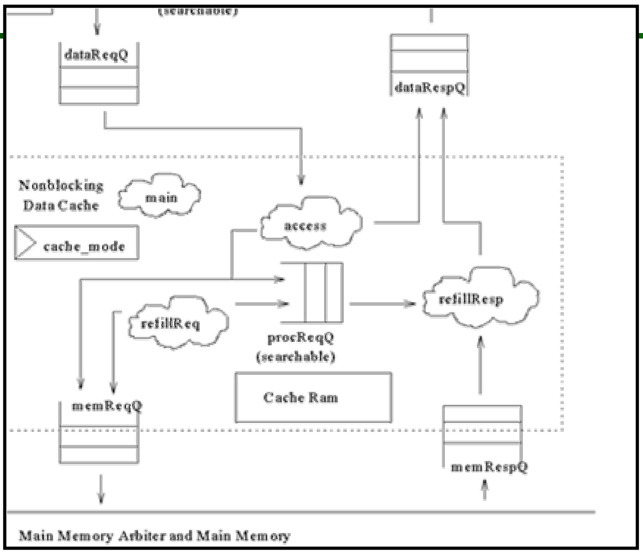
Outline

- Motivation
- High Level Description
- ***Microarchitecture***
- Results
- Conclusions

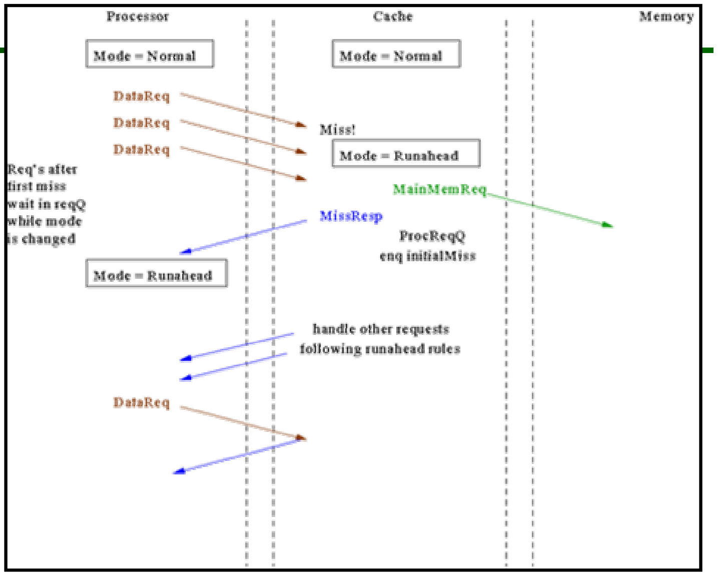
Processor Side



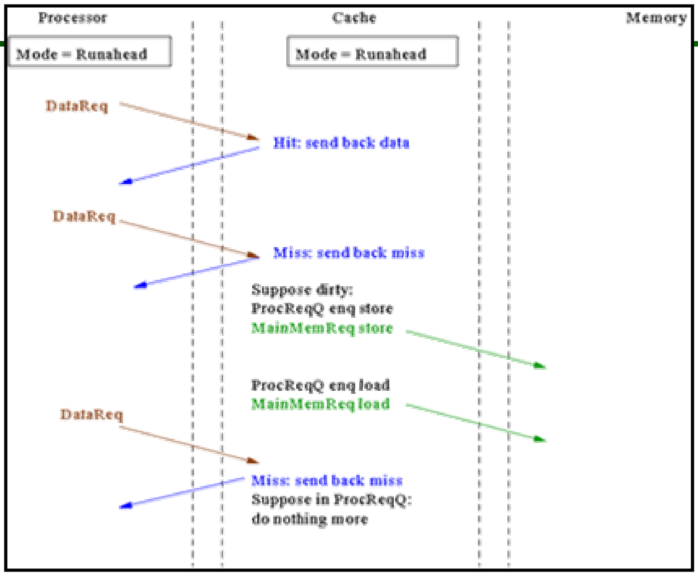
Cache Side



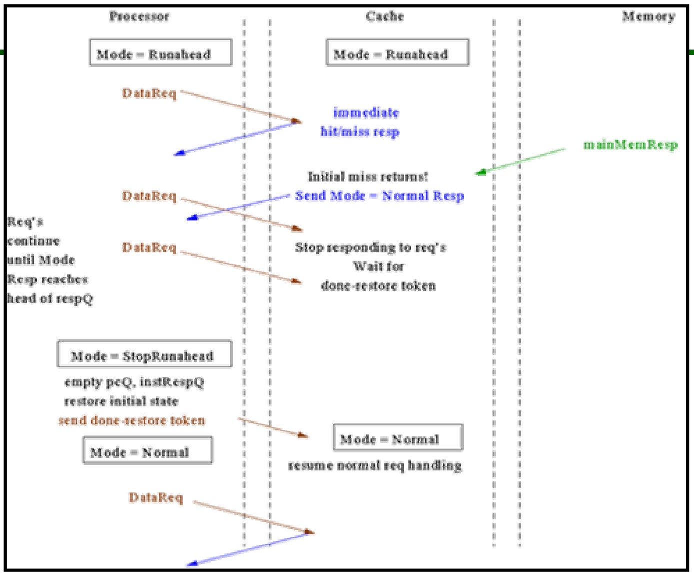
Execution - Enter Runahead



Execution - In Runahead



Execution - Exit Runahead



Design Explorations

- Store Cache Optimization
- Decisions when to exit runahead

Store Cache

- **Ld-α, St-β, Ld-β**
- Rather than return **Ld-β** as **INV**, return the value that was just stored.
- Use 4-entry table, as in Branch Predictor

When to Exit Runahead?

- When the “offending” miss returns? OR
- When all memory requests that are currently in-flight are processed?

Outline

- Motivation
- High Level Description
- Microarchitecture
- ***Results***
- Conclusions

Key Parameters

- Vary Latency of Main Memory
  - As the latency increases, the impact of runahead becomes more significant
  - At small latencies, the penalty for entering/exiting runahead can reduce performance

Key Parameters

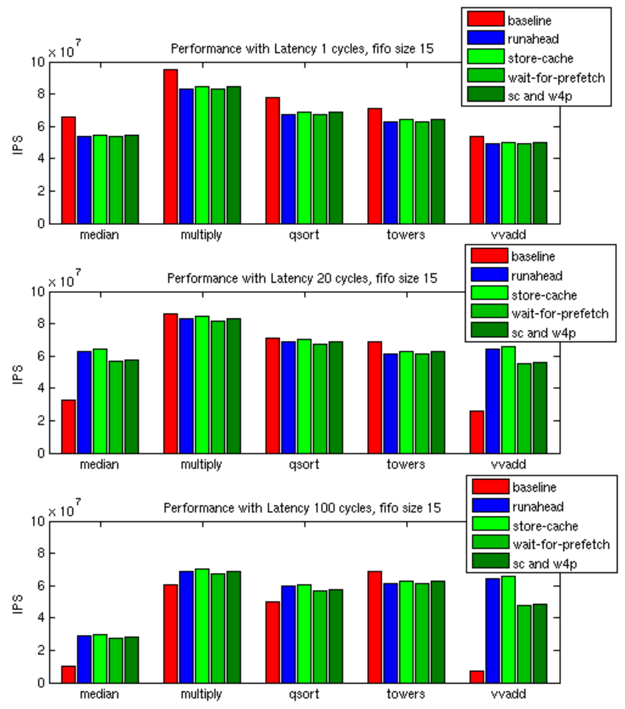
- Vary Size of FIFOs
  - As the FIFOs get larger, the processor is able to run further ahead and generate more parallel memory requests.
  - As the FIFOs get larger, the penalty for exiting runahead becomes more severe.

Testing Strategy

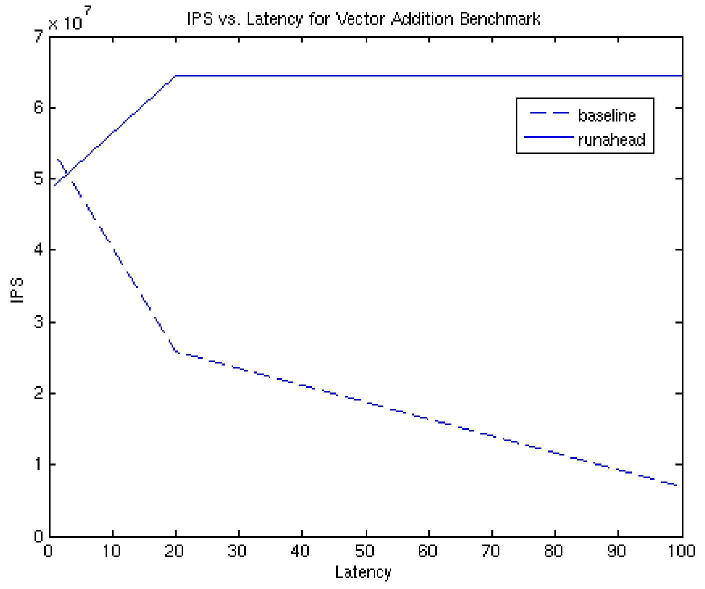
- Latencies of 1, 20, and 100 cycles
- Fifos of length 2, 5, 8, 15
- Standard benchmarks; focus on vvadd

We'll focus on length 15 fifos here since they allowed for the most extensive runahead.

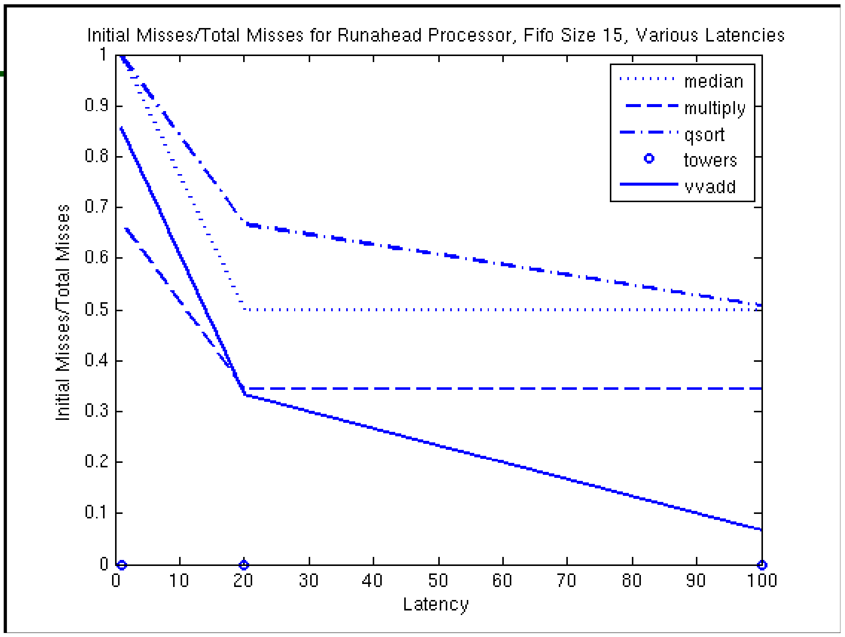
Results



Results



Results



Outline

- Motivation
- High Level Description
- Microarchitecture
- Results
- ***Conclusions***

Conclusions

- Runahead is good.

Conclusions

- Runahead is a cheap and simple way to improve IPS.
- The enter/exit runahead penalty is small enough that the IPS is always comparable to the Lab 3 processor.
- The control structure is (fairly) straightforward, with most improvements done on the cache side.

Extensions

- Aggressive Branch Prediction
  - Don’t stall when branch predicate is **INV**
- Save valid runahead computations
- Aggressive Prefetching
  - Predict addresses for Ld, St, when the given address is **INV**.