EECS 470 Final Project

Group 1
Shibo Chen, Zhen Feng, Chin-wei Hsu, Yueying Li, Wenhao Peng
Outline

- Overview
- Design and Analysis
  - Branch
  - Reservation Station
  - Execution
  - DCache
  - LSQ
- Conclusion
Branch Resolver

Conditional Branch:
Tournament Predictor

Unconditional Branch:
Return?
1. Return Address Stack
2. Branch History Table
3. Speculatively Return on Reg

Not Return?
1. Branch History Table
2. Speculatively Jump on Reg
333% Prediction Rate Increase On Average
We decide to use 32-entry Branch Predictor and BHT
Early Branch Resolution for instruction fetch

Without this feature:

Fetch the this one plz

No problem,

But plz wait 10 cycles
Early Branch Resolution for instruction fetch

You probably want to fetch this one

No problem, Here you are…
4% overall performance boost
Branch Prediction Analysis of Early Branch Resolution for Instruction Fetch

- Without Partial Early Branch Resolution
- Our Design
- Difference
Reservation Station + FUs
RS Design choices

• generic RS unit
  • Every types of instruction can be put in any RS unit
  • 8 units. Increase from 8 to 16 not improve IPC. 8 shows 25% shorter clock period after synthesize

• Issue logic
  • Issue as instruction types : branch > ld or st > mult > alu

• Prevent CDB structure hazard
  • Decrease issue number according to how many load or or multiplier will complete next cycle, other instructions will finish in one cycle
EX design choice

• 1 ld or st address calculator : goes into LSQ
• 1 branch calculator
• 2 4 stage pipeline multiplier
• 3 alu

• Calculate how many mult will complete next cycle from 3rd stage in multiplier
• Get if load will complete from LSQ
• At most 3 instruction will complete in one cycle
Data Cache
Design choices

• $2^N$ set associative with LRU policy
  • $N$ is a tuning knob that allows faster clocks or higher hit rate

• Write-through and write-on-allocate
  • For easy data structure: always coherent with memory

• Invalid load address masking
  • A bandpass filter allowing only valid addresses through
  • Invalid load requests will be flushed by branch mispredict
    • Returned deadbeef as data to LSQ
    • Saves many cycles and prevents unnecessary eviction
Reality

• DCache lies on the critical path across the processor
  • Data from memory needs to find a best place to settle

• Higher associativity gives diminishing CPI returns
  • Yet it adds to the load on memory bus, potentially increases CLK period

• Went with 2-way set associative
  • Although 4-way does give a somewhat better CPI (within 10% at best)
LSQ
Dispatch

• ST put into SQ when dispatch
  • Move tail
  • Set valid bit

• Only 1 LD allowed (no LQ, no speculate)
  • Give age to RS when dispatch
  • ld_busy to tell RS don’t issue LD
  • LD has to be in-order (if age is different)

• SQ size = 8
  • We found that it rarely is full
  • Compare 64-bit addr is heavy so we don’t want to make the SQ too large
  • If it is about to full (empty < 3), then tell dispatch to stall

<table>
<thead>
<tr>
<th>rob_idx</th>
<th>addr</th>
<th>value</th>
<th>valid</th>
<th>addr_ready</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQ structure
EX stage

• When ST comes in
  • Find the entry with same rob_idx
  • Set addr, value, addr_ready
  • Complete in 1 cycle

• When LD comes in
  • First set ld_busy_next = 1, save ld_addr and other info for completion
  • If (ld_busy)
    • First check if all SQ addr between its age and head are ready
    • If ready
      • Find matched addr: forward the value and complete, total 2 cycles
    • No matched: ask D$ and wait until it comes back
      • If D$ hits: D$ need 1 more cycle, so total 3 cycles
      • If D$ misses: needs many cycles
Retire and mispredict

• ROB tells how many ST should be retire
• Retire up to 1 ST each cycle
  • Give addr and value to D$

• If branch mispredict
  • tail_next = head + retire_num
  • Clear ld_busy
  • Ignore the comeback D$ valid until next time when LSQ ask D$ to load.
Conclusion and Recommendations

- 3-way superscalar
- 11.3 ns
- average IPC: 0.947

- Critical Path: LSQ <-> Memory.
  - Try to add more registers
  - Avoid large unstaged combinational logic
Work assignment

- Shibo Chen: Branch, ICache, Pipeline, ROB, Debugger
- Zhen Feng: Reservation Station, EX and FU, Pipeline
- Chin-wei Hsu: LSQ, Pipeline, Debugger, RS test, MapTable and Freelist
- Wenhao Peng: Script, Pipeline, ROB test, DCache, EX and FU, CDB
- Yueying Li: Dispatch, DCache, RS test, Branch, Pipeline
Thank you for listening. Any question?