Extending the 16 GPR standard beyond RV32E
Motivation

- Register file area/power/latency critical to any processor
  - Area critical in embedded devices
  - Power and latency critical for highly-ported, high-performance architectures
    - >20% core power on high-performance out-of-orders
    - Exacerbated by industry move to non-custom, synthesizable designs
- Large register files help with static code scheduling
  - Great for simple in-order and VLIW CPUs
  - Modern wide-issue out-of-orders have the potential to hide register spill latency and issue bandwidth
    - Becoming ubiquitous even in deeply embedded applications
    - Benefit of >16 architected registers debated

Source: Andrew Waterman's Thesis
Motivation II

• Reduced architected register files
  • Demand extends beyond integer only-codes
    • Market existence proof (ARM micros)
    • Greatly extends the viable range of potential energy-efficient, multi-threaded out-of-order CPUs
      • Small, dual-threaded embedded (~20 instruction windows)
      • Large, many-threaded application processors
  
• RISC-V relevant: Often reduce call/interrupt overhead and enable more frequent use of compressed instructions

• Ask: Make 16GPRs orthogonal like other ISA variants
Methodology

- Reduced available registers to x0-x15, f0-f15
  - Mostly unmodified calling conventions
  - System call number in x5 instead of x17
  - Referring to it as RV64R for this presentation
    - R is for “reduced”

- Toolchain + Simulator
  - GCC + glibc (riscv-next branch)
  - -O2 -falign-jumps=16 -falign-jumps=16 -falign-functions=16
  - Gem5
    - 2-wide in-order ~A8
    - 3-wide out-of-order ~A57

- Benchmarks
  - Coremark/Dhrystone/Whetstone
  - SPEC2006 (best-effort, reduced workloads, gem5 speed + bugs)
Simple Benchmarks

RV64GC vs RV64RC

<table>
<thead>
<tr>
<th>Static Code Size</th>
<th>Dynamic Instructions</th>
<th>Dynamic Instruction Bytes</th>
<th>In-Order Execution Time</th>
<th>Out-of-Order Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable static-code size increase due to register spills/loads</td>
<td>Lesser increase in dynamic instructions. Whetstone a special case (call overhead)</td>
<td>Generally more frequent usage of compressed instructions.</td>
<td>In-order impacted, out-of-order not hurt</td>
<td></td>
</tr>
</tbody>
</table>

Normalized to RV64GC
**SPEC2006 Static Code Size**

**RVR static code**
- 5.7% larger on integer benchmarks
- 6.2% larger on floating-point benchmarks
SPECINT2006 RV64RC Performance

- **400.Perlbench and 471.omnetpp**
  - Highest call frequency benchmarks – call every 35 & 23 instructions
  - Perlbench - 2.8% fewer dynamic function calls due to GCC's inlining heuristics (call overhead)

- **429.mcf**
  - Inlining heuristics greatly impacted
  - 28.7% fewer dynamic function calls w/similar performance
SPECFP2006 RV64RC Performance

- Literature suggests GCC/LLVM allocators fail on 470.lbm w/16 FP registers, but better allocations exist
- Pre-Allocation Instruction Scheduling with Register Pressure Minimization Using a Combinatorial Optimization Approach, G. Shobaki et. al
  - x86-64 + LLVM, reduces register spills from 12 to 2 leading to a 21% improvement
- Benchmark performance bottlenecked by memory system (data cache miss rates)
Multithreading + OoO designs

- Architectural overheads limit the viable design space of multithreaded out-of-order CPUs
  - Wide issue + Large Register Counts = Latency, Area, and Power issues
  - Problematic for even non-multithreaded designs (see BOOM v2)
  - Register file size = GPRs * NumThreads + InstructionWindowSize
- Limited design examples in the market
  - High-end x86 CPUs (AMD/Intel)
  - RISC out-of-orders
    - IBM Power4 => Power 5
      - 50% register file size increase to support 2\textsuperscript{nd} thread
      - 80-120 GPRs, 10% core area (24\% total SMT overhead)
    - IBM Power 8
      - Extended SMT to 4-threads per cluster
      - Required register caching to meet porting + cycle-time requirements
        - 124-entry register (64-architected values, 60-speculative) = 16 architected registers/thread
  - Xeon Phi (Knights Landing)
Multithreaded Design spaces

• Two new potential designs spaces
  • Dual-threaded, small window out-of-order CPUs (~20 instruction windows)
    • Impractical with current RISC-V register file size requirements
    • Integer-based storage & networking workloads with long-latency miss events
  • Large, high-IPC, many threaded CPU designs without resorting to register file caching that reduces/eliminates benefits of more GPRs
    • More benefit to reducing Integer RF compared to FP RF size
      • Integer RF = more heavily ported + forwarding paths
      • Floating Point RF = Fewer ports/ALUs driven
Conclusion

• Highlighted multiple potential market forces behind reduced register file designs

• Demonstrated impacts of reducing RISC-V architected register file sizes
  • Negligible impact on integer-based workloads
  • Minor performance degradation on FP-based workloads
  • Interesting impacts on calling conventions + gcc inlining heuristics

• riscv-toolchain modifications
Backup – Predicted Question

**Question:** ARM moved to 31 GPRs with their 64-bit ISA, they must have had a reason?

**Answer:** ARM already required 31 GPRs in the 32-bit v7 ISA, they just were not simultaneously exposed (banked on execution mode).