Qualcomm

Qualcomm Response to SiFive Presentation '*Retaining C for RVA Profiles*'

Clarification

- Our proposal would mandate **natural instruction alignment** in RVA.
  - Cannot be expressed with IALIGN
    - IALIGN=32, implies all instructions are aligned to 32-bits.
  - 64-bit instructions would be aligned to 64 bits. This may require padding with no-ops.

- In entirety:
  - Prohibit C in future RVA profiles
  - Skip 48-bit encodings in future RVA profiles
  - Mandate naturally aligned instructions in future RVA profiles
  - Add 32-bit instructions to improve code size (placeholder name is Zics, was Znew)
  - Eventually reuse the C space

Statements from SiFive Presentation

Qualcomm agrees

1. **C (specifically variable-length + misaligned) has “technical issues”**
   a. C impacts critical timing loops in a CPU design
   b. It is more difficult to find instruction starts with C, making common optimizations like instruction re-encoding more challenging.
   c. C causes instructions to straddle cache lines
2. **C provides a 20-30% static code size reduction relative to RISC-V without C.**
   a. This is important for some markets.
3. **There are “about 10,000 R-type” instructions left in the 32-bit encoding space**

Additional Analysis by Qualcomm (additions in blue)

1. **C (specifically variable-length + misaligned) has "technical issues"**
   a. C impacts critical timing loops in a CPU design
   b. It is more difficult to find instruction starts with C, making common optimizations like instruction re-encoding more challenging.
   c. C causes instructions to straddle cache lines
2. **C provides a 20-30% static code size reduction relative to RISC-V without C.**
   a. This is important for some markets.
3. **There are "about 10,000 R-type" instructions left in the 32-bit encoding space**
   a. RISC-V has about 7.5% of the 32-bit encoding space left[1]
   b. Equivalently, about 80 I/B/S-type instructions[2]
   c. Only one major 32-bit opcode left (assuming one is reserved for P)
   d. 90% of the free encodings are brownfield, which have limitations
4. **Adding C improves best-case performance by 2-3%**
   a. For the workloads measured by Qualcomm
   b. No contradictory data has been presented thus far

[1] R-type instruction = 2^15 codepoints, (10,000 * 2^15) / 2^32 ~= 7.5%)
[2] 1 I/B/S-type instruction = 128 R-type instructions. 10,000 / 128 ~= 80

Questions Raised

1. **Are the costs of C a burden on RVA markets?**
   - Yes, the costs (on encoding, µarch, and software) are a burden to carry for limited to no performance benefit.
   - While we have focused on the encoding/µarch burden, C's negative effects also extend to software. E.g., with C, JITs must deal with unaligned instructions that cannot be atomically updated
2. **Would removing C without a code size replacement make RISC-V uncompetitive?**
   - RISC-V static code size without C is on par with best-in-class for RVA markets so is not at a competitive disadvantage.
   - Other factors such as ecosystem have a greater impact on adoption of RISC-V.
3. **Should RVA serve a market that wants to (1) run binary distributions and (2) cares about "single-digit % memory size savings."**
   - Such a market represents a niche corner of the binary app markets and, as such, should not burden the mainstream RVA market.

Specific Claims - Response

1. Claim: C reduces power
   - Instruction cache power is ~1% of CPU power – saving a **fraction of 1%** is not a good tradeoff.
   - C **adds power elsewhere**. Taken as a whole, we don't anticipate any power benefit to C.
2. Claim: Instruction straddling is inevitable
   - The claim conflates two issues: fetch group straddling and instruction straddling
   - A fetch group straddling a cache line is inevitable and has minimal impact
   - An instruction straddling a cache line or page boundary **only occurs with misaligned instructions**
3. Claim: C's effect on critical timing loops is modest
   - These loops are *critical*, and **even moderate impacts can affect cycle time** (possibly more than 2-3% best case win from C)
4. Claim: Decoupled instruction fetch blunts the impact of variable instruction starts
   - Decoupled instruction fetch does not address other instruction start problems, e.g., icache re-encoding
   - Decoupled instruction fetch does not absolve hardware from finding branches in long sequences; it just moves the problem in front of a fetch buffer which still needs fed at a high rate

Instruction cracking vs. instruction fusion

Cracking is not fundamentally more difficult than fusion

- Not mentioned: fusion has its own unique challenges
  - Precise exceptions require special handling
  - Finding fusion sequences is not always easy (e.g., when sequence straddles a cache line)
- Qualcomm view: Neither instruction cracking or fusion are problems; neither are as complex as C

- Disputed: “Cracking … requires complex decode -> dispatch buffer management…”
  - Yes, at some point in your design you will have to account for > 1 micro-op coming from a single instruction slot.
  - An implementation can decide where it is convenient to do so.
  - We observe that if the expansion is handled in a stage can move single instructions (opposed to a whole fetch group), the management may just be another hold signal.
- Disputed: “…and increases per-instruction tracking costs”
  - Once cracked, bookkeeping typically just requires a sequence number
- Disputed: “cracking … adds mux complexity … that can add to branch resolution latency”
  - We fail to see how cracking affects branch resolution latency. Cracked instructions are still tied to an architectural instruction, and recover/flush should not be affected.

Thank you

Qualcomm

Follow us on:

For more information, visit us at:

qualcomm.com & qualcomm.com/blog