SiFive

Retaining C (IALIGN=16) for RVA Profiles

Krste Asanovic

October 12, 2023

RISC-V Terminology

ILEN: maximum instruction size

- ◇ Currently ILEN=32 across ratified specs
- ◇ Length encodings >32b not frozen (encoding will change from draft in original spec)

IALIGN: RISC-V supports two values:

- IALIGN=32, instructions start at multiple-of-4 byte addresses
- IALIGN=16 instructions start at multiple-of-2 byte addresses

Why RISC-V adopted variable-length (VL) instructions

- Contemporary instruction sets overflow 32 bits, and fixed 64b instructions make code-size uncompetitive
- VL instructions reduce static code size (reduce cost)
- VL instructions reduce dynamic instruction size (improve performance/energy)
- Longer >32b instructions provide substantial code size and/or performance benefit for some operations
- Currently only 16b and 32b sizes ratified, but plans for longer >32b

- With current profiles, compilers, benchmarks, general agreement is static/dynamic size savings of current C are somewhere in 20-30% range

Technical issues with VL instructions

- Straddling
- I-cache refill time re-encoding
- Finding instruction starts

Straddling

- Instruction may straddle cache line or protection page boundary
- Instructions straddling cache and page boundaries are not a significant implementation challenge for competitive superscalar as have to fetch across cache-line and page boundaries even with fixed-width instructions
- Even with IALIGN=32, straddling unavoidable with ILEN>32, which will be inevitable in RVA profiles
- => Not an argument to remove 16b instructions

I-cache Refill-Time Reencoding

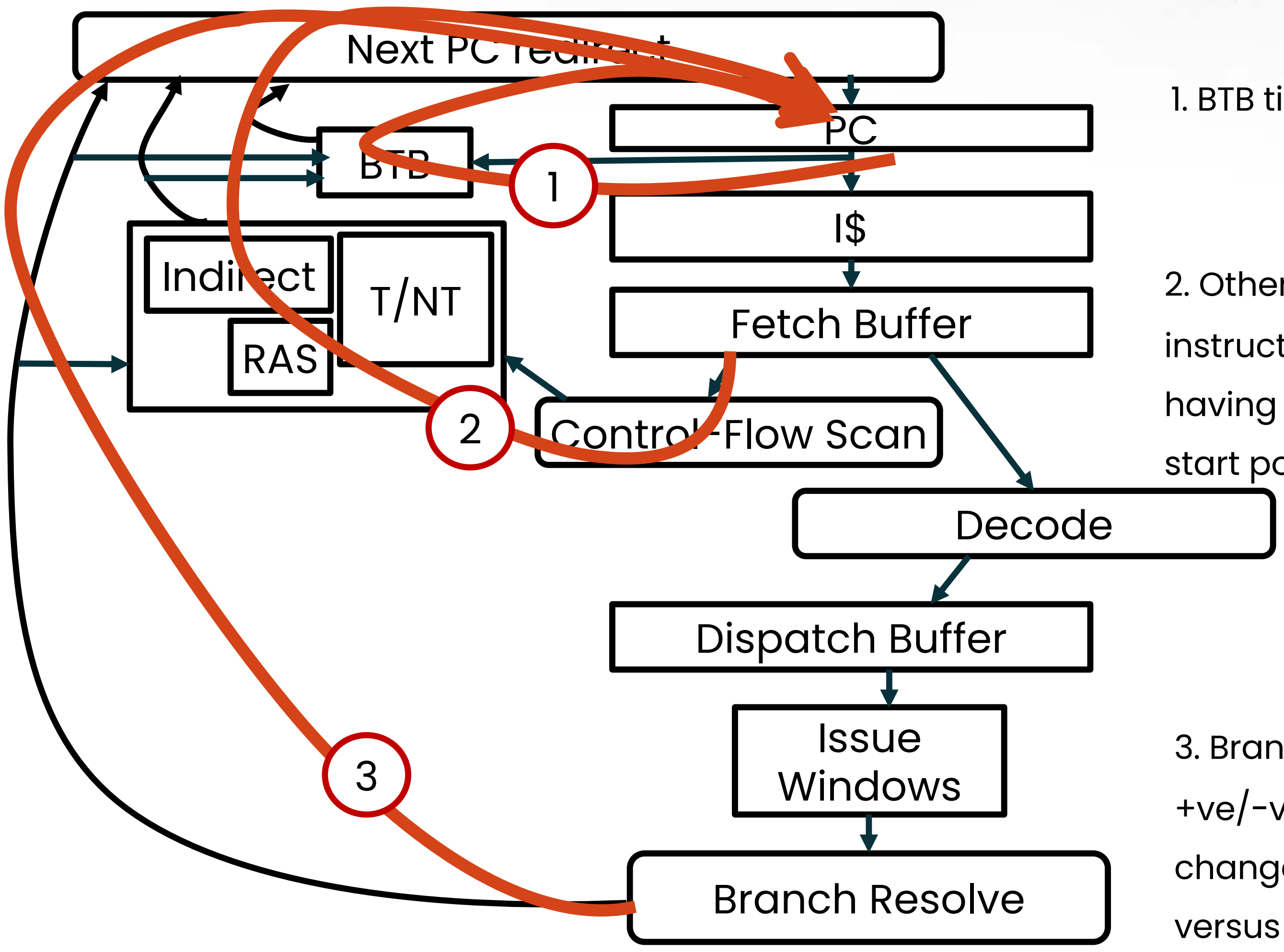
- One common trick in older archs is reencoding instructions at refill time and executing recoded instructions from I-cache
  - e.g., convert low bits of branch PC-relative offsets to absolute
- *Oblivious* re-encoding at I-cache refill time only practical for fixed-width ISA where instruction start points are fixed
- ILEN>32 makes oblivious reencoding impractical even with IALIGN=32

- More advanced re-encoding schemes are possible with any ISA design

IALIGN=16 doubles potential starts

- IALIGN=16 has twice as many potential instruction start locations as IALIGN=32

Moderate Processor Front-End Critical Loops Impact

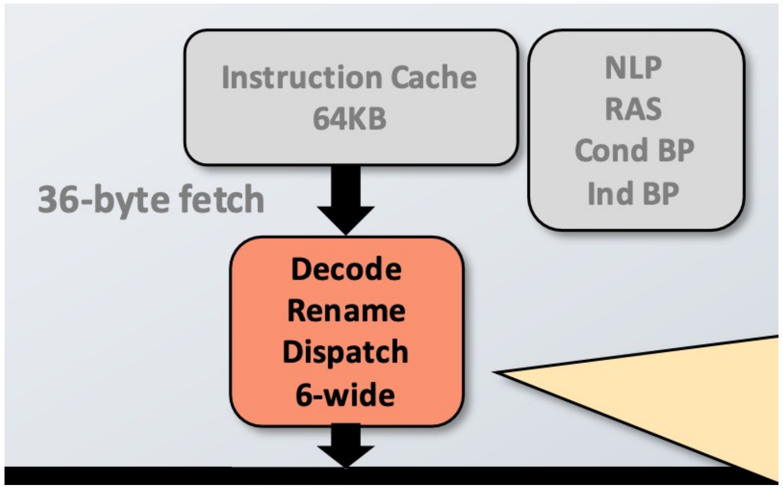


1. BTB tight loop, not affected by IALIGN

2. Other predictors dependent on instruction encoding affected by having to scan fetch buffer at more start points for control-flow redirect

3. Branch resolution latency maybe +ve/-ve effect depending on changes in mux cost (more starts versus fewer bits)

Example: SiFive P870 Front End



[from HotChips 2023 presentation on P870]

- 36-byte fetch is 9-18 RISC-V instructions
- Fetch width far above dispatch width (6-wide) to quickly refill pipe after PC redirects
- Would need ~48 bytes without C

- P870 native RISC-V front-end can scan up to 36 bytes with small impact on predictor loop latency
  - Existence proof that wide scanning is feasible in high-end native RISC-V design

Scaling to bigger superscalars

	Average over all samples	Average in single trace sample	
(no Zicond)	insts/taken branches	min	max
GB5	13.0	8.6	51.9
SPECint06	10.5	7.2	45.8
SPECfp06	28.7	11.5	855.2

- Control-flow changes on integer workloads already at point where wider front end will have to predict multiple PCs to be effective
- Taken branch frequency puts upper limit on problem of control-flow scanning in middle of front-end
- In any case, move to larger BTB structures that are decoupled from instruction fetch (and hence encoding) makes finding instruction start boundaries less critical

Qualcomm proposal

- Fix IALIGN=32 in future RVA profiles
- Add complex 32b instructions to mitigate 30-40% code size increase

Opcode usage

- Current RISC-V 30b opcode space still has vast space available
  - Approx 10,000 R-type instructions slots available
  - e.g. ”Znew” proposal fits in small fraction of brownfield space

- Sacrificing C to open up other 3 encodings in 2 bits doesn’t help with encoding new instructions that need substantially greater than 30-32b to encode
  - e.g., 32b immediates, longer calls, more source/destination register specifiers

- IALIGN=16 supports 16b HINTs (e.g. existing NTL hint)
  - HINTs tend to occur in hot loops, impacting dynamic fetch costs
- IALIGN=16 supports C.MOPS which reduce code size cost of new security features
  - Checks add code at exit/entry to every function+ indirect branch targets
- IALIGN=16 enables 48b instructions that substantially improve code size
  - E.g., Just replacing 64b AUIPC-based sequences in Linux kernel with 48b instructions would save 3.6% code size over and above linker relaxation)

SiFive

Complex Instruction Problems

- Moving from simple 32b instructions to complex 32b instructions adds considerable mid-core complexity
- Cracking high-frequency instructions in a superscalar requires complex decode->dispatch buffer management and increases per-instruction tracking costs
  - While low-frequency instructions can be cracked more simply by serializing instruction execution (e.g., CAS), highly superscalar cracking is significantly more expensive.
- Variable amount of cracking in highly superscalar decode adds mux complexity in downstream uop queue datapath that can add to branch resolution latency, removing/reversing purported advantage over C
- Substantially complicates ALL core designs, as even simple cores now require cracking circuitry, while not matching code size savings of C.

- In contrast, complex cores may selectively “fuse” C only when maps to a single internal uop, whereas cracking is effectively mandatory for all microarchitectures.

Binary ecosystems run on small cores

- Binary software ecosystems are not only for large superscalar processors
  - Many small processors need to run this software in environments where static code size and processor area are critical cost factors
    - e.g., in-order dual-issue (e.g., ARM A53/A55) very popular for Android, perhaps the most common cores numerically
  - Requiring increase from 32KB to 48KB I-cache is considered very negatively (RV64GC has substantial advantage over AArch64 for sockets upgrading from AArch32).
  - Single-digit % memory size savings are significant in these markets running binary software distributions. E.g., some sockets are memory-constrained and requesting ILP32 ABI for RV64GC to cut data segment size (possible 10-15% saving)
  - This will continue to be a large market segment for RVA, not going away
- Not realistic to expect software ecosystems to support both IALIGN=16 and IALIGN=32 binaries.

Summary

- Dropping C will cause large disruption in software ecosystem, both one-off and on-going
  - Without new complex instructions, code-size penalty will hurt RISC-V adoption, and any complex instruction extension will take time to ratify/implement/support even if community agrees to proceed, causing significant delay (>12 months?) in stabilizing the ecosystem at critical time in RISC-V rollout
  - Needing separately tuned/QA builds of common ecosystem components across IALIGN=16 and IALIGN=32 profiles adds to RISC-V software burden and builds in permanent fragmentation
- No clearly discernible technical benefit for native RISC-V cores, and some clear technical disadvantages for both low-end and high-end implementations