Better Living Through Bit Manipulation
Higher Performance at Lower Power

RISC-V BitManip Task Group

Ken Dockser, Qualcomm, Task Group Chair
Clifford Wolf, Symbiotic GmbH, Co-char and spec author
RISC-V Bit-Manip Task Group

- **Mission**
  - define extensions to the Unprivileged ISA
  - for additional “bit-based” (non-integer) instructions
  - operating on the integer x0..x31 registers
  - for higher performance and lower power

- **Spec 0.90**
  - Instructions for bit counts, rotate shift, set/clear bits, bit permutations, extrac/deposit with bitmask, ...
  - Organized into individual Zb_ extensions
  - “Zbb” (base) for small cores
  - “B” for large cores (definition of “B” is TBD!)
  - “Zbt” for instructions with three source operands (not in “B”)
Design Criteria (1/2)

• Architecture Consistency
  - use existing instruction formats
  - build on existing ISA encodings for instruction decoder simplicity

• Threshold Metric: Any new instruction should
  - replace at least three instructions, or avoid a branch, or
  - be used frequently and is cheap to implement

• Data-Driven
  - we need to show the merit of each instruction in micro-benchmarks and
  - show the merit of the extensions as a whole in real world applications
Design Criteria (2/2)

• Hardware Simplicity
  - instructions must be reasonably simple to implement with acceptable area and timing
  - reference implementations of re-usable “B-ALUs” under permissive license will be provided

• Compiler Support
  - instructions should, at least in some instances, be detectable by a compiler in portable code, or
  - can be used via compiler intrinsics to significantly speed up common tasks, often provided via libraries. (such as CRC, or strlen, or ...)
Higher Performance at Lower Power

- Do more with fewer instructions and cycles
  - Thus increasing performance

- Without adding too many additional transistors
  - Turning higher instruction throughput to lower power
  - “Zbb” is aimed at small cores, “B” for large core

- And with fewer memory accesses for lookup-tables
  - Thus further increasing performance
  - And further decreasing power
Current Status

- Z-extensions (excluding Zbb) should be fairly stable now
- C reference models and opcode encodings
- Support for “B” in Imperas RISC-V simulator
- Patches for binutils and riscv-isa-sim (spike)
- “portable” intrinsics in <rvintrin.h>
  - Using asm templates for native code
  - And behavioral C models for emulation testing
- Patch for gcc intrinsics is on its way

Next Steps

- Review post-0.90 / post-workshop feedback
- Complete support in software toolchains
- Build and release reference “B-ALUs”
- Build benchmarks
  - Both micro-benchmarks
  - And real-world applications
- Finalize instruction selection for “Zbb” and “B” extensions
- Build compliance tests and formal models

The images on this slide are from section “2.2.3 Generalized Shuffle (shfl,unshfl,shfli,unshfli)” of the 0.90 spec. Read the spec!
Full List of New Instructions
What we need now

• Feedback on the 0.90 proposal that we just released
  - The window for proposing new instructions, or proposing changes to the instructions we have, is closing quickly. Submit your feedback NOW.
    - Join the task group https://lists.riscv.org/g/tech-bitmanip/
    - Or create a github issue https://github.com/riscv/riscv-bitmanip
    - Or talk to us IRL here at the RISC-V Workshop!

• Benchmark problems and (even better) complete benchmarks in versions with and without instructions from the “B” extension.
  - We have implemented compiler intrinsic functions that help with writing “B” extension code. See <rvintrin.h>.
Example usage of <rvintrin.h>

```c
#include <rvintrin.h>

int find_nth_set_bit(unsigned int value, int cnt)
{
    return _rv32_ctz(_rv32_bdep(1 << cnt, value));
}
```
Example usage of `<rvintrin.h>`

```c
#include <rvintrin.h>

// CRC for Aeronautical Information Exchange Model (AIXM)
uint32_t crc32q(const uint32_t *data, int length) {
    uint32_t crc = 0;
    for (int i = 0; i < length; i++) {
        crc ^= _rv32_grev(data[i], -8);
        crc = _rv32_clmulr(crc, 0xFF7FBFB1);
        crc = _rv32_clmul(crc, 0x814141AB);
    }
    return crc;
}
```
What we need now:

- Feedback on the 0.90 proposal that we just released
  - The window for proposing new instructions, or proposing changes to the instructions we have, is closing quickly. Submit your feedback NOW.
    - Join the task group [https://lists.riscv.org/g/tech-bitmanip/](https://lists.riscv.org/g/tech-bitmanip/)
    - Or create a github issue [https://github.com/riscv/riscv-bitmanip](https://github.com/riscv/riscv-bitmanip)
    - Or talk to us IRL here at the RISC-V Workshop!

- Benchmark problems and (even better) complete benchmarks in versions with and without instructions from the “B” extension.
  - We have implemented compiler intrinsic functions that help with writing “B” extension code. See [rvintrin.h](https://github.com/riscv/riscv-bitmanip).
Imperas support of BitManip (B) extension to riscvOVPsim reference simulator available for free on github

Lee Moore, Imperas Software (moore@imperas.com)  
RISC-V Zurich (11th June 2019)
Bit Manipulation Extensions

- Simulator provides early access for evaluation of performance and compiler development
- Currently modeled as a Custom Extension in riscvOVPsim until definition has stabilized (and ratified)

Supported Instructions:
Following list is supported for 32 & 64 bit implementations

- ANDN, GREV, SLO, SRO, ROL, ROR, FSL, FSR,
- GREVI, SLOI, SROI, RORI, CLZ, CTZ, PCNT,
- BMATFLIP, CRC32_B, CRC32_H, CRC32_W,
- CRC32_D, CRC32C_B, CRC32C_H, CRC32C_W,
- CRC32C_D, CMIX, CMOV, CLMUL, CLMULH, MIN,
- MAX, MINU, MAXU, BDEP, BEXT, SHFL, UNSHFL,
- BMATXOR, BMATOR, SHFLI, UNSHFLI
Testing

• For all instructions a bespoke test has been written. In the absence of a toolchain hand coded binary implementations are produced using MACRO definitions to produce correct bit patterns

• Example CLZ

```assembly
# DECODE_ENTRY(0, CLZ, "|0110000|00000|.....|001|.....|0010011|
.macro CLZ rd rs1
   .int 0x60001013 | ((rd&0x1F)<<7) | ((rs1&0x1F)<<15)
.endm

// Call to perform a count leading zeros of register x8, placing result in x9
CLZ 9 8
```
Coverage of Model B extensions for tests

- Line Coverage 96.1%
- Functions Coverage 98.9%

**LCOV - code coverage report**

<table>
<thead>
<tr>
<th>Filename</th>
<th>Line Coverage</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>extB.c</td>
<td>96.1 % 828 / 862</td>
<td>98.9 % 86 / 87</td>
</tr>
</tbody>
</table>

Current view: top level - riscv.ovpworld.org/intercept/extB/1.0/model

Test: Imperas Test Coverage Results
Date: 2019-06-06

Hit Total Coverage
Lines: 828 862 96.1 %
Functions: 86 87 98.9 %
extra slides
Short Instruction Descriptions (1/4)

• Simple instructions
  - Count leading/trailing zeros (clz, ctz)
  - Population count (pcnt)
  - Min/max (min[u], max[u])
  - Logic with rs2 negate (andn, orn, xnor)
  - Shift ones (slo[i], sro[i])
  - Rotate shift (rol, ror[i])
  - Single-bit set/clear/invert/extrace (sbset, sbclr, sbinv, sbext)
  - Packing two XLEN/2 words into one XLEN word (pack)
Short Instruction Descriptions (1/4)

- Bit permutation instructions
  - Bit permutations are invertable functions over bit indices
    - Rotate shift = add/subtract (mod XLEN)
    - Generalized reverse = XOR with mask
    - Generalize shuffle = bitperm within bit indices
  - Pseudo-instructions for most common cases
    - Bitreversal (brev, aka bitreflect)
    - Bytereversal (bswap, aka endianness-swap)
    - “zip” and “unzip” (perfect bitwise outer (un-)shuffle)
    - Swap bit pairs (brev.p), swap middle bits of each nibble (zip.n)
Short Instruction Descriptions (1/4)

- Ternary Instructions (Zbt)
  - Conditional move (cmov)
  - Conditional “mix” (cmix, i.e. rd := (rs1 & rs2) | (rs3 & ~rs2))
  - Funnel shift (fsl,fsr,fsri)

- Bitwise extract/deposit (Zbe, bext/bdep)
  - Similar to x86 pext/pdecb instructions
  - bext = extract marked bits (rs2) from rs1, compressed to right
  - bdep = deposit compressed bits from rs1 in marked positions (rs2)
Short Instruction Descriptions (4/4)

- Carry-less multiply (clmul, clmulh, clmulr)
  - Classical use-cases: CRC with arbitrary polynomial, GCM
  - But also many bit hacks (see references in draft spec)
- Dedicated CRC instructions
  - For (small?) processors without clmul support
  - Only support for CRC32 and CRC32C polynomials
- Bit-matrix instructions (bmat[x]or, bmatflip, RV64/128-only)
  - Treat 64-bit numbers as 8x8 matrices
  - Matrix-matrix-product with AND as multiply and [X]OR as addition
Bit Permutation Network(s)

rotate
grev

(un)shuffle