RISC-V Compressed Extension

Andrew Waterman, Yunsup Lee, David Patterson, and Krste Asanović

{waterman|yunsup|patterson|krste}@eecs.berkeley.edu

http://www.riscv.org

2nd RISC-V Workshop, Berkeley, CA
June 29, 2015
“C”: Compressed Instruction Extension

- Compressed code important for:
  - low-end embedded to save static code space
  - high-end commercial workloads to reduce cache footprint

- Standard extension (released 5/28/15) adds 16-bit compressed instructions with 5- or 6-bit opcode and
  - 2-addresses with all 32 registers
  - 1-address with all 32 registers & 5-bit immediate
  - 2-addresses with popular 8 registers & 5-bit immediate
  - 1-address with popular 8 registers & 8-bit immediate
  - 11-bit immediate

- Each C instruction expands to single base I instruction
  - Compiler can be oblivious to C extension
  - Assembly lang programmers can also ignore C extension

- ~50% instructions ⇒ ~25% reduction in code size
### Variable-Length Encoding

<table>
<thead>
<tr>
<th>Address Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xxxxxxxxxxxxxxaa</code></td>
<td>16-bit (aa ≠ 11)</td>
</tr>
<tr>
<td><code>xxxxxxxxxxxxxxx xxxxxxxxxxxxxbb</code></td>
<td>32-bit (bbb ≠ 111)</td>
</tr>
<tr>
<td><code>···xxxx xxxxxxxxxxxx xxxxxxxx011111</code></td>
<td>48-bit</td>
</tr>
<tr>
<td><code>···xxxx xxxxxxxxxxxx xxxxxxxx011111</code></td>
<td>64-bit</td>
</tr>
<tr>
<td><code>···xxxx xxxxxxxxxxxx xnnnxxxx111111</code></td>
<td>(80+16*nnn)-bit, nnn ≠ 111</td>
</tr>
<tr>
<td><code>···xxxx xxxxxxxxxxxx x111xxxx111111</code></td>
<td>Reserved for ≥192-bits</td>
</tr>
</tbody>
</table>

Byte Address: base+4  base+2  base

- Extensions can use any multiple of 16 bits as instruction length
- Branches/Jumps target 16-bit boundaries even in fixed 32-bit base
  - Consumes 1 extra bit of jump/branch address
Prior Work

- Retroactively added 16-bit instructions to RISCs ISAs of 1980s to reduce code size for embedded apps
- ARM Thumb: All instructions 16-bits
  - New ISAs
  - Mode change to use ARMv7 instructions
- ARM Thumb2 and MIPS16: Mix of 16-bit and 32-bit instructions
  - New ISAs (different from ARMVv7 and MIPS32)
  - Mode change to use ARMv7 instructions
Methodology

- Implement proposed instruction in assembler
- Measure impact of proposed instruction on static code size using SPEC2006 and other codes bases
- Discarded if little benefit, e.g.,
  - 3 register arithmetic/logic operations
  - ARMv7-like “swizzling” / table lookup of constants
- Discarded if opportunity costs too high, e.g.,
  - Load/store byte/halfword
    - Help a little, but takes up too much opcode space vs. benefits
  - Load/store floating-point single/double
    - Lots of opcode space again vs. benefits
    - Only helps FP programs vs. all programs
    - RVF optional so only helps some RV computers vs. all RV computers
RVC Overview

- 1\textsuperscript{st} 10 RVC instructions \(~14\%\) code reduction
- 2\textsuperscript{nd} 10 RVC instructions \(~6\%\) more code reduction
- 3\textsuperscript{rd} 10 RVC instructions \(~5\%\) more code reduction
- Draft describes 24 more “extended” RVC instructions that get \(~1\%\) more code reduction
  - Maybe more compression for other compilers than gcc, other languages than C, assembly language programming?
  - Please comment on “standard” vs. “extended” RVC
- Recent results on compiler optimization to reduce size of register save/restore code on procedure entry/exit
**RVC Reg-Reg Operations**

<table>
<thead>
<tr>
<th>funct4</th>
<th>rd/rs1</th>
<th>rs2</th>
<th>op</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

- Register destination and 1\textsuperscript{st} source identical
  \[ \text{Reg[rd]} = \text{Reg[rd]} \text{ op Reg[rs2]} \]  (CR format)
- **C.MV # move**
  - Expands to \texttt{add rd, x0, rs2}
- **C.ADD # add**
  - Expands to \texttt{add rd, rd, rs2}
- **C.ADDW # add word**
  - Expands to \texttt{addw rd, rd, rs2}
- **C.SUB # subtract**
  - Expands to \texttt{sub rd, rd, rs2}
- 7.4\% of code
Load/Store SP + imm field

- **C.LWSP, C.LDSP, C.LQSP**
- Load Word/Double word/Quad word from Stack Pointer + imm*{4|8|16} to Reg[rd] (CI format)
  - Expands to \( l\{w|d|q\} \) rd, offset(x2)
- 3.5% of code (10.9% total)

- **C.SWSP, C.SDSP, C.SQSP**
- Store Word/Double word/Quad word to Stack Pointer + imm*{4|8|16} from Reg[rs2] (CSS)
  - Expands to \( s\{w|d|q\} \) rs2, offset(x2)
- 2.8% of code (13.7% total)
Load/Store Reg + imm field

<table>
<thead>
<tr>
<th>funct3</th>
<th>imm</th>
<th>rs1’</th>
<th>imm</th>
<th>rd’</th>
<th>op</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- **C.LW, C.LD, C.LQ**
- Load Word/Double word/Quad word from Reg[rs1’] + imm*{4|8|16} to Reg[rd’] (CL format)
  - Expands to \( l\{w|d|q\} \text{ rd’}, \text{ offset (rs1’)} \)
- 2.2% of code (15.9% total)

<table>
<thead>
<tr>
<th>funct3</th>
<th>imm</th>
<th>rs1’</th>
<th>imm</th>
<th>rs2’</th>
<th>op</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- **C.SW, C.SD, C.SQ**
- Store Word/Double word/Quad word to Reg[rs1’] + imm*{4|8|16} from Reg[rs2’] (CS format)
  - Expands to \( s\{w|d|q\} \text{ rd’}, \text{ offset (rs1’)} \)
- 0.7% of code (16.6% total)
Load Constant

- **C.LI # load immediate**
  
  Reg[rd] = imm (CI format)
  
  - Expands to `addi rd, x0, nzimm[5:0]`

- **1.6% of code (18.2% total)**

- **C.LUI # load upper immediate**
  
  Reg[rd] = imm*4096 (CI format)
  
  - Expands to `lui rd, nzimm[17:12]`

- **0.4% of code (18.6% total)**

---

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
Add Immediate Operations

- **C.ADDI**

  Add Immediate
  
  \[ \text{Reg[rd]} = \text{Reg[rd]} + \text{imm} \text{ (CI format)} \]
  
  - Expands to \text{addi rd, rd, nzimm[5:0]}

- **C.ADDIW**

  Add Immediate Word
  
  \[ \text{Reg[rd]} = \text{Reg[rd]} + \text{imm} \text{ (CI format)} \]
  
  - Expands to \text{addiw rd, rd, nzimm[5:0]}

- 1.8% of code (20.4% total)
Remaining 10 RVC instructions
(4.9% of code, 25.2% total)

- **C.BEQZ**  # branch on equal to zero
- **C.BNEZ**  # branch on not equal to zero
- **C.J**  # jump
- **C.JR**  # jump register
- **C.JAL**  # jump and link
- **C.JALR**  # jump and link register
- **C.SLLI**  # shift left logical
- **C.ADDI16SP**
  - SP = SP + sign-extended Immediate scaled by 16
- **C.ADDI4SPN**
  - Reg = SP + zero-extended Immediate scaled by 4
- **C.EBREAK**
  - Environment break, for debuggers
Extended RVC another ~1% using gcc
(Please opine whether to add extended)

<table>
<thead>
<tr>
<th>24 Extended RVC Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.ADD3</td>
</tr>
<tr>
<td>C.AND3</td>
</tr>
<tr>
<td>C.OR3</td>
</tr>
<tr>
<td>C.SUB3</td>
</tr>
<tr>
<td>C.ADDIN</td>
</tr>
<tr>
<td>C.ANDIN</td>
</tr>
<tr>
<td>C.ORIN</td>
</tr>
<tr>
<td>C.XORIN</td>
</tr>
<tr>
<td>C.SLT</td>
</tr>
<tr>
<td>C.SLTR</td>
</tr>
<tr>
<td>C.SLTU</td>
</tr>
<tr>
<td>C.SLTUR</td>
</tr>
</tbody>
</table>

- 3 Registers (0.34%)
- 2 Registers & Imm (0.13%)
- Comparisons (0.01%)
- Logical Imm (0.11%)
- Logical (0.02%)
- Branches (0.10%)
- Shifts (0.24%)
MIPS delayed branch slots increase code size
RV64C only 64-bit address ISA with 16-bit instructions
Thumb2 only 32-bit address ISA smaller than RV32C
Load/Store Multiple?

- Thumb2 has load/store multiple (LM/SM)
  - LM/SM violates 1-to-1 mapping of RVC to RVI instructions
- Save/restore registers on procedure entry/exit using LM/SM reduces code size
- When prefer smaller code over speed, instead of in-lined loads and stores, call procedures to save/restore registers on procedure entry/exit
  - gcc allocates registers in order, so just need number of registers to save/restore
  - Can do in just 1 JAL for save + 1 Jump for restore by jumping into middle of save/restore procedures to determine number of registers to save or restore
- RVC code size shrinks another ~5%
SPECint2006 with save/restore optimization (relative to “standard” RVC)

- RISC-V now smallest ISA for 32- and 64-bit addresses
  - Average 34% smaller for RV32C, 42% smaller for RV64C
Questions?