RISC-V Compressed Extension

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

{waterman|yunsup|pattrsn|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

- 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

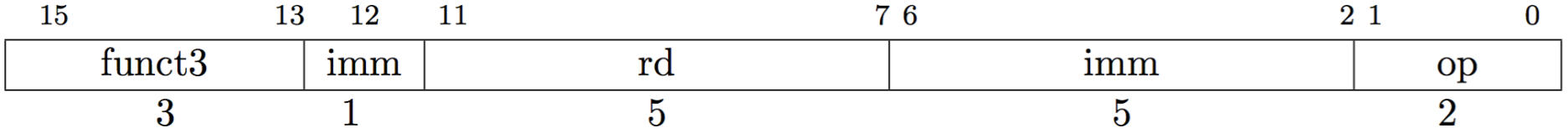
RVC Overview

- 1st 10 RVC instructions ~14% code reduction
- 2nd 10 RVC instructions ~6% more code reduction
- 3rd 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

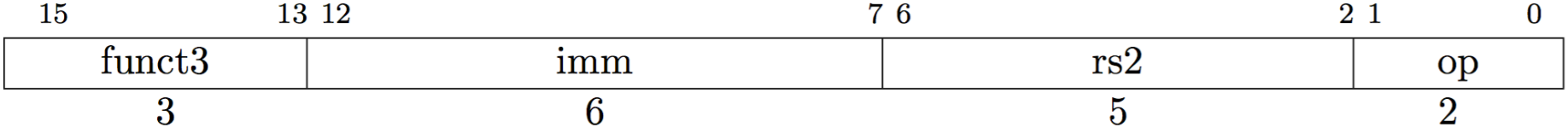
RVC Reg-Reg Operations

- Register destination and 1st source identical
  Reg[rd] = Reg[rd] op Reg[rs2] (CR format)
- `C.MV # move`
  - Expands to `add rd, x0, rs2`
- `C.ADD # add`
  - Expands to `add rd, rd, rs2`
- `C.ADDW # add word`
  - Expands to `addw rd, rd, rs2`
- `C.SUB # subtract`
  - Expands to `sub rd, rd, rs2`
- 7.4% of code

Load/Store SP + imm field

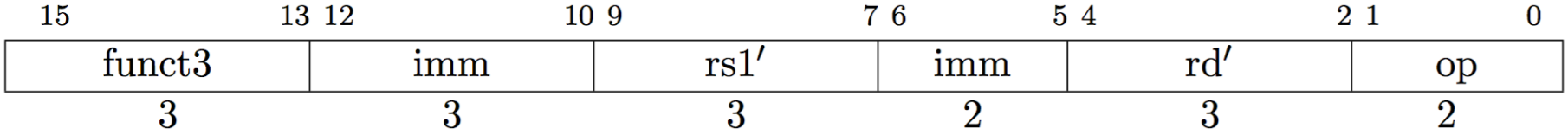


- C.LWSP, C.LDSP, C.LQSP
- Load Word/Double word/Quad word from Stack Pointer + imm*{4|8|16} to Reg[rd] (CI format)
  - Expands to `l{w|d|q} rd, offset(x2)`
- 3.5% of code (10.9% total)

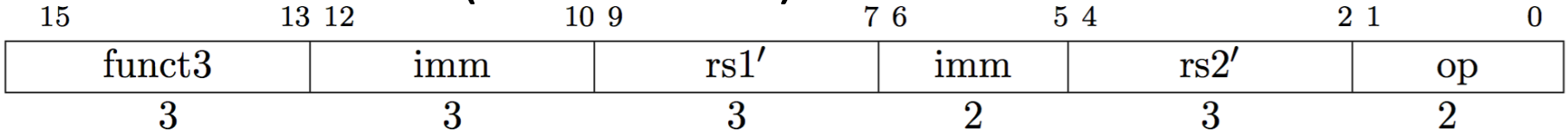


- C.SWSP, C.SDSP, C.SQSP
- Store Word/Double word/Quad word to Stack Pointer + imm*{4|8|16} from Reg[rs2] (CSS)
  - Expands to `s{w|d|q} rs2, offset(x2)`
- 2.8% of code (13.7% total)

Load/Store Reg + imm field

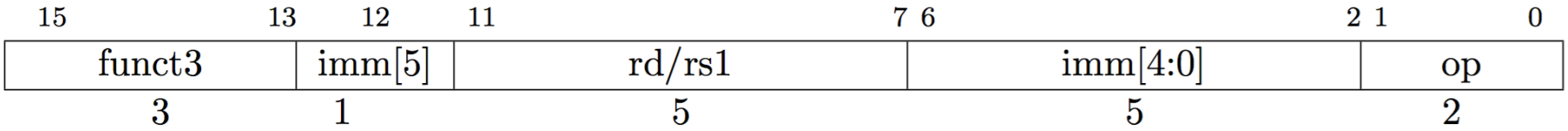


- C.LW, C.LD, C.LQ
- Load Word/Double word/Quad word from Reg[rs1'] + imm*{4|8|16} to Reg[rd'] (CL format)
  - Expands to `l{w|d|q} rd', offset(rs1')`
- 2.2% of code (15.9% total)



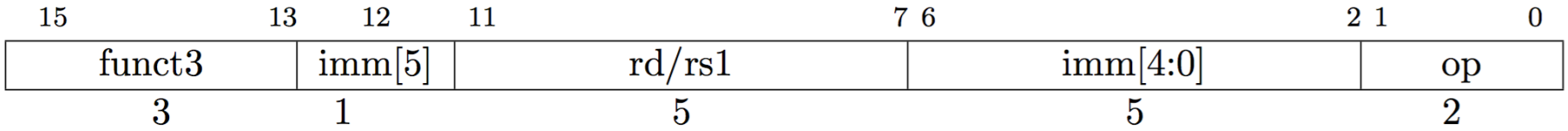
- C.SW, C.SD, C.SQ
- Store Word/Double word/Quad word to Reg[rs1'] + imm*{4|8|16} from Reg[rs2'] (CS format)
  - Expands to `s{w|d|q} rd', offset(rs1')`
- 0.7% of code (16.6% total)

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)

Add Immediate Operations



- `C.ADDI`
- Add Immediate

  Reg[rd] = Reg[rd] + imm (CI format)
  - Expands to `addi rd, rd, nzimm[5:0]`
- `C.ADDIW`
- Add Immediate Word

  Reg[rd] = Reg[rd] + imm (CI format)
  - Expands to `addiw rd, rd, nzimm[5:0]`
- 1.8% of code (20.4% total)

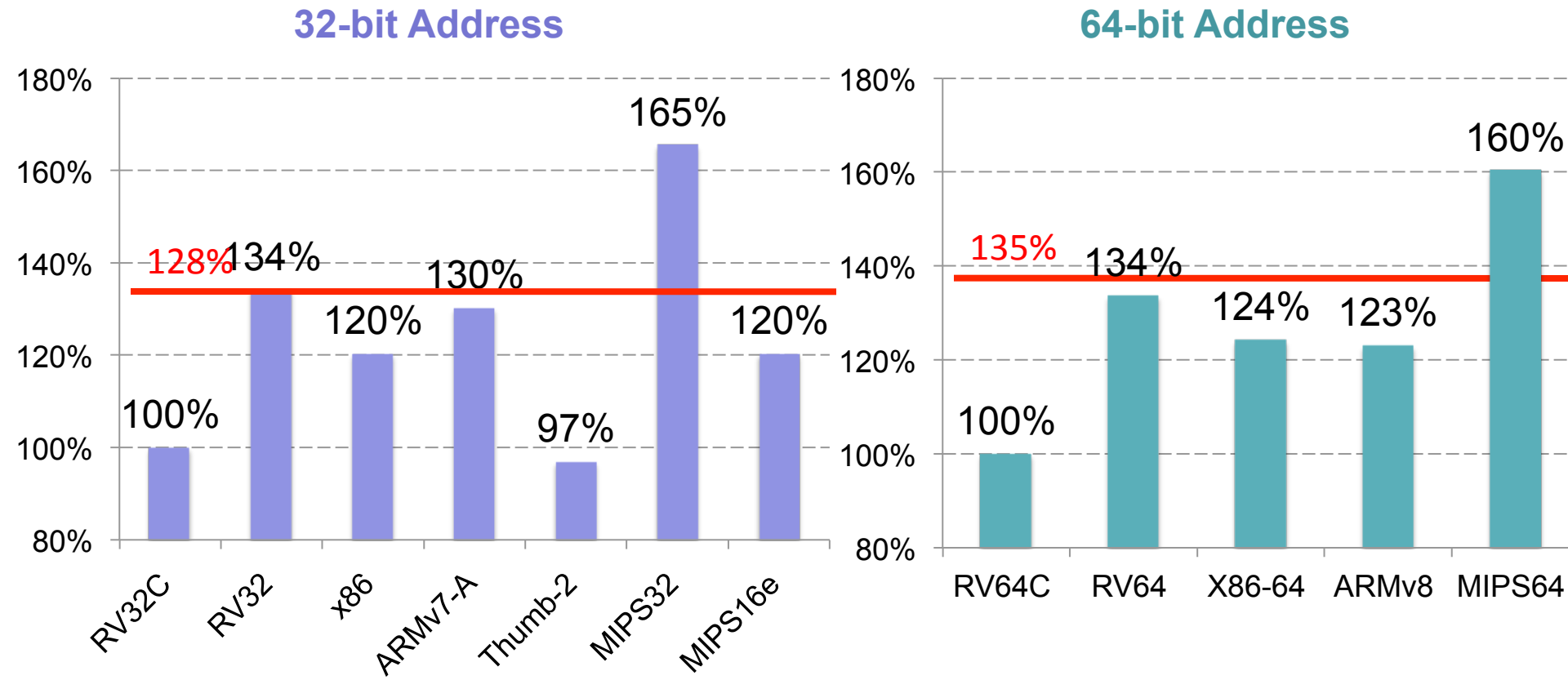
Remaining 10 RVC instructions (4.9% of code, 25.2% total)

- `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)

24 Extended RVC Instructions			
C.ADD3	3 Registers (0.34%)	C.SLL	Shifts (0.24%)
C.AND3		C.SLLIW	
C.OR3		C.SLLR	
C.SUB3		C.SRA	
C.ADDIN	2 Registers & Imm (0.13%)	C.SRAI	
C.ANDIN		C.SRL	
C.ORIN		C.SRLI	
C.XORIN		C.SRLR	
C.SLT	Comparisons (0.01%)	C.BGEZ	Branches (0.10%)
C.SLTR		C.BLTZ	
C.SLTU		C.ANDI	Logical Imm (0.11%)
C.SLTUR		C.XOR	Logical (0.02%)

SPECint2006 Compression Results (relative to "standard" RVC)

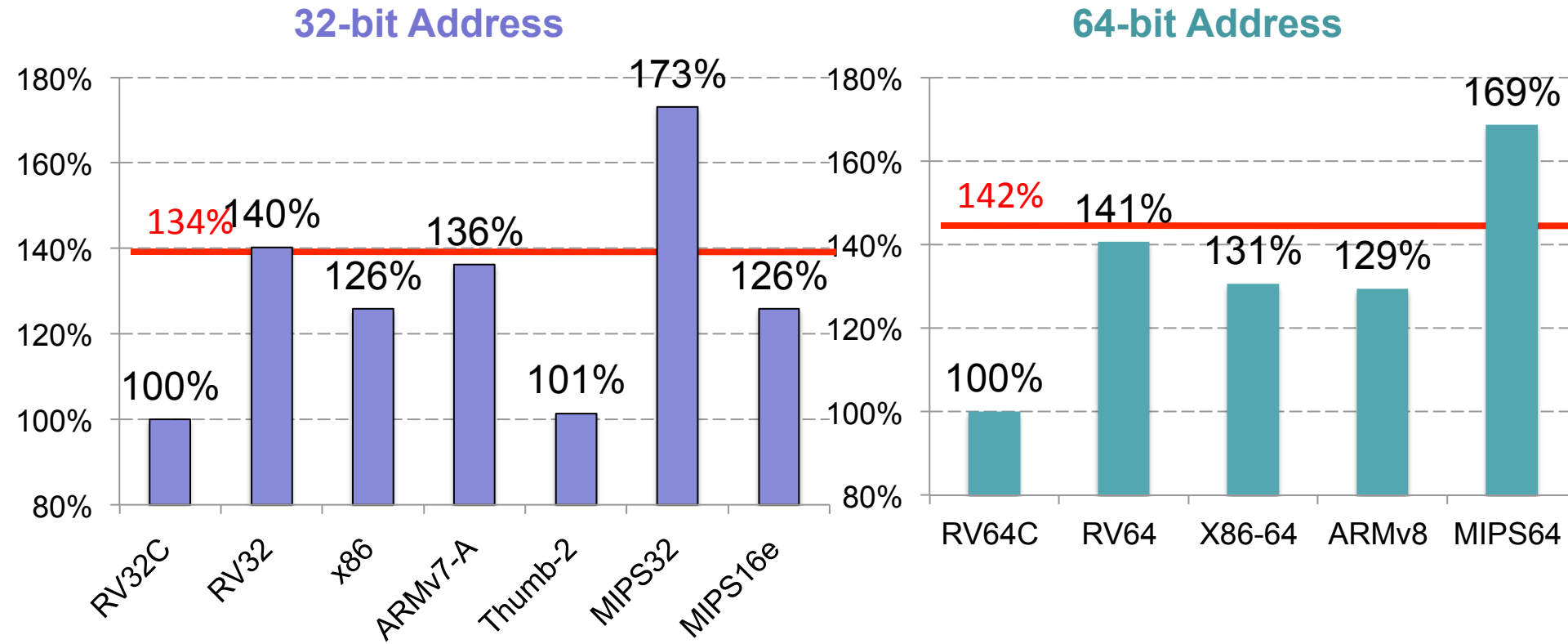


- MIPS delayed branch slots increase code size
- RV64C only 64-bit address ISA with 16-bit instructions
- Thumb2 only 32-bit address ISA smaller than RV32C

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?