Towards a RISC-V Instruction Set Extension for Multi-word Arithmetic

Youngjin Eum, Naifeng Zhang, Larry Tang, Franz Franchetti

Department of Electrical and Computer Engineering, Carnegie Mellon University

Problem

- Multi-word arithmetic has applications in cryptography when working with large numbers
- Multi-word arithmetic is expensive to implement in RISC-V architecture due to lack of carry flag
- Only one carry flag in other architectures is limiting

ISA Extension

Idea: Extension of RISC-V ISA with 4 distinct carry registers

- Instructions to set and use the carry
  - Carry-producing/consuming addition/subtraction
  - Multiply high 64 bits, low 64 bits
  - Load from carry register to integer register
  - Store from integer register to carry register
- Individually addressable carry registers
- Can implement cheaply multi-word addition, subtraction, multiplication, and modulo operators

```
add  a0, a0, a3
add  a1, a1, a2
sltu a2, a1, a2
add  a0, a0, a2
```

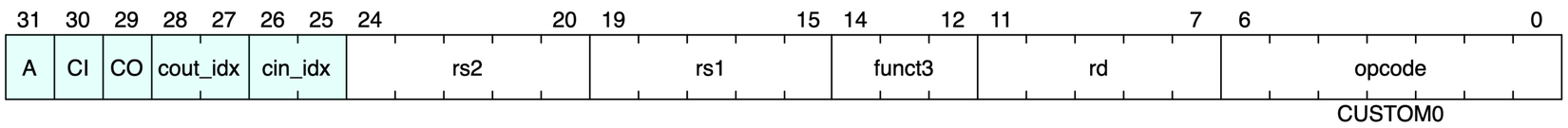
Base RISC-V code for adding 128-bit integers

```
addqc  a1, a1, a2, c0
adc    a0, a0, a3, c0
```

Extended RISC-V code for adding 128-bit integers

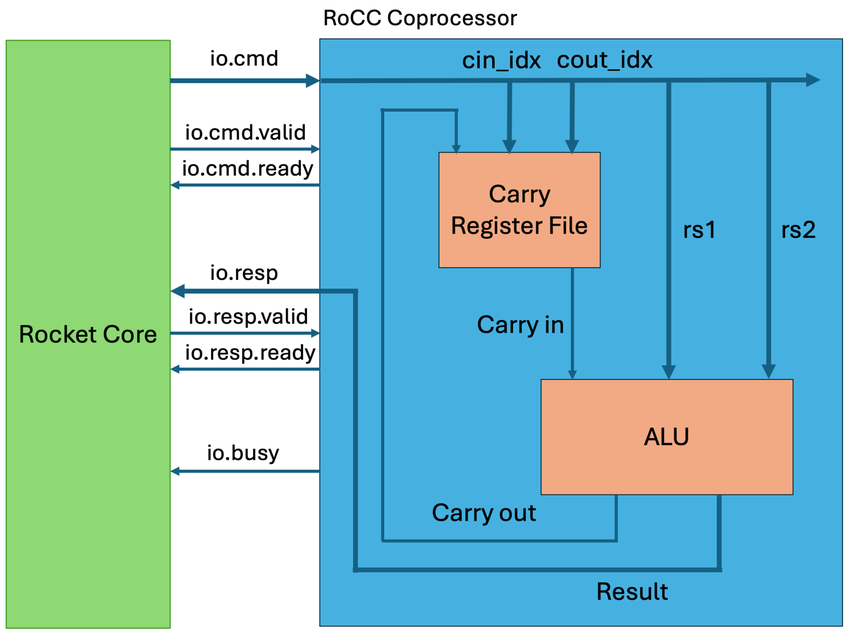
Below is an example encoding of carry-producing/consuming addition/subtraction instruction.

Encoding is identical to RISC-V R-type instruction.
A = add/sub, CI = carry in used, CO = carry out used
cout_idx and cin_idx index into the carry register file



Hardware Implementation

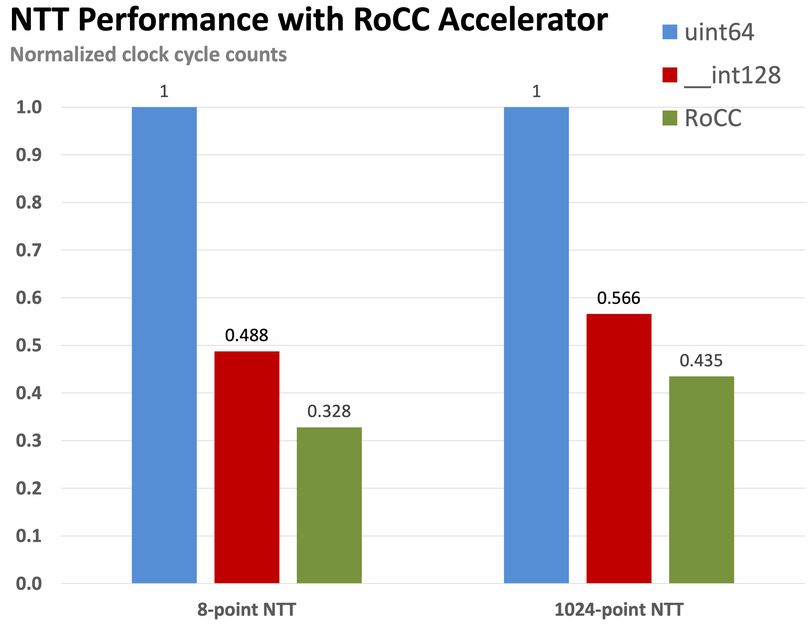
- Custom accelerator implemented using Chipyard v0.12
- Using RoCC interface with in-order Rocket core
- 4 carry registers in carry register file, updated when carry out is used




```
1  class RoCCCarryModule(outer: RoCCCarry) extends
   LazyRoCCModuleImp(outer) {
2  val opcode = io.cmd.bits.inst.opcode
3  val funct7 = io.cmd.bits.inst.funct
4  val cout_index = funct7(3, 2)
5  val cin_index = funct7(1, 0)
6  val sum = Wire(UInt(65.W))
7
8  when (funct7(6) === 0.U) {
9    when (funct7(5) === 1.U) {
10     // carry in
11     sum := io.cmd.bits.rs1 +& io.cmd.bits.rs2 +&
       regs(cin_index)
12   } .otherwise {
13     // no carry in
14     sum := io.cmd.bits.rs1 +& io.cmd.bits.rs2
15   }
16
17   when (funct7(4) === 1.U && io.cmd.fire) {
18     // carry out
19     regs(cout_index) := sum(64)
20   }
21 }
22 // ... omitted
23 io.resp.bits.data := sum(63, 0)
```

- Addressing carry indices with funct7 field
- +& operator in Chisel to preserve carry

Results



- Evaluated using NTT benchmark generated using SPIRAL NTTX package
  - Leading application of multi-word arithmetic
- Baseline results using manual computation of carries (uint64_t) and compiler optimized arithmetic (__int128)
- Implementation of modular addition, subtraction, and multiplication using RoCC custom instructions
- Cycle counts computed through reading cycle CSR before and after NTT function call
- Using both 8-point NTT and 1024-point NTT to evaluate performance across input sizes
- Mitigate effect of cold caches; ran 20 trials of NTT kernel and discarded the first 10. Average of the rest of the trials used
- Observed 1.3 – 1.5x speedup in comparison to __int128

Future Work

- Vectorizing the extension using RVV1.0 to extract more parallelism
- Standalone functional unit in an out-of-order processor to avoid overheads associated with RoCC