The Capstone-RISC-V Instruction Set Reference

Table of Contents

Contributors to this document include (in alphabetical order): Jason Zhijingcheng Yu

Version Information: Draft version. Refer to the commit hash.

1. Introduction

The Capstone project is an effort to explore the design of a new CPU instruction set architecture that achieves multiple security goals including memory safety and isolation with one unified hardware abstraction.

1.1. Goals

The ultimate goal of Capstone is to unify the numerous hardware abstracts that have been added as extensions to existing architectures as afterthought mitigations to security vulnerabilities. This goal requires a high level of flexibility and extensibility of the Capstone architecture. More specifically, we aim to support the following in a unified manner.

Exclusive access

Software should be guaranteed exclusive access to certain memory regions if needed. This is in spite of the existence of software traditionally entitled to higher privileges such as the OS kernel and the hypervisor.

Revocable delegation

Software components should be able to delegate authority to other components in a revocable manner. For example, after an untrusted library function has been granted access to a memory region, the caller should be able to revoke this access.

Dynamically extensible hierarchy

The hierarchy of authority should be dynamically extensible, unlike traditional platforms which follow a static hierarchy of hypervisor-kernel-user. This makes it possible to use the same set of abstractions for memory isolation and memory sharing regardless of where a software component lies in the hierarchy.

Safe context switching

A mechanism of context switching without trusting any other software component should be provided. This allows for a minimal TCB if necessary in case of a highly security-critical application.

1.2. Major Design Elements

The Capstone architecture design is based on the idea of capabilities, which are unforgeable tokens that represent authority to perform memory accesses and control flow transfers. Capstone extends the traditional capability model with new capability types including the following.

Linear capabilities

Linear capabilities are guaranteed not to alias with other capabilities. Operations on linear capabilities maintain this property. For example, linear capabilities cannot be duplicated. Instead, they can only be moved around across different registers or between registers and memory. They can hence enable safe exclusive access to memory regions. Capabilities that do not have this property are called *non-linear* capabilities.

Revocation capabilities

Revocation capabilities cannot be used to perform memory accesses or control flow transfers. Instead, they convey the authority to revoke other capabilities. Each revocation capability is derived from a linear capability and can later be used to revoke (i.e., invalidate) capability derived from the same linear capability. This mechanism enables revocable and arbitrarily extensible chains of delegation of authority.

Uninitialised capabilities

Uninitialised capabilities convey write-only authority to memory. They can be turned into linear capabilities after the memory region has been "initialised", that is, when the whole memory region has been overwritten with fresh data. Uninitialised capabilities enable safe initialisation of memory regions and prevent secret leakage without incurring extra performance overhead.

1.3. Capstone-RISC-V ISA Overview

While Capstone does not assume any specific modern ISA, we choose to propose a Capstone extension to RISC-V due to its open nature and the availability of toolchains and simulators.

The Capstone-RISC-V ISA is a 64-bit RISC-V extension that makes the following types of changes to the base architecture:

- Each general-purpose register is extended to 129 bits to accomodate 128-bit capabilities.
- New instructions for manipulating capabilities are added.
- New instructions for memory accesses using capabilities are added.
- New instructions for control flow transfers using capabilities are added.
- Semantics of a small number of existing instructions are changed to support capabilities.
- Semantics of interrupts and exceptions are changed to support capabilities.

1.4. Assembly Mnemonics

Each Capstone-RISC-V instruction is given a mnemonic prefixed with `CS.`. In contexts where it is clear we are discussing Capstone-RISC-V instructions, we will omit the `CS.` prefix for brevity.

In assembly code, the list of operands to an instruction is supplied following the instruction mnemonic, with the operands separated by commas, in the order of `rd`, `rs1`, `rs2`, `imm` for any operand the instruction expects.

1.5. Notations

When specifying the semantics of instructions, we use the following notations to represent the type of each operand:

I

Integer register.

C

Capability register.

S

Sign-extended immediate.

Z

Zero-extended immediate.

1.6. Bibliography

The initial design of Capstone has been discussed in the following paper:

- Capstone: A Capability-based Foundation for Trustless Secure Memory Access by Jason Zhijingcheng Yu, Conrad Watt, Aditya Badole, Trevor E. Carlson, Prateek Saxena. In *Proceedings of the 32nd USENIX Security Symposium*. Annaheim, CA, USA. August 2023.

2. Programming Model

The Capstone-RISC-V ISA has extended the part of the machine state, including both some registers and the memory, to enable the storage and handling of capabilities.

2.1. Capabilities

2.1.1. Width

The width of a capability is 128 bits. We represent this as `CLEN = 128`. Note that this does not affect the width of a raw address, which is `XLEN = 64` bits.

2.1.2. Fields

Each capability has the following architecturally-visible fields:

Table 1. Fields in a capability

Name	Range	Description
`type`	`0..5`	The type of the capability: `0` = linear, `1` = non-linear, `2` = revocation, `3` = uninitialised, `4` = sealed, `5` = sealed-return
`cursor`	`0..2^XLEN-1`	The memory address the capability points to (to be used for the next memory access)
`base`	`0..2^XLEN-1`	The base memory address of the memory region associated with the capability
`length`	`0..2^XLEN-1`	The length of the memory region associated with the capability
`perms`	`0..4`	The permissions associated with the capability: `0` = no access, `1` = read-only, `2` = read-execute, `3` = read-write, `4` = read-write-execute

Note

Implementations are free to maintain additional fields to capabilities or compress the representation of the above fields, as long as each capability fits in `CLEN` bits.

2.2. Extension to General-Purpose Registers

The Capstone-RISC-V ISA extends each of the 32 general-purpose registers, so it contains either a capability or a raw XLEN-bit integer. The type of data contained in a register is maintained and confusion of the type is not allowed. In assembly code, the type of data expected in a register operand is indicated by the alias used for the register, as summarised in the following table.

XLEN-bit integer	Capability
x0/zero	c0/cnull
x1/ra	c1/cra
x2/sp	c2/csp
x3/gp	c3/cgp
x4/tp	c4/ctp
x5/t0	c5/ct0
x6/t1	c6/ct1
x7/t2	c7/ct2
x8/s0/fp	c8/cs0/cfp
x9/s1	c9/cs1
x10/a0	c10/ca0
x11/a1	c11/ca1
x12/a2	c12/ca2
x13/a3	c13/ca3
x14/a4	c14/ca4
x15/a5	c15/ca5
x16/a6	c16/ca6
x17/a7	c17/ca7
x18/s2	c18/cs2
x19/s3	c19/cs3
x20/s4	c20/cs4
x21/s5	c21/cs5
x22/s6	c22/cs6
x23/s7	c23/cs7
x24/s8	c24/cs8
x25/s9	c25/cs9
x26/s10	c26/cs10
x27/s11	c27/cs11

XLEN-bit integer	Capability
x28/t3	c28/ct3
x29/t4	c29/ct4
x30/t5	c30/ct5
x31/t6	c31/ct6

2.3. Extension to Other Registers

The program counter (pc) register is extended to contain a capability.

2.4. Extension to Memory

The memory is addressed using an XLEN-bit integer at byte-level granularity. In addition to raw integers, each CLEN-bit aligned address can also store a capability. The type of data contained in a memory location is maintained and confusion of the type is not allowed.

2.5. Instruction Set

The Capstone-RISC-V instruction set is based on the RV64G instruction set. The (uncompressed) instructions are fixed 32-bit wide, and laid out in memory in little-endian order. In the encoding space of the RV64G instruction set, Capstone-RISC-V instructions occupies the "custom-2" subset, i.e., the opcode of all Capstone-RISC-V instructions is 0b1011011.

Capstone-RISC-V instruction encodings follow two basic formats: R-type and I-type, as described below (more details are also provided in the *RISC-V ISA Manual*).

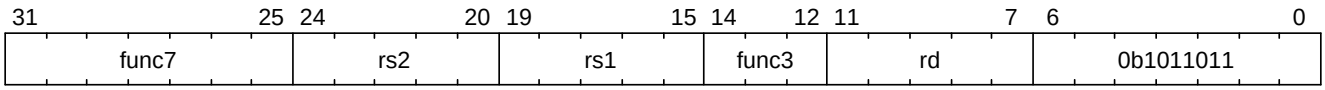

Figure 1. R-type instruction format

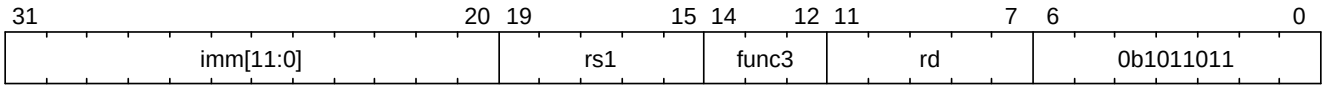

Figure 2. I-type instruction format

R-type instructions receive up to three register operands, and I-type instructions receive up to two register operands and a 12-bit-wide immediate operand.

3. Capability Manipulation Instructions

3.1. Cursor, Bounds, and Permissions Manipulation

3.1.1. Capability Movement

3.1.2. Cursor Increment

The CINCOFFSET and CINCOFFSETIMM instructions increment the `cursor` of a capability by a give amount.

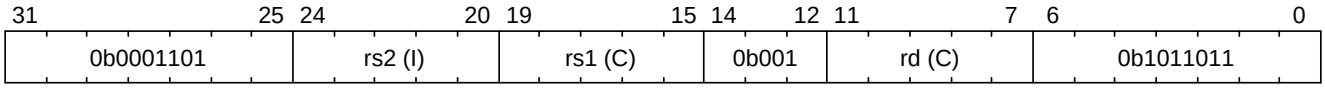

Figure 3. CINCOFFSET instruction format

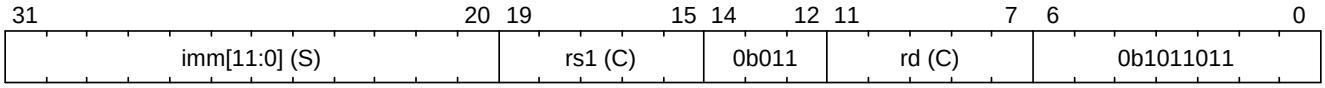

Figure 4. CINCOFFSETIMM instruction format

3.1.3. Cursor Setter and Getter

3.1.4. Bounds Shrinking

3.1.5. Bounds Splitting

3.1.6. Permission Tightening

3.2. Type Manipulation

3.2.1. Delinearisation

3.2.2. Initialisation

3.2.3. Sealing

3.3. Dropping

3.4. Revocation

3.4.1. Revocation Capability Creation

The MREV instruction creates a revocation capability.

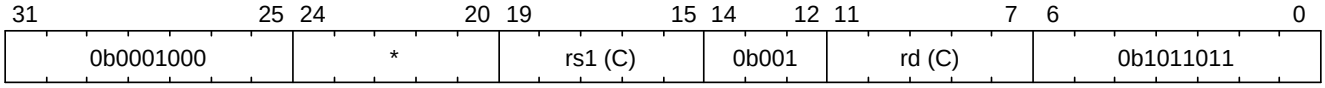

Figure 5. MREV instruction format

3.4.2. Revocation Operation

The REVOKE instruction revokes a capability.

4. Memory Access Instructions

4.1. Load/Store with Capabilities

4.2. Load/Store Capabilities

5. Control Flow Instructions

5.1. Jump to Capabilities

5.2. Domain Crossing

5.3. World Switching

6. Adjustments to Existing Instructions

TODO

7. Interrupts and Exceptions

TODO

8. Memory Consistency Model

TODO

Appendix A: Debugging Instructions

TODO

Appendix B: Instruction Listing

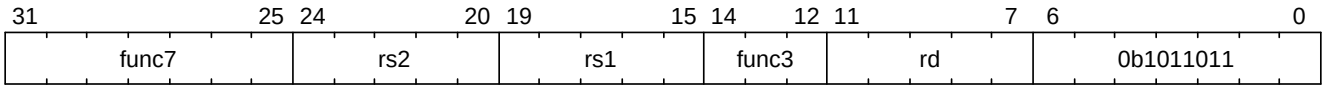

Figure 6. Instruction format: R-type

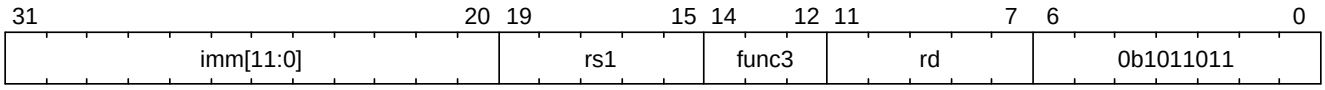

Figure 7. Instruction format: I-type

Table 2. Debugging instructions

Mnemonic	Format	Func3	Func7	rs1	rs2	rd	imm[11:0]
QUERY	R	000	0000000	I	-	-	-
DROP	R	000	0000001	C	-	-	-
RCUPDATE	R	000	0000010	I	-	I	-
ALLOC	R	000	0000011	I	-	I	-
REVOKET	R	000	0000100	I	-	-	-
CAPCREATE	R	000	0000101	-	-	C	-
CAPTYPE	R	000	0000110	I	-	C	-
CAPNODE	R	000	0000111	I	-	C	-
CAPPERM	R	000	0001000	I	-	C	-
CAPBOUND	R	000	0001001	I	I	C	-
CAPPRINT	R	000	0001010	I	-	-	-
TAGSET	R	000	0001011	I	I	-	-
TAGGET	R	000	0001100	I	-	I	-

Table 3. Capability manipulation instructions

Mnemonic	Format	Func3	Func7	rs1	rs2	rd	imm[11:0]
REVOKE	R	001	0000000	C	-	-	-
SHRINK	R	001	0000001	I	I	C	-
TIGHTEN	R	001	0000010	I	-	C	-
DELIN	R	001	0000011	-	-	C	-
LCC	R	001	0000100	C	-	I	-
SCC	R	001	0000101	I	-	C	-
SPLIT	R	001	0000110	C	I	C	-
SEAL	R	001	0000111	-	-	C	-
MREV	R	001	0001000	C	-	C	-

Mnemonic	Format	Func3	Func7	rs1	rs2	rd	imm[11:0]
INIT	R	001	0001001	-	-	C	-
MOVC	R	001	0001010	C	-	C	-
DROPI	R	001	0001011	C	-	-	-
CAPGET	R	001	0001100	-	-	C	-
CINCOFFSET	R	001	0001101	C	C	C	-
CINCOFFSETIMM	I	011	-	C	-	C	S

Table 4. Memory access instructions

Mnemonic	Format	Func3	Func7	rs1	rs2	rd	imm[11:0]
LDC	R	001	0010000	C	-	C	-
STC	R	001	0010001	C	C	-	-
LDD	R	001	0010010	C	-	I	-
STD	R	001	0010011	C	-	I	-
LDW	R	001	0010100	C	-	I	-
STW	R	001	0010101	C	I	-	-
LDH	R	001	0010110	C	-	I	-
STH	R	001	0010111	C	I	-	-
LDB	R	001	0011000	C	-	I	-
STB	R	001	0011001	C	I	-	-

Table 5. Control flow instructions

Mnemonic	Format	Func3	Func7	rs1	rs2	rd	imm[11:0]
CALL	R	001	0100000	-	I	C	-
RETURN	R	001	0100001	-	I	C	-
JMP	R	001	0100010	C	-	-	-
JNZ	R	001	0100011	C	I	-	-
CAPENTER	R	001	0100100	C	-	-	-
CAPEXIT	R	001	0100101	-	I	C	-
CAPEXITSEAL	R	001	0100110	-	I	C	-

Note

For instruction operands:

I

Integer register

C

Capability register

-

Not used

For immediates:

S

Sign-extended

Z

Zero-extended

-

Not used

Appendix C: Assembly Code Examples

TODO

Appendix D: Abstract Binary Interface (Non-Normative)

TODO