Chapter 11. "Zimop" May-Be-Operations Extension, Version 1.0

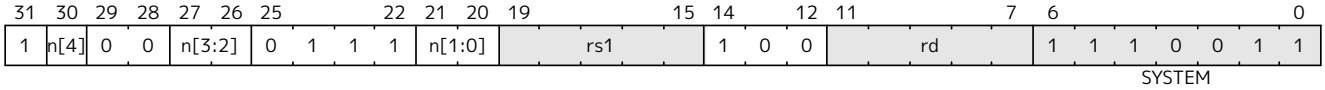
This chapter defines the "Zimop" extension, which introduces the concept of instructions that *may be operations* (MOPs). MOPs are initially defined to simply write zero to `x[rd]`, but are designed to be redefined by later extensions to perform some other action. The Zimop extension defines an encoding space for 40 MOPs.

It is sometimes desirable to define instruction-set extensions whose instructions, rather than raising illegal-instruction exceptions when the extension is not implemented, take no useful action (beyond writing `x[rd]`). For example, programs with control-flow integrity checks can execute correctly on implementations without the corresponding extension, provided the checks are simply ignored. Implementing these checks as MOPs allows the same programs to run on implementations with or without the corresponding extension.

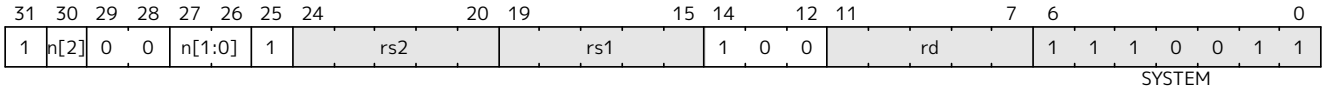
Although similar in some respects to HINTs, MOPs cannot be encoded as HINTs, because unlike HINTs, MOPs are allowed to alter architectural state.

Because MOPs may be redefined by later extensions, standard software should not execute a MOP unless it is deliberately targeting an extension that has redefined that MOP.

The Zimop extension defines 32 MOP instructions named MOP.R.*n*, where *n* is an integer between 0 and 31, inclusive. Unless redefined by another extension, these instructions simply write 0 to `x[rd]`. Their encoding allows future extensions to define them to read `x[rs1]`, as well as write `x[rd]`.



The Zimop extension additionally defines 8 MOP instructions named MOP.RR.*n*, where *n* is an integer between 0 and 7, inclusive. Unless redefined by another extension, these instructions simply write 0 to `x[rd]`. Their encoding allows future extensions to define them to read `x[rs1]` and `x[rs2]`, as well as write `x[rd]`.



The recommended assembly syntax for MOP.R.n is MOP.R.n rd, rs1, with any `x`-register specifier being valid for either argument. Similarly for MOP.RR.n, the recommended syntax is MOP.RR.n rd, rs1, rs2. The extension that redefines a MOP may define an alternate assembly mnemonic.

These MOPs are encoded in the SYSTEM major opcode in part because it is expected their behavior will be modulated by privileged CSR state.

These MOPs are defined to write zero to `x[rd]`, rather than performing no operation, to simplify instruction decoding and to allow testing the presence of features by branching on the zeroness of the result.

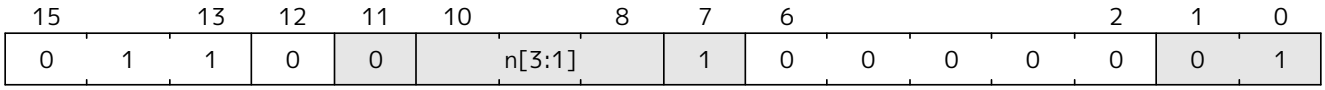
The MOPs defined in the Zimop extension do not carry a syntactic dependency from `x[rs1]` or `x[rs2]` to `x[rd]`, though an extension that redefines the MOP may impose such a requirement.

Not carrying a syntactic dependency relieves straightforward implementations of reading `x[rs1]` and `x[rs2]`.

11.1. "Zcmop" Compressed May-Be-Operations Extension, Version 1.0

This section defines the "Zcmop" extension, which defines eight 16-bit MOP instructions named C.MOP.*n*, where *n* is an odd integer between 1 and 15, inclusive. C.MOP.*n* is encoded in the reserved encoding space corresponding to C.LUI x*n*, 0, as shown in Table 10. Unlike the MOPs defined in the Zimop extension, the C.MOP.*n* instructions are defined to *not* write any register. Their encoding allows future extensions to define them to read register `x[n]`.

The Zcmop extension requires the Zca extension.



Very few suitable 16-bit encoding spaces exist. This space was chosen because it already has unusual behavior with respect to the `rd/rs1` *field—it encodes* `c.addi16sp` *when the field contains* `x2`*--and is therefore of lower value for most purposes.*

Table 10. C.MOP.n instruction encoding.

Mnemonic	Encoding	Redefinable to read register
C.MOP.1	`0110000010000001`	`x1`
C.MOP.3	`0110000110000001`	`x3`
C.MOP.5	`0110001010000001`	`x5`
C.MOP.7	`0110001110000001`	`x7`
C.MOP.9	`0110010010000001`	`x9`
C.MOP.11	`0110010110000001`	`x11`
C.MOP.13	`0110011010000001`	`x13`
C.MOP.15	`0110011110000001`	`x15`

The recommended assembly syntax for C.MOP.n is simply the nullary C.MOP.n. The possibly accessed register is implicitly `xn`*.*

The expectation is that each Zcmop instruction is equivalent to some Zimop instruction, but the choice of expansion (if any) is left to the extension that redefines the MOP. Note, a Zcmop instruction that does not write a value can expand into a write to `x0`*.*