Interpreter Manual

Ætérnal (Sirraide)

16 December 2022

Contents

1 Introduction 3

2 Instruction Set 4

2.1 nop . . . 4

2.2 ret . . . 4

2.3 mov rd, rs/*imm* . . . 4

2.4 add rd, rs1/*imm1*, rs2/*imm2* . . . 4

2.5 sub rd, rs1/*imm1*, rs2/*imm2* . . . 4

2.6 muli rd, rs1/*imm1*, rs2/*imm2* . . . 4

2.7 mulu rd, rs1/*imm1*, rs2/*imm2* . . . 5

2.8 divi rd, rs1/*imm1*, rs2/*imm2* . . . 5

2.9 divu rd, rs1/*imm1*, rs2/*imm2* . . . 5

2.10 remi rd, rs1/*imm1*, rs2/*imm2* . . . 5

2.11 remu rd, rs1/*imm1*, rs2/*imm2* . . . 5

2.12 shl rd, rs1/*imm1*, rs2/*imm2* . . . 6

2.13 shr rd, rs1/*imm1*, rs2/*imm2* . . . 6

2.14 sar rd, rs1/*imm1*, rs2/*imm2* . . . 6

2.15 call ra/*addr* . . . 6

2.16 jmp ra/*addr* . . . 6

2.17 jnz rc, ra/*addr* . . . 6

2.18 ld rd, *imm* . . . 7

2.19 ld rd, rs, *imm* . . . 7

2.20 st *imm*, rs . . . 7

2.21 st rd, *imm*, rs . . . 7

3 Encoding 8

1 Introduction

2 Instruction Set

Each instruction consist of a 1-byte opcode and several operands. For the instruction encoding see § 3.

2.1 nop

This instruction unsurprisingly does nothing.

2.2 ret

This instruction pops the current stack frame off the frame stack and returns from the current function. If called at the top stack frame, it will instead stop the interpreter, returning the value of the return register.

2.3 mov rd, rs/*imm*

This instruction moves an immediate or the value of rs into rd. rd cannot be r0. rs/*imm* is encoded using *r/imm* encoding.

2.4 add rd, rs1/*imm1*, rs2/*imm2*

This instruction adds rs1/*imm1* and rs2/*imm2* and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the sum, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

2.5 sub rd, rs1/*imm1*, rs2/*imm2*

This instruction subtracts rs2/*imm2* from rs1/*imm1* and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the difference, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

2.6 muli rd, rs1/*imm1*, rs2/*imm2*

This instruction multiplies rs1/*imm1* and rs2/*imm2* using signed multiplication and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the product, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

This instruction is identical to mulu, save that the latter performs unsigned multiplication.

2.7 mulu rd, rs1/*imm1*, rs2/*imm2*

This instruction multiplies rs1/*imm1* and rs2/*imm2* using unsigned multiplication and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the product, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

This instruction is identical to muli, save that the latter performs signed multiplication.

2.8 divi rd, rs1/*imm1*, rs2/*imm2*

This instruction divides rs1/*imm1* by rs2/*imm2* using signed division and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the quotient, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

This instruction is identical to divu, save that the latter performs unsigned division.

If rs2/*imm2*, a division error is raised.

2.9 divu rd, rs1/*imm1*, rs2/*imm2*

This instruction divides rs1/*imm1* by rs2/*imm2* using unsigned division and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the quotient, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

This instruction is identical to divi, save that the latter performs signed division.

If rs2/*imm2*, a division error is raised.

2.10 remi rd, rs1/*imm1*, rs2/*imm2*

This instruction divides rs1/*imm1* by rs2/*imm2* using signed division and stores the remainder in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the remainder, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

This instruction is identical to remu, save that the latter performs unsigned division.

If rs2/*imm2*, a division error is raised.

2.11 remu rd, rs1/*imm1*, rs2/*imm2*

This instruction divides rs1/*imm1* by rs2/*imm2* using unsigned division and stores the remainder in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the remainder, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

This instruction is identical to remi, save that the latter performs signed division.

If rs2/*imm2*, a division error is raised.

2.12 shl rd, rs1/*imm1*, rs2/*imm2*

This instruction shifts rs1/*imm1* left by rs2/*imm2* and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the result, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

Only the lower 6 bits of rs2/*imm2* are actually used.

2.13 shr rd, rs1/*imm1*, rs2/*imm2*

This instruction shifts rs1/*imm1* right by rs2/*imm2* and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the result, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

This instruction performs a logical (unsigned) shift.

Only the lower 6 bits of rs2/*imm2* are actually used.

2.14 sar rd, rs1/*imm1*, rs2/*imm2*

This instruction shifts rs1/*imm1* right by rs2/*imm2* and stores the result in rd. At most one of the two source values may be an immediate; the immediate, if any, is encoded using *r/imm* encoding. If the size of rd is smaller than the result, the result will be truncated. Disregarding *r/imm* encoding, no register may be r0.

This instruction performs an arithmetic (signed) shift.

Only the lower 6 bits of rs2/*imm2* are actually used.

2.15 call ra/*addr*

This instruction calls the function at *addr* or in register ra. The function in ra/*addr* is encoded using *r/addr* encoding. For argument/return registers see § 3.

2.16 jmp ra/*addr*

This instruction unconditionally jumps to *addr* or the address in register ra. The target address ra/*addr* is encoded using *r/addr* encoding.

2.17 jnz rc, ra/*addr*

This instruction performs a conditional jumps on the value of rc to *addr* or the address in register ra. The target address ra/*addr* is encoded using *r/addr* encoding.

2.18 ld rd, *imm*

This instruction loads a value at *imm* into register **rd**. The size of **rd** determines the number of bytes loaded from the address.

2.19 ld rd, rs, *imm*

This instruction loads a value from the address stored in **rs** offset by *imm* into register **rd**. The size of **rd** determines the number of bytes loaded from the address.

2.20 st *imm*, rs

This instruction stores the value in register **rs** into memory at the address *imm*. The size of **rs** determines the number of bytes stored to the address.

2.21 st rd, *imm*, rs

This instruction stores the value in register **rs** to the address in **rd** offset by *imm*. The size of **rs** determines the number of bytes loaded from the address.

3 Encoding