The RISC-V Instruction Set Manual Volume I

Unprivileged Architecture

Version 20240411

Table of Contents

Preamble

Contributors to all versions of the spec in alphabetical order (please contact editors to suggest corrections): Derek Atkins, Arvind, Krste Asanović, Rimas Avižienis, Jacob Bachmeyer, Christopher F. Batten, Allen J. Baum, Abel Bernabeu, Alex Bradbury, Scott Beamer, Hans Boehm, Preston Briggs, Christopher Celio, Chuanhua Chang, David Chisnall, Paul Clayton, Palmer Dabbelt, L Peter Deutsch, Ken Dockser, Paul Donahue, Aaron Durbin, Roger Espasa, Greg Favor, Andy Glew, Shaked Flur, Stefan Freudenberger, Marc Gauthier, Andy Glew, Jan Gray, Gianluca Guida, Michael Hamburg, John Hauser, John Ingalls, David Horner, Bruce Hoult, Bill Huffman, Alexandre Joannou, Olof Johansson, Ben Keller, David Kruckemyer, Tariq Kurd, Yunsup Lee, Paul Loewenstein, Daniel Lustig, Yatin Manerkar, Luc Maranget, Ben Marshall, Margaret Martonosi, Phil McCoy, Nathan Menhorn, Christoph Müllner, Joseph Myers, Vijayanand Nagarajan, Rishiyur Nikhil, Jonas Oberhauser, Stefan O'Rear, Markku-Juhani O. Saarinen, Albert Ou, John Ousterhout, Daniel Page, David Patterson, Christopher Pulte, Jose Renau, Josh Scheid, Colin Schmidt, Peter Sewell, Susmit Sarkar, Ved Shanbhogue, Brent Spinney, Brendan Sweeney, Michael Taylor, Wesley Terpstra, Matt Thomas, Tommy Thorn, Philipp Tomsich, Caroline Trippel, Ray VanDeWalker, Muralidaran Vijayaraghavan, Megan Wachs, Paul Wamsley, Andrew Waterman, Robert Watson, David Weaver, Derek Williams, Claire Wolf, Andrew Wright, Reinoud Zandijk, and Sizhuo Zhang.

This document is a derivative of "The RISC-V Instruction Set Manual, Volume I: User-Level ISA Version 2.1" released under the following license: ©2010-2017 Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanović. Creative Commons Attribution 4.0 International License. Please cite as: "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191214-draft", Editors Andrew Waterman and Krste Asanović, RISC-V Foundation, December 2019.

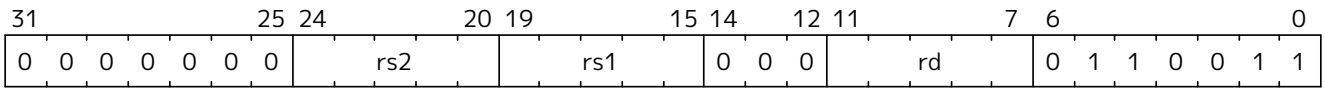
1. ADD

Synopsis

Mnemonic

```
add rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_ADD(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = rs1_val + rs2_val,
  X(rd) = result;
  RETIRE_SUCCESS
}
```

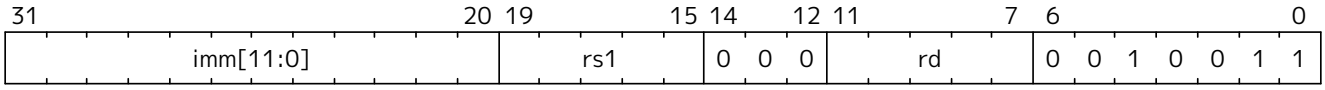
2. ADDI

Synopsis

Mnemonic

```
addi rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

```
function clause execute (RISCV_ADDI (imm, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let immext : xlenbits = sign_extend(imm);
  let result = rs1_val + immext,
  X(rd) = result;
  RETIRE_SUCCESS
}
```

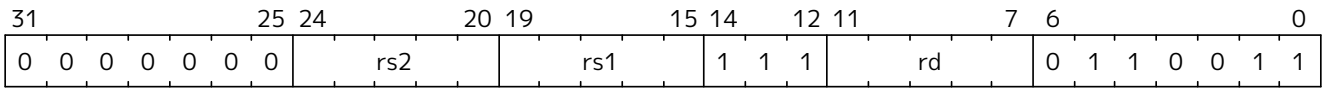
3. AND

Synopsis

Mnemonic

```
and rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_AND(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = rs1_val & rs2_val,
  X(rd) = result;
  RETIRE_SUCCESS
}
```

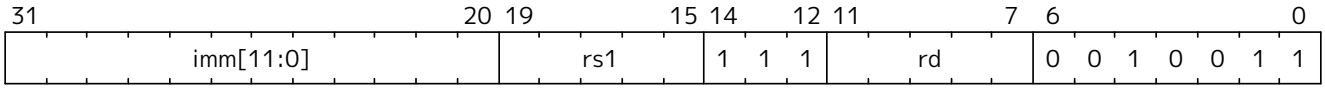
4. ANDI

Synopsis

Mnemonic

```
andi rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

```
function clause execute (RISCV_ANDI (imm, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let immext : xlenbits = sign_extend(imm);
  let result = rs1_val & immext,
  X(rd) = result;
  RETIRE_SUCCESS
}
```

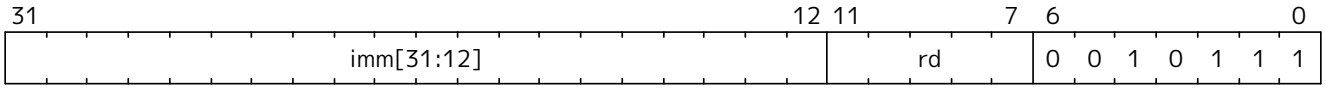
5. AUIPC

Synopsis

Mnemonic

```
auipc rd, imm[31:12]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
imm20	input	20-bit immediate

Sail Code

```
function clause execute RISCV_AUIPC(imm, rd, op) = {
  let off : xlenbits = sign_extend(imm @ 0x000);
  let ret = get_arch_pc() + off
  X(rd) = ret;
  RETIRE_SUCCESS
}
```

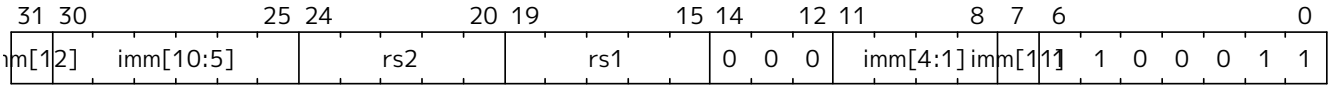
6. BEQ

Synopsis

Mnemonic

```
beq imm[12|10:5], rs1, rs2, imm[4:1|11]
```

Encoding



Description

Arguments

Register	Direction	Definition
bimm12hi	input	High bits of 13-bit branch offset
rs1	input	Source register 1
rs2	input	Source register 2
bimm12lo	input	Low bits of 13-bit branch offset

Sail Code

```
function clause execute (RISCV_BEQ(imm, rs2, rs1, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let taken = rs1_val == rs2_val,
  let t : xlenbits = PC + sign_extend(imm);
  if taken then {
    /* Extensions get the first checks on the prospective target
address. */
    match ext_control_check_pc(t) {
      Ext_ControlAddr_Error(e) => {
        ext_handle_control_check_error(e);
        RETIRE_FAIL
      },
      Ext_ControlAddr_OK(target) => {
        if bit_to_bool(target[1]) & not(extensionEnabled(Ext_C)) then {
          handle_mem_exception(target, E_Fetch_Addr_Align());
          RETIRE_FAIL;
        } else {
          set_next_pc(target);
          RETIRE_SUCCESS
        }
      }
    }
  } else RETIRE_SUCCESS
```

```
}
```

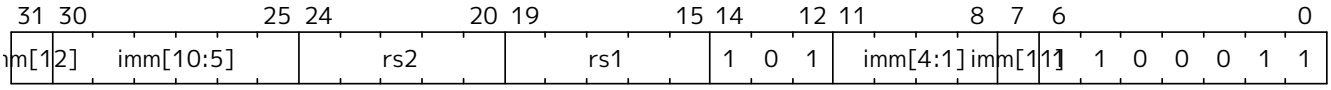
7. BGE

Synopsis

Mnemonic

```
bge imm[12|10:5], rs1, rs2, imm[4:1|11]
```

Encoding



Description

Arguments

Register	Direction	Definition
bimm12hi	input	High bits of 13-bit branch offset
rs1	input	Source register 1
rs2	input	Source register 2
bimm12lo	input	Low bits of 13-bit branch offset

Sail Code

```
function clause execute (RISCV_BGE(imm, rs2, rs1, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let taken = rs1_val >=_s rs2_val,
  let t : xlenbits = PC + sign_extend(imm);
  if taken then {
    /* Extensions get the first checks on the prospective target
address. */
    match ext_control_check_pc(t) {
      Ext_ControlAddr_Error(e) => {
        ext_handle_control_check_error(e);
        RETIRE_FAIL
      },
      Ext_ControlAddr_OK(target) => {
        if bit_to_bool(target[1]) & not(extensionEnabled(Ext_C)) then {
          handle_mem_exception(target, E_Fetch_Addr_Align());
          RETIRE_FAIL;
        } else {
          set_next_pc(target);
          RETIRE_SUCCESS
        }
      }
    }
  } else RETIRE_SUCCESS
```

```
}
```

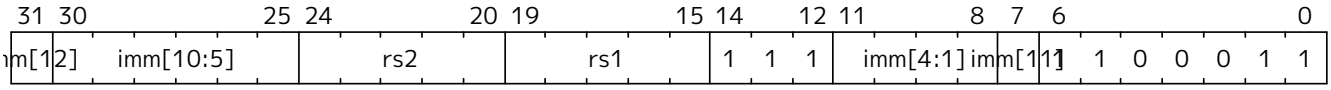
8. BGEU

Synopsis

Mnemonic

```
bgeu imm[12|10:5], rs1, rs2, imm[4:1|11]
```

Encoding



Description

Arguments

Register	Direction	Definition
bimm12hi	input	High bits of 13-bit branch offset
rs1	input	Source register 1
rs2	input	Source register 2
bimm12lo	input	Low bits of 13-bit branch offset

Sail Code

```
function clause execute (RISCV_BGEU(imm, rs2, rs1, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let taken = rs1_val >=_u rs2_val
  let t : xlenbits = PC + sign_extend(imm);
  if taken then {
    /* Extensions get the first checks on the prospective target
address. */
    match ext_control_check_pc(t) {
      Ext_ControlAddr_Error(e) => {
        ext_handle_control_check_error(e);
        RETIRE_FAIL
      },
      Ext_ControlAddr_OK(target) => {
        if bit_to_bool(target[1]) & not(extensionEnabled(Ext_C)) then {
          handle_mem_exception(target, E_Fetch_Addr_Align());
          RETIRE_FAIL;
        } else {
          set_next_pc(target);
          RETIRE_SUCCESS
        }
      }
    }
  } else RETIRE_SUCCESS
```

```
}
```

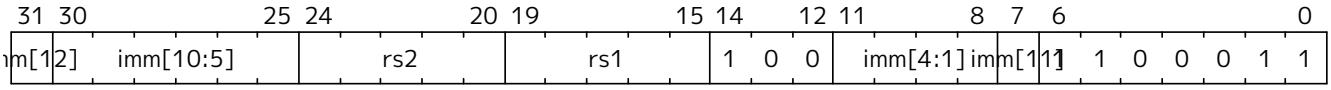
9. BLT

Synopsis

Mnemonic

```
blt imm[12|10:5], rs1, rs2, imm[4:1|11]
```

Encoding



Description

Arguments

Register	Direction	Definition
bimm12hi	input	High bits of 13-bit branch offset
rs1	input	Source register 1
rs2	input	Source register 2
bimm12lo	input	Low bits of 13-bit branch offset

Sail Code

```
function clause execute (RISCV_BLT(imm, rs2, rs1, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let taken = rs1_val <_s rs2_val,
  let t : xlenbits = PC + sign_extend(imm);
  if taken then {
    /* Extensions get the first checks on the prospective target
address. */
    match ext_control_check_pc(t) {
      Ext_ControlAddr_Error(e) => {
        ext_handle_control_check_error(e);
        RETIRE_FAIL
      },
      Ext_ControlAddr_OK(target) => {
        if bit_to_bool(target[1]) & not(extensionEnabled(Ext_C)) then {
          handle_mem_exception(target, E_Fetch_Addr_Align());
          RETIRE_FAIL;
        } else {
          set_next_pc(target);
          RETIRE_SUCCESS
        }
      }
    }
  } else RETIRE_SUCCESS
```

```
}
```

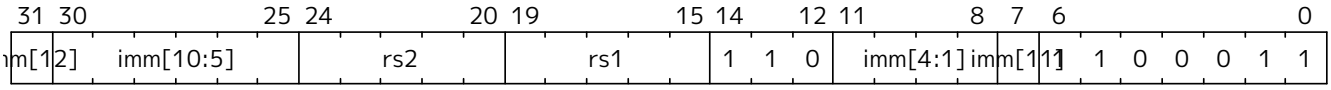
10. BLTU

Synopsis

Mnemonic

```
bltu imm[12|10:5], rs1, rs2, imm[4:1|11]
```

Encoding



Description

Arguments

Register	Direction	Definition
bimm12hi	input	High bits of 13-bit branch offset
rs1	input	Source register 1
rs2	input	Source register 2
bimm12lo	input	Low bits of 13-bit branch offset

Sail Code

```
function clause execute (RISCV_BLTU(imm, rs2, rs1, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let taken = rs1_val <_u rs2_val,
  let t : xlenbits = PC + sign_extend(imm);
  if taken then {
    /* Extensions get the first checks on the prospective target
address. */
    match ext_control_check_pc(t) {
      Ext_ControlAddr_Error(e) => {
        ext_handle_control_check_error(e);
        RETIRE_FAIL
      },
      Ext_ControlAddr_OK(target) => {
        if bit_to_bool(target[1]) & not(extensionEnabled(Ext_C)) then {
          handle_mem_exception(target, E_Fetch_Addr_Align());
          RETIRE_FAIL;
        } else {
          set_next_pc(target);
          RETIRE_SUCCESS
        }
      }
    }
  } else RETIRE_SUCCESS
```

```
}
```

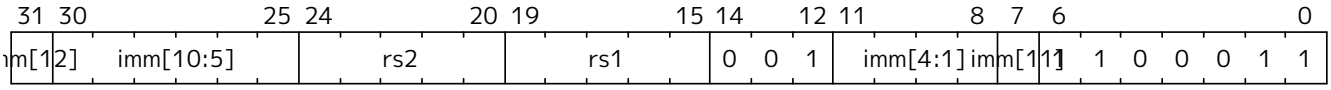
11. BNE

Synopsis

Mnemonic

```
bne imm[12|10:5], rs1, rs2, imm[4:1|11]
```

Encoding



Description

Arguments

Register	Direction	Definition
bimm12hi	input	High bits of 13-bit branch offset
rs1	input	Source register 1
rs2	input	Source register 2
bimm12lo	input	Low bits of 13-bit branch offset

Sail Code

```
function clause execute (RISCV_BNE(imm, rs2, rs1, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let taken = rs1_val != rs2_val,
  let t : xlenbits = PC + sign_extend(imm);
  if taken then {
    /* Extensions get the first checks on the prospective target
address. */
    match ext_control_check_pc(t) {
      Ext_ControlAddr_Error(e) => {
        ext_handle_control_check_error(e);
        RETIRE_FAIL
      },
      Ext_ControlAddr_OK(target) => {
        if bit_to_bool(target[1]) & not(extensionEnabled(Ext_C)) then {
          handle_mem_exception(target, E_Fetch_Addr_Align());
          RETIRE_FAIL;
        } else {
          set_next_pc(target);
          RETIRE_SUCCESS
        }
      }
    }
  } else RETIRE_SUCCESS
```

```
}
```

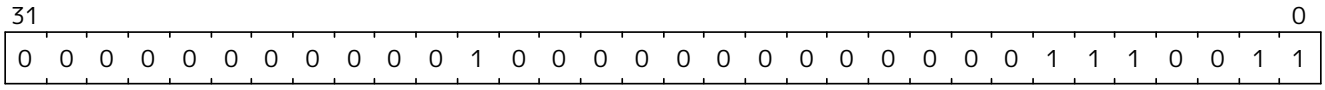
12. EBREAK

Synopsis

Mnemonic

```
ebreak
```

Encoding



Description

Arguments

Register	Direction	Definition

Sail Code

```
function clause execute EBREAK() = {
  handle_mem_exception(PC, E_Breakpoint());
  RETIRE_FAIL
}
```

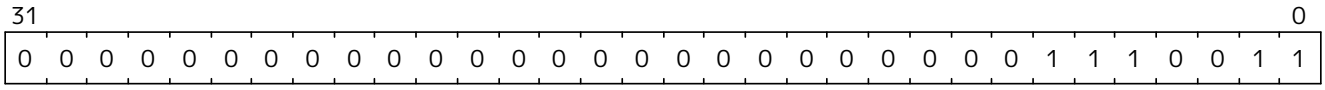
13. ECALL

Synopsis

Mnemonic

```
ecall
```

Encoding



Description

Arguments

Register	Direction	Definition

Sail Code

```
function clause execute ECALL() = {
  let t : sync_exception =
    struct { trap = match (cur_privilege) {
                      User       => E_U_EnvCall(),
                      Supervisor => E_S_EnvCall(),
                      Machine    => E_M_EnvCall()
                    },
             excinfo = (None() : option(xlenbits)),
             ext     = None() };
  set_next_pc(exception_handler(cur_privilege, CTL_TRAP(t), PC));
  RETIRE_FAIL
}
```

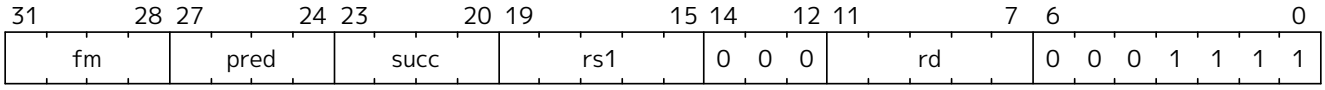
14. FENCE

Synopsis

Mnemonic

```
fence fm, pred, succ, rs1, rd
```

Encoding



Description

Arguments

Register	Direction	Definition
rs1	input	Source register 1
rd	output	Destination register

Sail Code

```
function clause execute (FENCE(pred, succ)) = {
  // If the FIOM bit in menvcfg/senvcfg is set then the I/O bits can
imply R/W.
  let fiom = is_fiom_active();
  let pred = effective_fence_set(pred, fiom);
  let succ = effective_fence_set(succ, fiom);

  match (pred, succ) {
    (_ : bits(2) @ 0b11, _ : bits(2) @ 0b11) =>
sail_barrier(Barrier_RISCV_rw_rw),
    (_ : bits(2) @ 0b10, _ : bits(2) @ 0b11) =>
sail_barrier(Barrier_RISCV_r_rw),
    (_ : bits(2) @ 0b10, _ : bits(2) @ 0b10) =>
sail_barrier(Barrier_RISCV_r_r),
    (_ : bits(2) @ 0b11, _ : bits(2) @ 0b01) =>
sail_barrier(Barrier_RISCV_rw_w),
    (_ : bits(2) @ 0b01, _ : bits(2) @ 0b01) =>
sail_barrier(Barrier_RISCV_w_w),
    (_ : bits(2) @ 0b01, _ : bits(2) @ 0b11) =>
sail_barrier(Barrier_RISCV_w_rw),
    (_ : bits(2) @ 0b11, _ : bits(2) @ 0b10) =>
sail_barrier(Barrier_RISCV_rw_r),
    (_ : bits(2) @ 0b10, _ : bits(2) @ 0b01) =>
sail_barrier(Barrier_RISCV_r_w),
    (_ : bits(2) @ 0b01, _ : bits(2) @ 0b10) =>
sail_barrier(Barrier_RISCV_w_r),
```

```
    (_ : bits(4)       , _ : bits(2) @ 0b00) => (),
    (_ : bits(2) @ 0b00, _ : bits(4)       ) => (),

    _ => { print("FIXME: unsupported fence");
           () }
  };
  RETIRE_SUCCESS
}
```

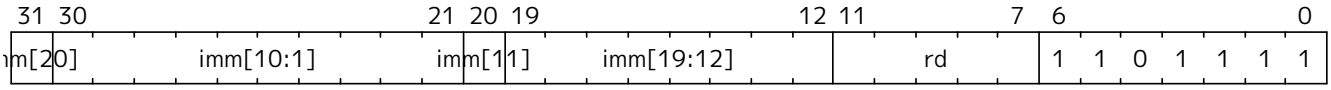
15. JAL

Synopsis

Mnemonic

```
jal rd, imm[20|10:1|11|19:12]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
jimm20	input	20-bit jump offset

Sail Code

```
function clause execute (RISCV_JAL(imm, rd)) = {
  let t : xlenbits = PC + sign_extend(imm);
  /* Extensions get the first checks on the prospective target address.
*/
  match ext_control_check_pc(t) {
    Ext_ControlAddr_Error(e) => {
      ext_handle_control_check_error(e);
      RETIRE_FAIL
    },
    Ext_ControlAddr_OK(target) => {
      /* Perform standard alignment check */
      if bit_to_bool(target[1]) & not(extensionEnabled(Ext_C))
      then {
        handle_mem_exception(target, E_Fetch_Addr_Align());
        RETIRE_FAIL
      } else {
        X(rd) = get_next_pc();
        set_next_pc(target);
        RETIRE_SUCCESS
      }
    }
  }
}
```

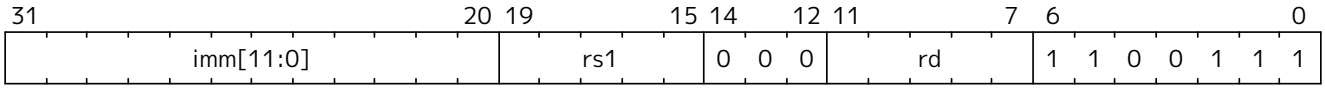
16. JALR

Synopsis

Mnemonic

```
jalr rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

Instruction jalr sail code not found in the expected format.

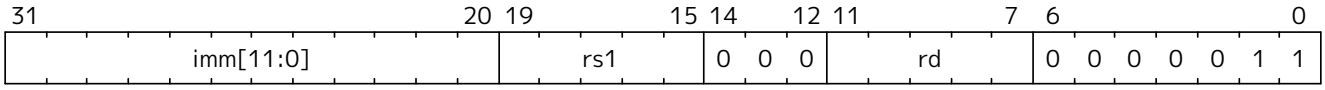
17. LB

Synopsis

Mnemonic

```
lb rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

Instruction lb sail code not found in the expected format.

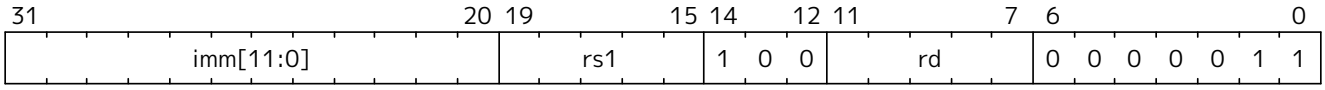
18. LBU

Synopsis

Mnemonic

```
lbu rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

Instruction lbu sail code not found in the expected format.

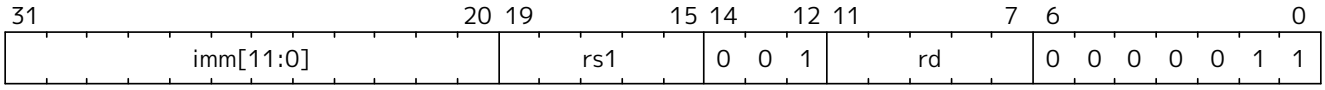
19. LH

Synopsis

Mnemonic

```
lh rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

Instruction lh sail code not found in the expected format.

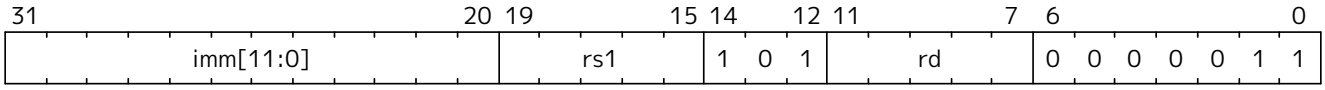
20. LHU

Synopsis

Mnemonic

```
lhu rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

Instruction lhu sail code not found in the expected format.

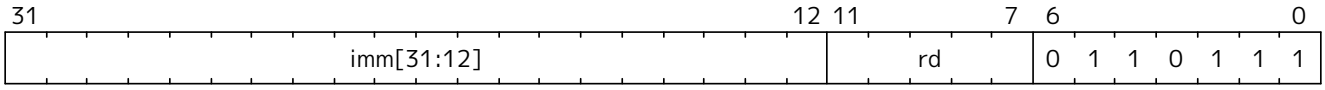
21. LUI

Synopsis

Mnemonic

```
lui rd, imm[31:12]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
imm20	input	20-bit immediate

Sail Code

```
function clause execute RISCV_LUI(imm, rd, op) = {
  let off : xlenbits = sign_extend(imm @ 0x000);
  let ret = off,
  X(rd) = ret;
  RETIRE_SUCCESS
}
```

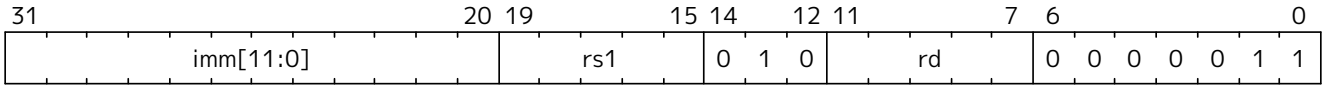
22. LW

Synopsis

Mnemonic

```
lw rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

Instruction lw sail code not found in the expected format.

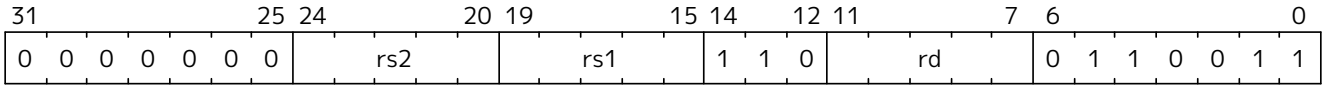
23. OR

Synopsis

Mnemonic

```
or rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_OR(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = rs1_val | rs2_val,
  X(rd) = result;
  RETIRE_SUCCESS
}
```

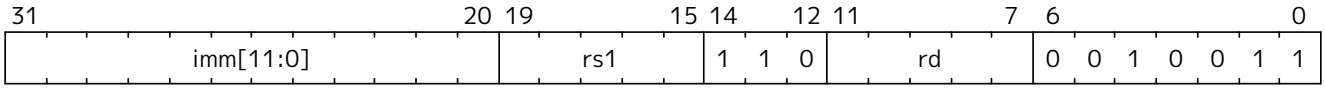
24. ORI

Synopsis

Mnemonic

```
ori rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

```
function clause execute (RISCV_ORI (imm, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let immext : xlenbits = sign_extend(imm);
  let result = rs1_val | immext,
  X(rd) = result;
  RETIRE_SUCCESS
}
```

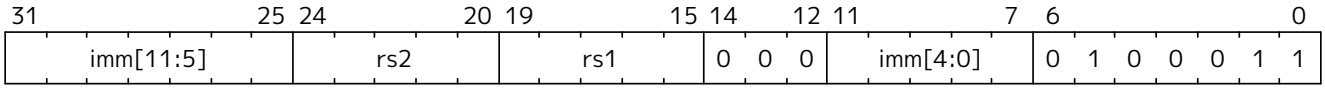
25. SB

Synopsis

Mnemonic

```
sb imm[11:5], rs1, rs2, imm[4:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

Instruction sb sail code not found in the expected format.

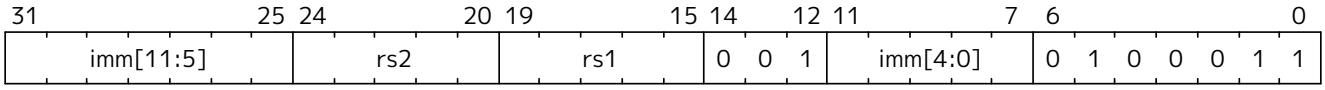
26. SH

Synopsis

Mnemonic

```
sh imm[11:5], rs1, rs2, imm[4:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

Instruction sh sail code not found in the expected format.

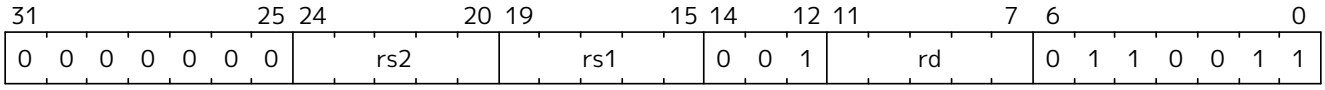
27. SLL

Synopsis

Mnemonic

```
sll rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_SLL(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = if   sizeof(xlen) == 32
                 then rs1_val << (rs2_val[4..0])
                 else rs1_val << (rs2_val[5..0]),
  X(rd) = result;
  RETIRE_SUCCESS
}
```

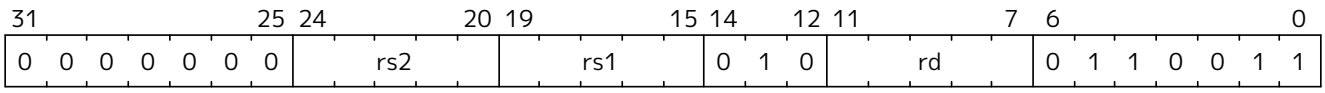
28. SLT

Synopsis

Mnemonic

```
slt rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_SLT(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = zero_extend(bool_to_bits(rs1_val <_s rs2_val)),
  let result = zero_extend(bool_to_bits(rs1_val <_u rs2_val)),
  X(rd) = result;
  RETIRE_SUCCESS
}
```

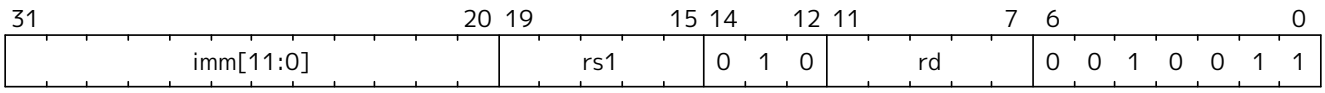
29. SLTI

Synopsis

Mnemonic

```
slti rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

```
function clause execute (RISCV_SLTI (imm, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let immext : xlenbits = sign_extend(imm);
  let result = zero_extend(bool_to_bits(rs1_val <_s immext)),
  let result = zero_extend(bool_to_bits(rs1_val <_u immext)),
  X(rd) = result;
  RETIRE_SUCCESS
}
```

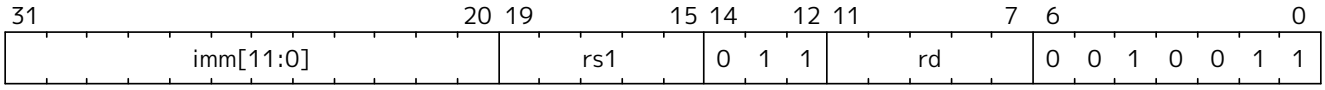
30. SLTIU

Synopsis

Mnemonic

```
sltiu rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

```
function clause execute (RISCV_SLTIU (imm, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let immext : xlenbits = sign_extend(imm);
  let result = zero_extend(bool_to_bits(rs1_val <_u immext)),
  X(rd) = result;
  RETIRE_SUCCESS
}
```

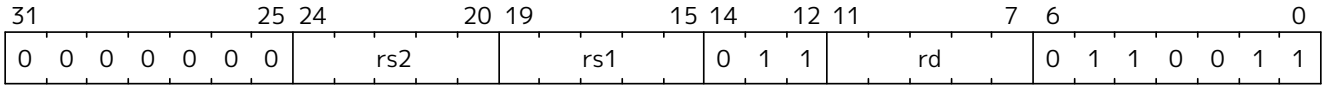
31. SLTU

Synopsis

Mnemonic

```
sltu rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_SLTU(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = zero_extend(bool_to_bits(rs1_val <_u rs2_val)),
  X(rd) = result;
  RETIRE_SUCCESS
}
```

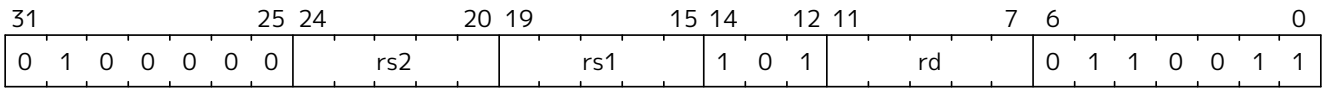
32. SRA

Synopsis

Mnemonic

```
sra rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_SRA(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = if   sizeof(xlen) == 32
                 then shift_right_arith32(rs1_val, rs2_val[4..0])
                 else shift_right_arith64(rs1_val, rs2_val[5..0])
  X(rd) = result;
  RETIRE_SUCCESS
}
```

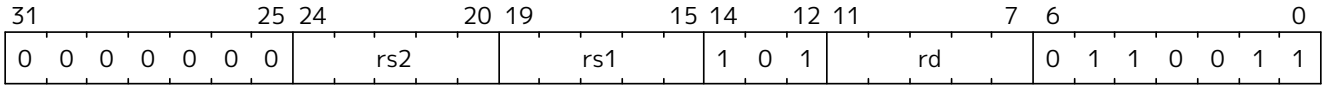
33. SRL

Synopsis

Mnemonic

```
srl rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_SRL(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = if   sizeof(xlen) == 32
                 then rs1_val >> (rs2_val[4..0])
                 else rs1_val >> (rs2_val[5..0]),
  X(rd) = result;
  RETIRE_SUCCESS
}
```

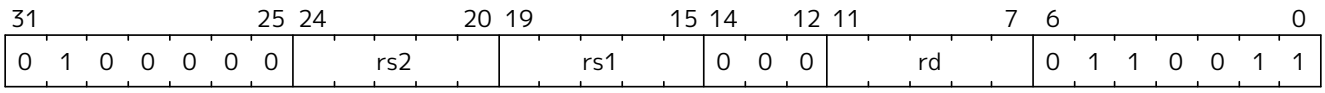
34. SUB

Synopsis

Mnemonic

```
sub rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_SUB(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = rs1_val - rs2_val,
  X(rd) = result;
  RETIRE_SUCCESS
}
```

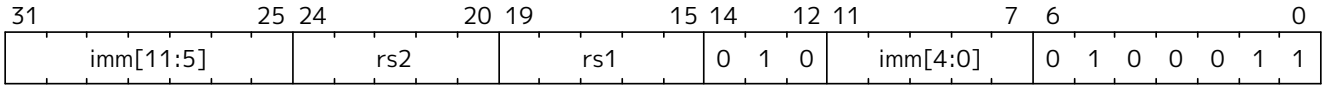
35. SW

Synopsis

Mnemonic

```
sw imm[11:5], rs1, rs2, imm[4:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

Instruction sw sail code not found in the expected format.

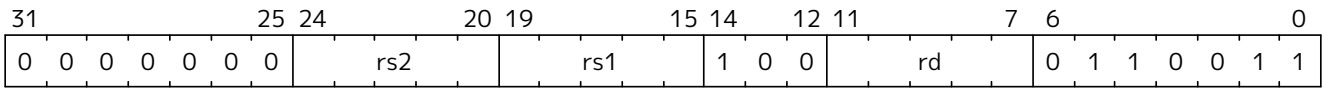
36. XOR

Synopsis

Mnemonic

```
xor rd, rs1, rs2
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
rs2	input	Source register 2

Sail Code

```
function clause execute (RISCV_XOR(rs2, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result = rs1_val ^ rs2_val,
  X(rd) = result;
  RETIRE_SUCCESS
}
```

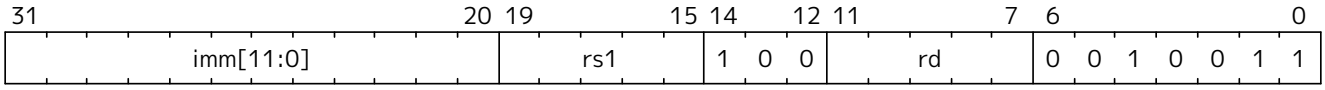
37. XORI

Synopsis

Mnemonic

```
xori rd, rs1, imm[11:0]
```

Encoding



Description

Arguments

Register	Direction	Definition
rd	output	Destination register
rs1	input	Source register 1
imm12	input	12-bit immediate

Sail Code

```
function clause execute (RISCV_XORI (imm, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let immext : xlenbits = sign_extend(imm);
  let result = rs1_val ^ immext
  X(rd) = result;
  RETIRE_SUCCESS
}
```


RISC-V®