Adding Your Own CPU Instructions

Presented by

Ayaz Akram

Let's start building gem5

```
> scons build/RISCV/gem5.opt -j17
```

Outline

- **What is an ISA**
- ISA and CPU independence
- high-level concepts to understand ISA implementation in gem5
- Journey of instructions in gem5
- gem5 ISA parser
- Adding your own instructions in gem5
- Testing your instructions

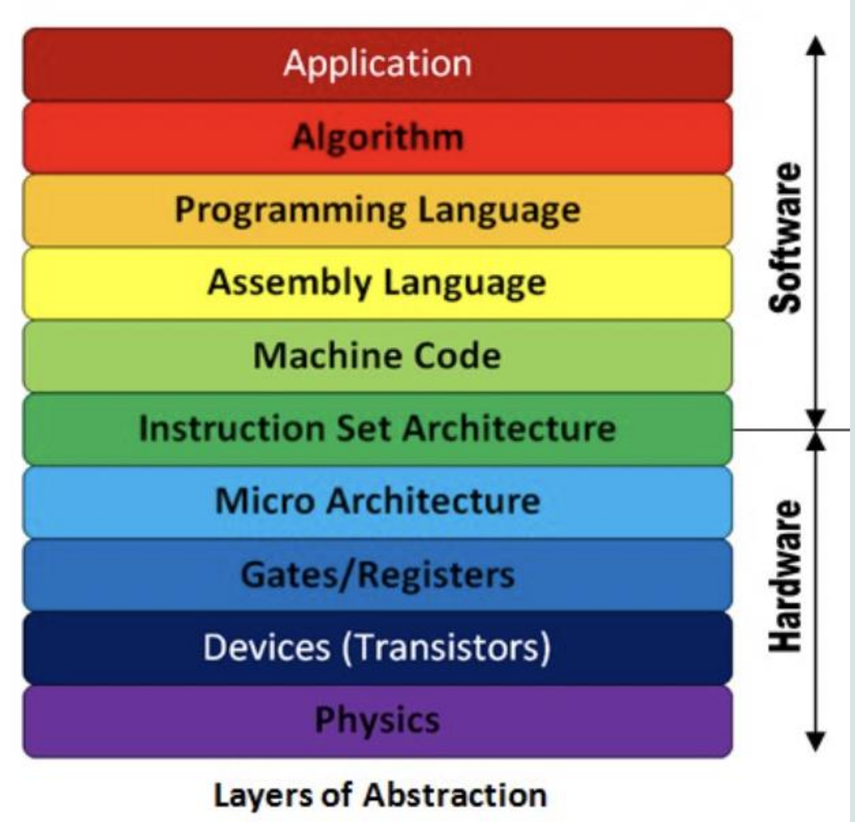
What is an ISA?

What is an ISA?

Interface between the hardware and the software

Components of an ISA

- **Instructions** that a processor can execute
- Registers
- Memory model
- Exception handling



Layers of Abstraction

Outline

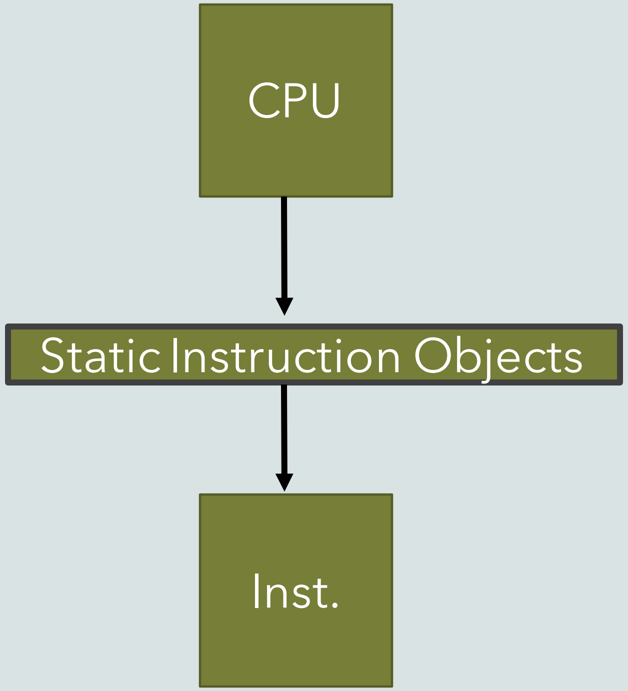
- What is an ISA
- **ISA and CPU independence**
- high-level concepts to understand ISA implementation in gem5
- Journey of instructions in gem5
- gem5 ISA parser
- Adding your own instructions in gem5
- Testing your instructions

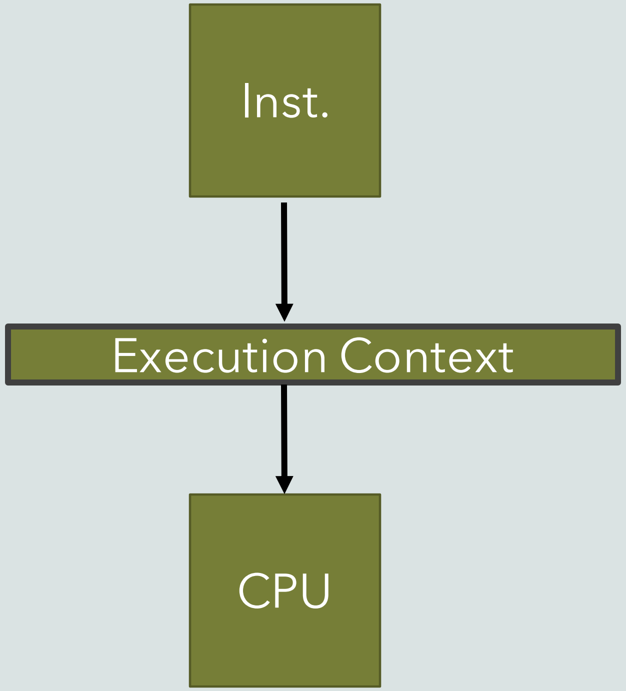
ISA & CPU Independence

gem5

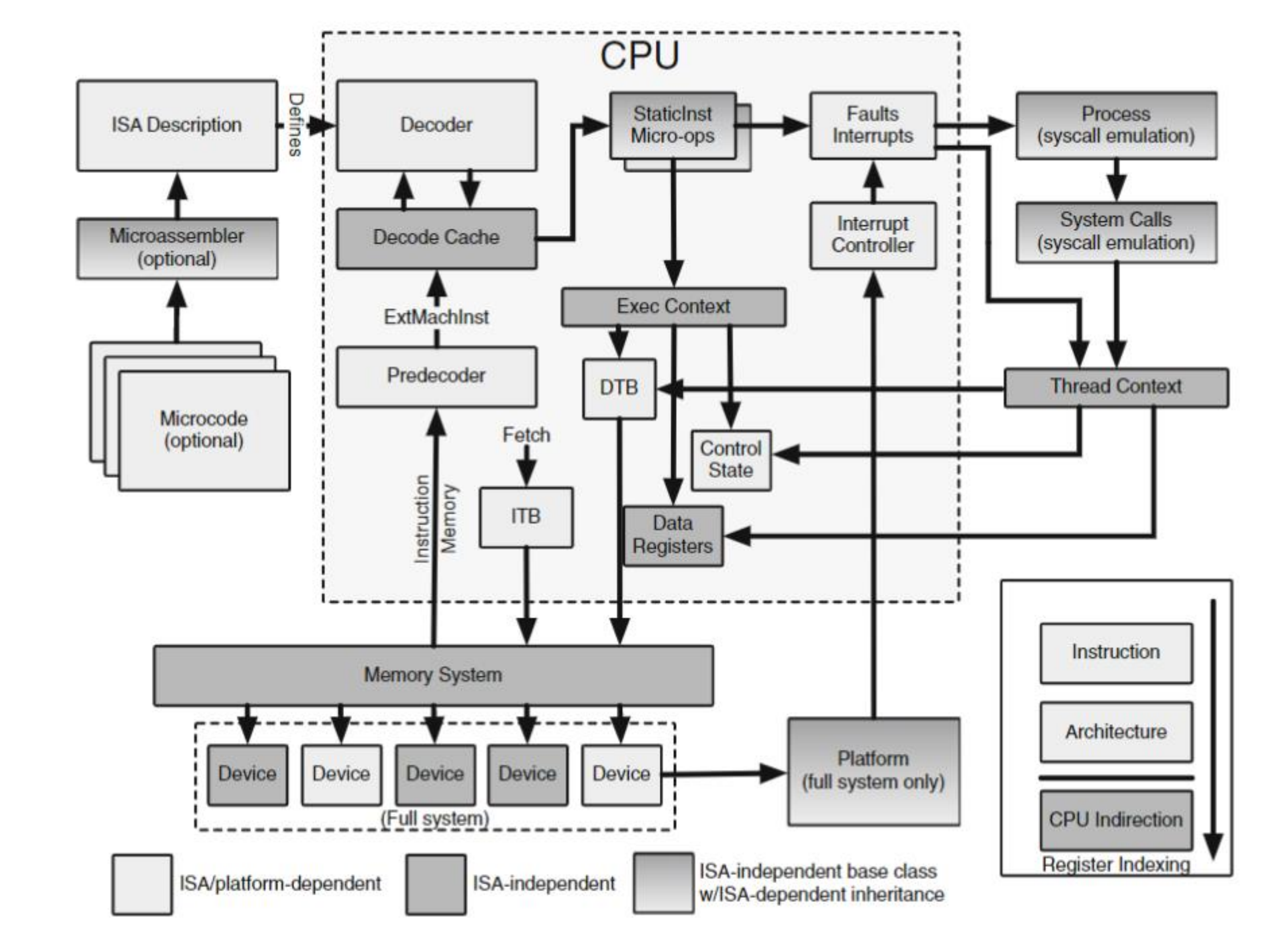
Instruction Behavior

- To make CPU models ISA independent, gem5 relies on **two generic interfaces**

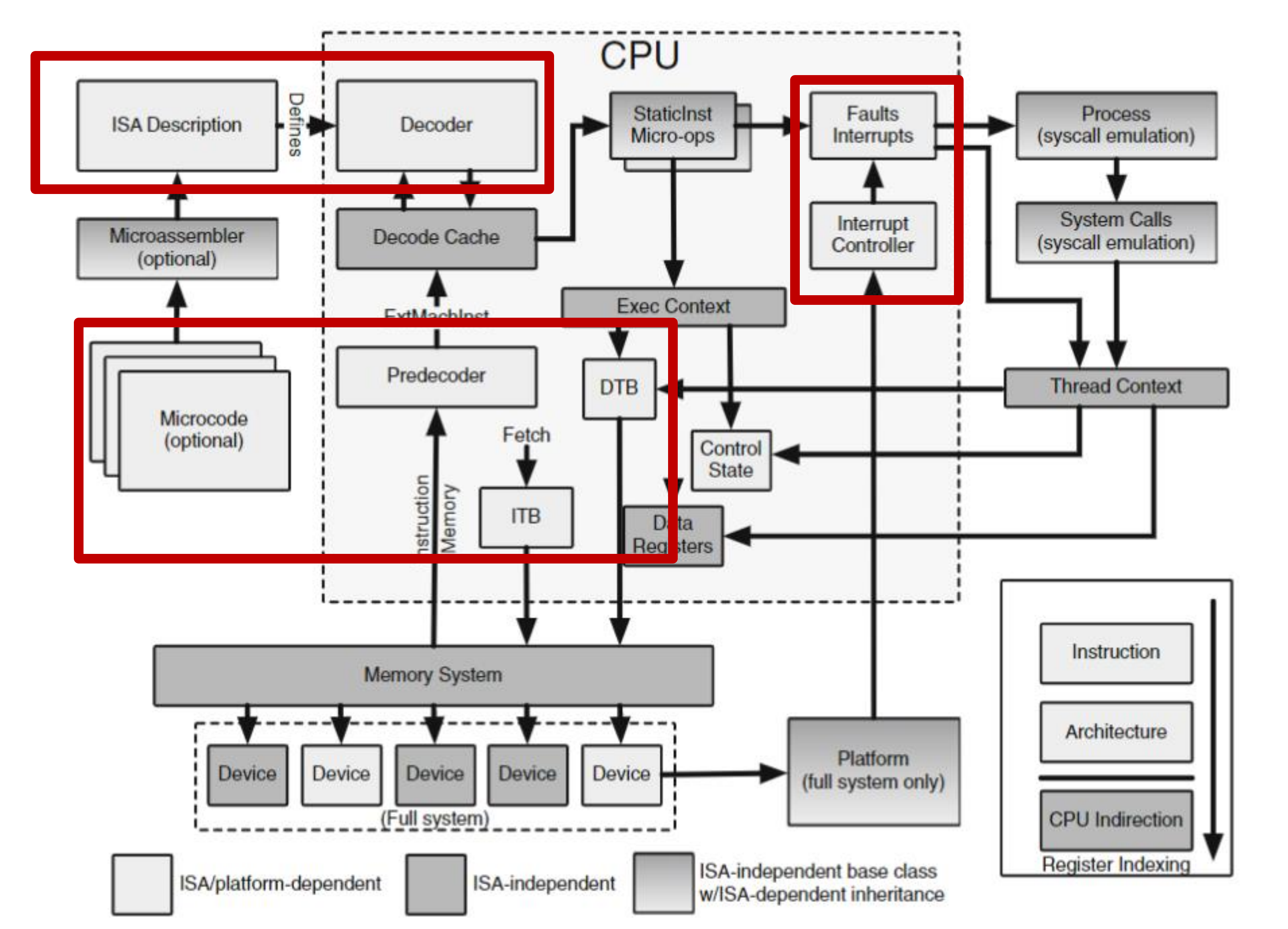




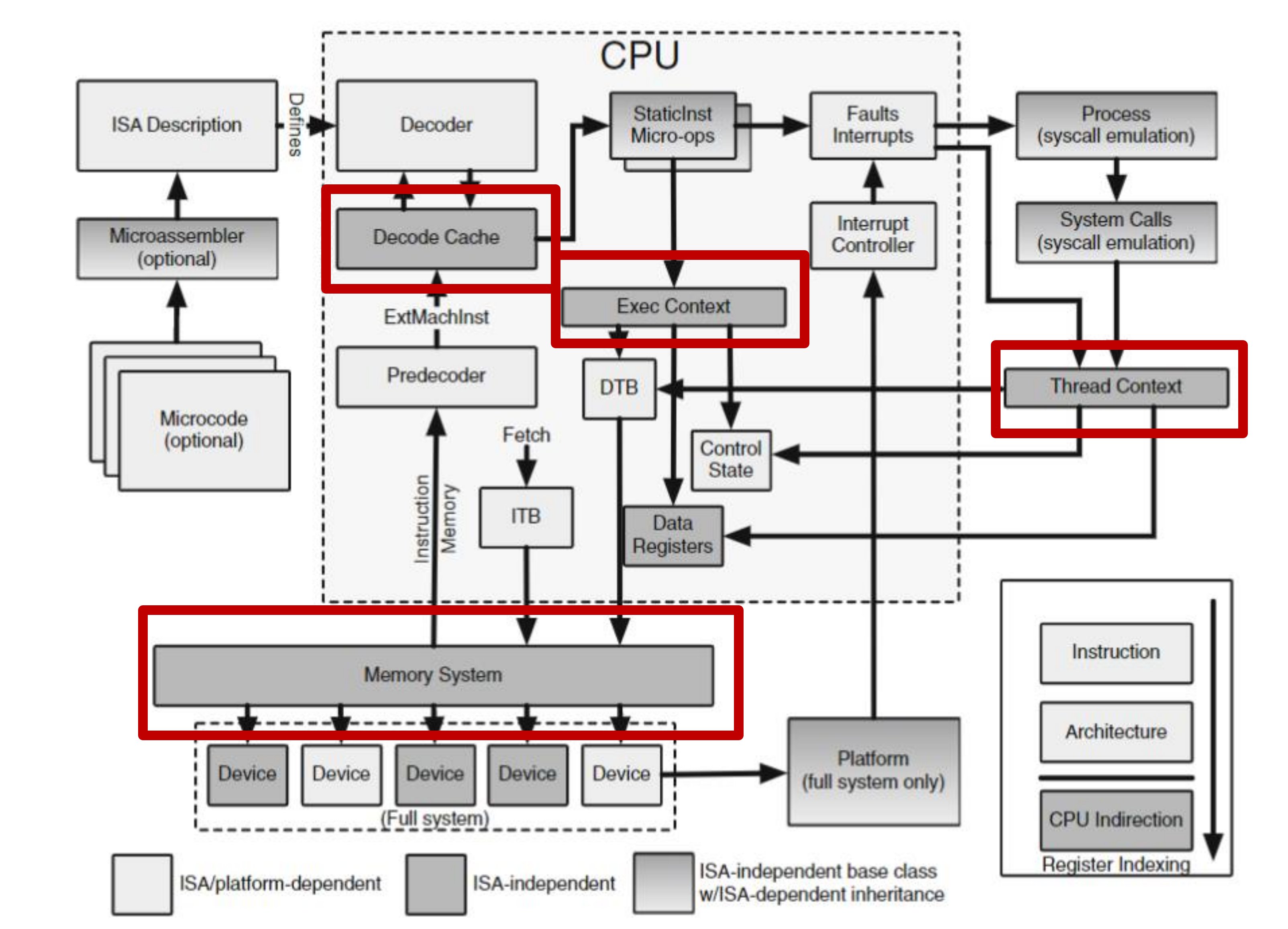
Overview of ISA Independence



ISA Dependent Components



ISA Independent Components



Outline

- What is an ISA
- ISA and CPU independence
- **High-level concepts to understand ISA implementation in gem5**
- Journey of instructions in gem5
- gem5 ISA parser
- Adding your own instructions in gem5
- Testing your instructions

High-level concepts

Execution of instructions and CPU models

StaticInst vs. DynInst

StaticInst (static information)

Op class

Source and dest. Registers

Static inst. -- binary inst. (1:1 mapping)

Flag to show if the inst. has uops

Virtual functions

- *execute()*
- *initiateAcc()*
- *completeAcc()*
- *disassemble()*

DynInst (dynamic information)

Instruction PC, predicted next-PC

Instruction result

Thread number

CPU

Renamed reg. indices

Provides the *ExecContext* interface

Execution Context

ExecContext provides methods to

- Read/write PC
- Read/write other registers
- Read/write memory
- Trigger full-system functionality
- Examples: SimpleCPU, DynInst

Thread Context

ThreadContext provides methods to

- Read/write PC
- Read/write other registers
- Access thread related classes like ITB, DTB, mem ports

Abstract class – CPU must create its own ThreadContext

Journey of an Instruction in gem5!

Let's use the same script from cpu-models session

```
> cp /workspaces/gem5-bootcamp-env/materials/using-gem5/05-cpu-models/finished-material/cpu-models-normal-cache.py .

> mv cpu-models.py inst_trace.py
```

Update the CustomResource path as below

```
tests/test-progs/hello/bin/riscv/linux/hello
```

Update the CPU model to TIMING

We will trace the execution of an Add and a LW instruction!

```
> gdb build/RISCV/gem5.opt
```

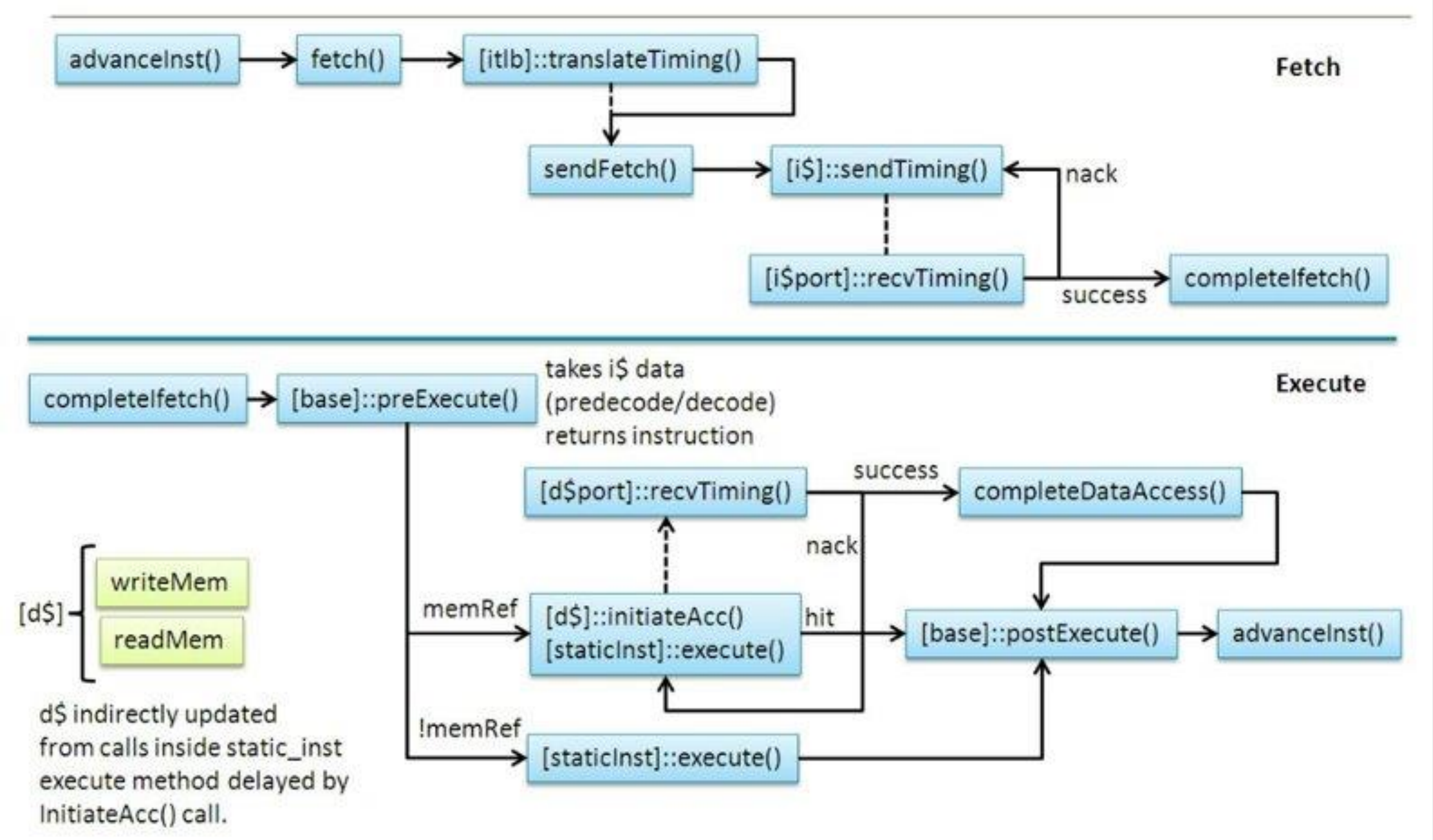
Inside gdb for an Add instruction

```
> b Add::Add
> b Add::execute
> run inst_trace.py
> bt 10
```

Inside gdb for a Lw instruction

```
> b Lw::Lw
> b Lw::initiateAcc
> b Lw::completeAcc
> run inst_trace.py
> bt 10
```

TimingSimpleCPU Reference



Let's understand how the decode and execute code is generated!

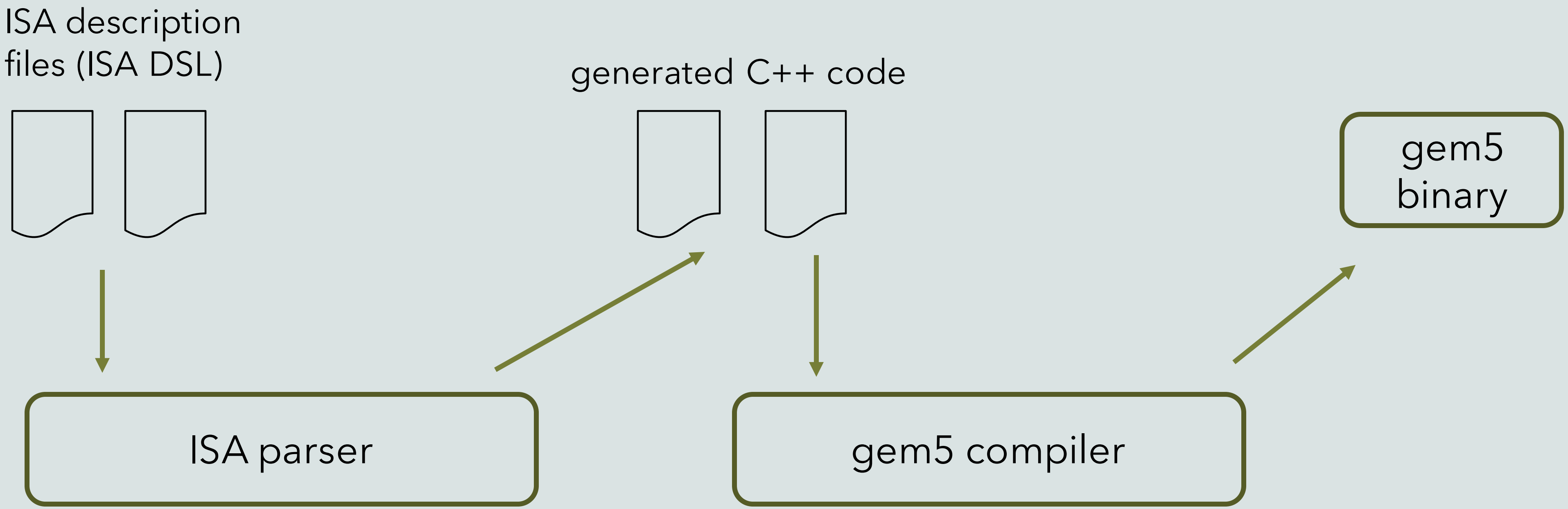
ISA Definition

- Description files contain decode and declaration sections
- src/arch/*/isa
- A domain-specific language for ISAs written in python (src/arch/isa_parser.py)
- Output in build/…/generated
- Decodes instructions (decoder/*.isa)
- Implements instructions (insts/*.isa)

ISA Parser

Written in a DSL compiled by a python script

High-level Flow of Instruction Definition


ISA description files (ISA DSL)

generated C++ code

gem5 binary

ISA parser

gem5 compiler

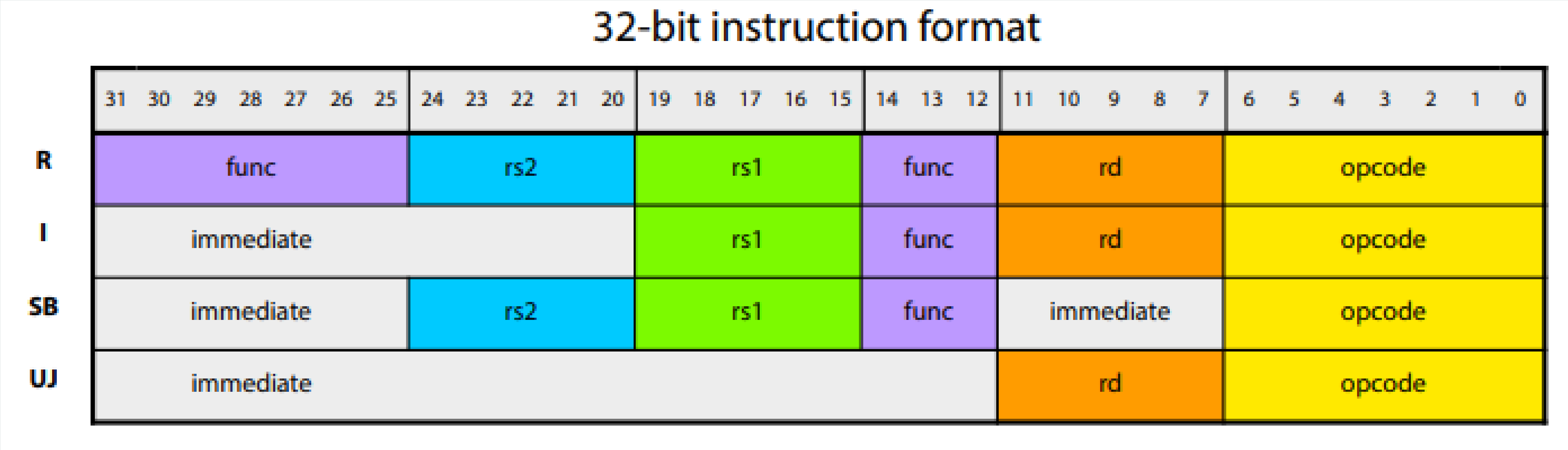
Deduces instruction's characteristics (from a brief description) using library of Python classes and functions

RISC-V Instruction Examples

Mnemonic	Instruction	Type	Description
ADD Rd, Rs1, Rs2	Add	R	Rd <-- Rs1 + Rs2
LW Rd, Imm2(Rs1)	Load word	I	Rd <-- mem[Rs1 + Imm2]

Instruction Encoding

ADD FUNC7 = 0x0 Src2 Src1 FUNC3=0 Dest. OPCODE=0xC QUAD=3



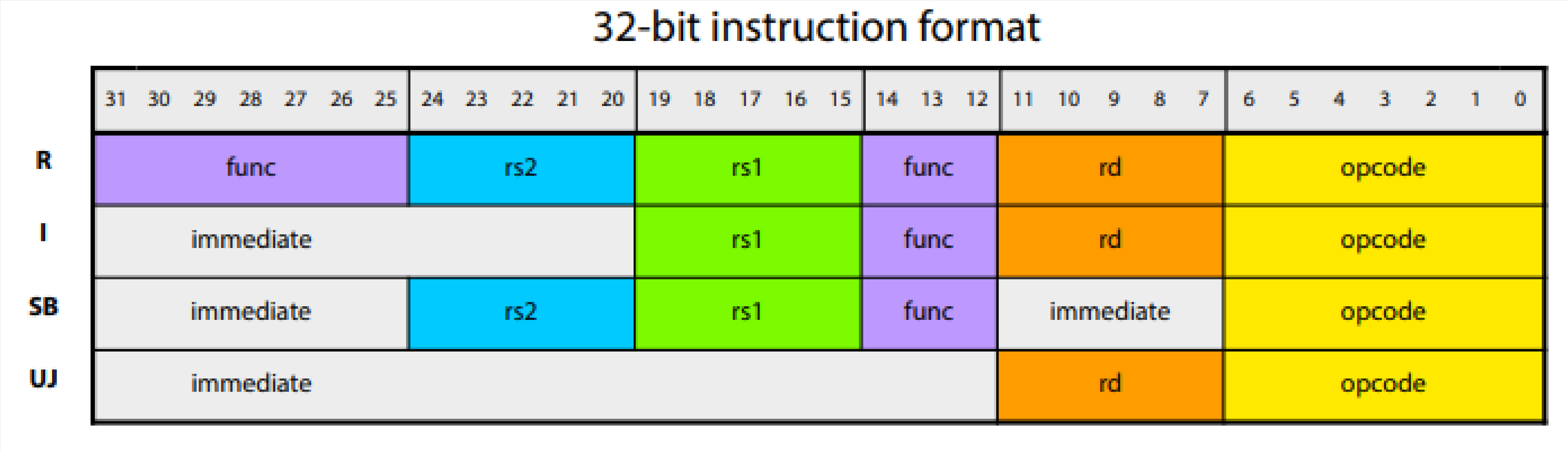
32-bit instruction format

	31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0
R	func	rs2	rs1	func	rd	opcode
I	immediate		rs1	func	rd	opcode
SB	immediate	rs2	rs1	func	immediate	opcode
UJ	immediate				rd	opcode

LW IMM12 Src1 FUNC3=2 Dest. OPCODE=0x0 QUAD=3

gem5 Instruction Decoding

ADD FUNC7 = 0x0 Src2 Src1 **FUNC3=0** Dest. **OPCODE=0xC** **QUAD=3**



32-bit instruction format

	31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0
R	func	rs2	rs1	func	rd	opcode
I	immediate		rs1	func	rd	opcode
SB	immediate	rs2	rs1	func	immediate	opcode
UJ	immediate				rd	opcode

LW IMM12 Src1 **FUNC3=2** Dest. **OPCODE=0x0** **QUAD=3**

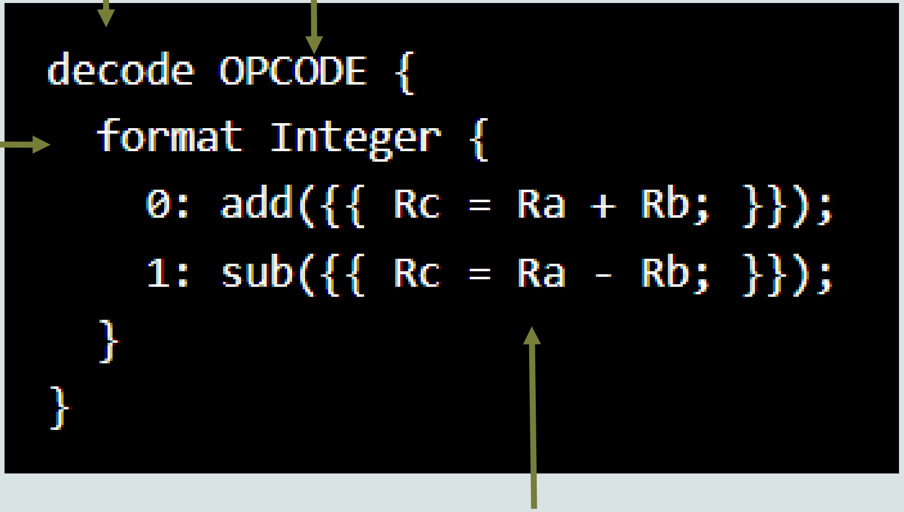
Decoding an Instruction

Decode Section

Like C switch statements
Decodes a field of the inst.

Bitfield definition (**declaration section**)

Format determines what function will be invoked

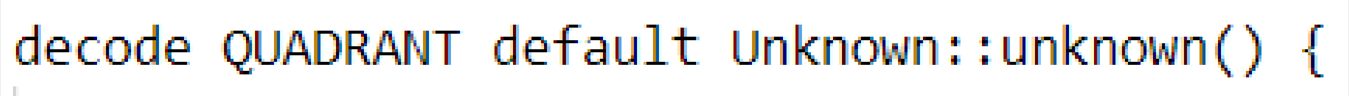


```
decode OPCODE {
  format Integer {
    0: add({{ Rc = Ra + Rb; }});
    1: sub({{ Rc = Ra - Rb; }});
  }
}
```

Code literal (similar to a string constant, delimited by double braces ({{ and }}))

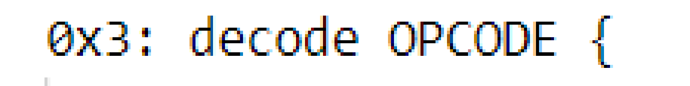
Decoding an ADD Instruction

A set of nested decode blocks!



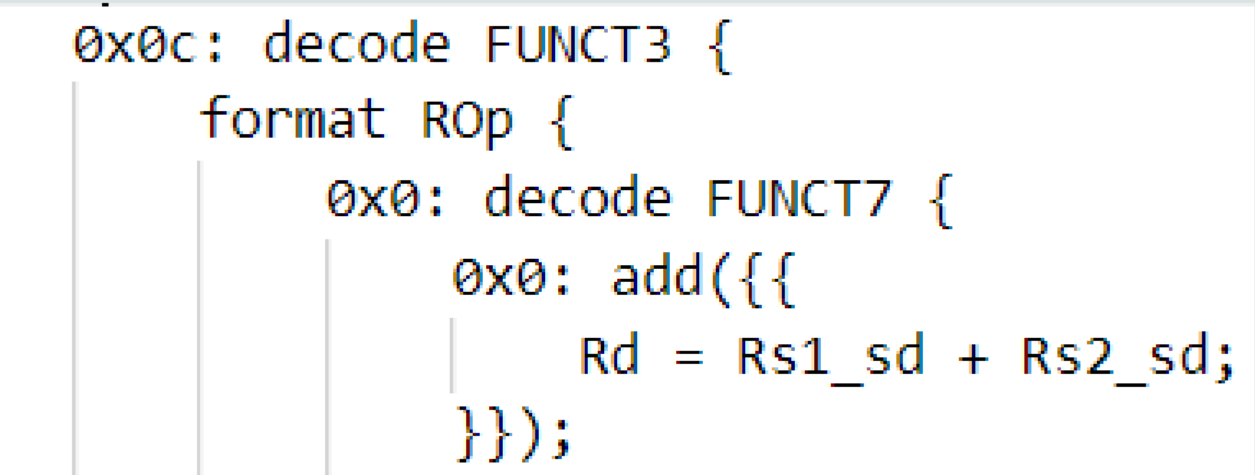
```
decode QUADRANT default Unknown::unknown() {
```

⋮



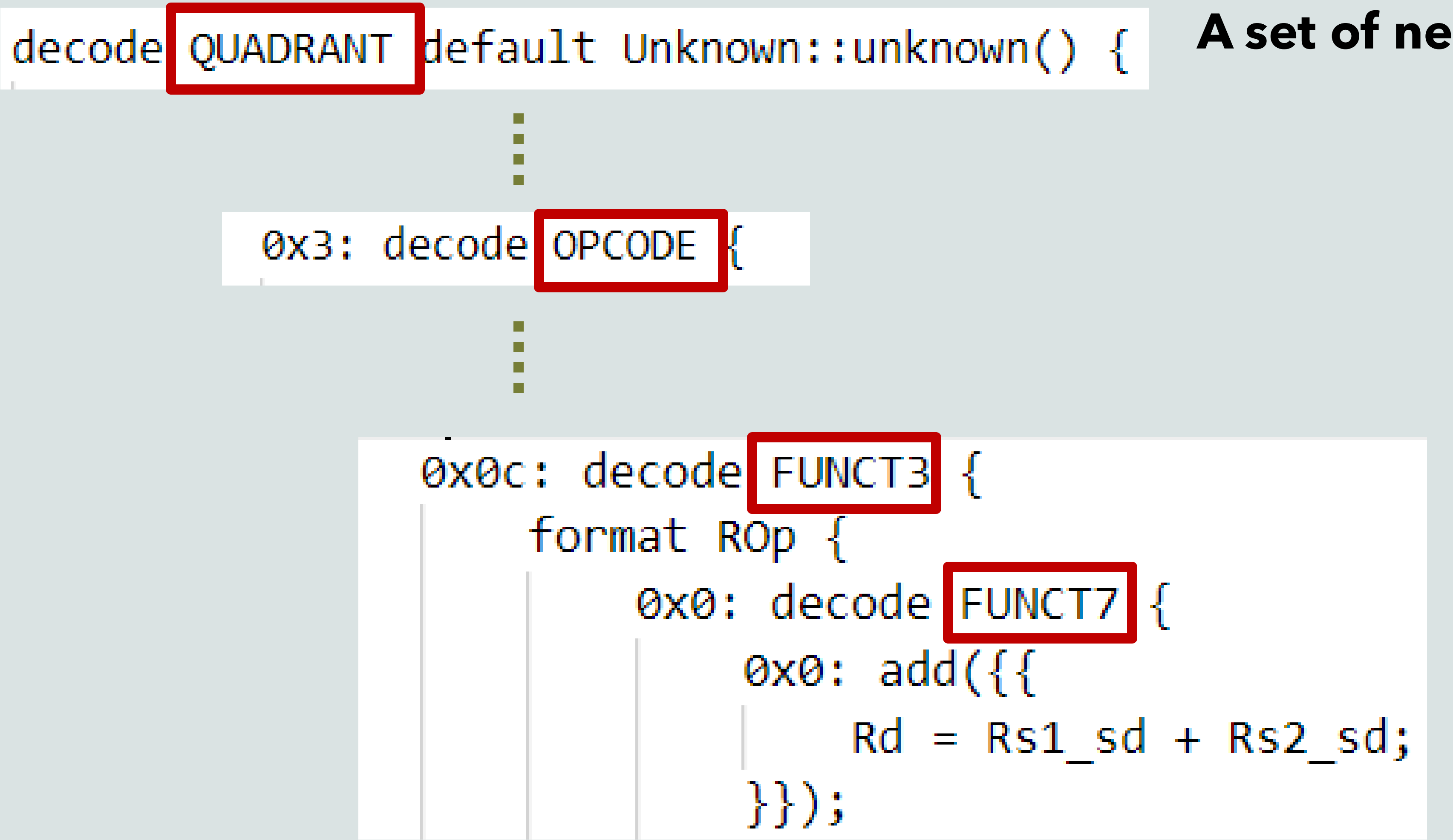
```
0x3: decode OPCODE {
```

⋮



```
0x0c: decode FUNCT3 {
    format ROp {
        0x0: decode FUNCT7 {
            0x0: add({{
                Rd = Rs1_sd + Rs2_sd;
            }});
```

Decoding an ADD Instruction



```
decode QUADRANT default Unknown::unknown() {
```

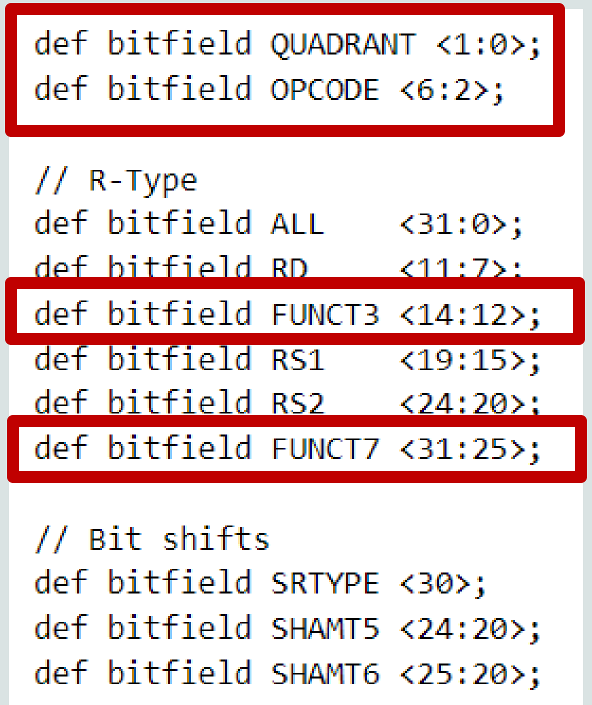
```
  0x3: decode OPCODE {
```

```
   0x0c: decode FUNCT3 {
       format ROp {
           0x0: decode FUNCT7 {
               0x0: add({{
                   Rd = Rs1_sd + Rs2_sd;
               }});
```

A set of nested decode blocks!

Bitfield definition

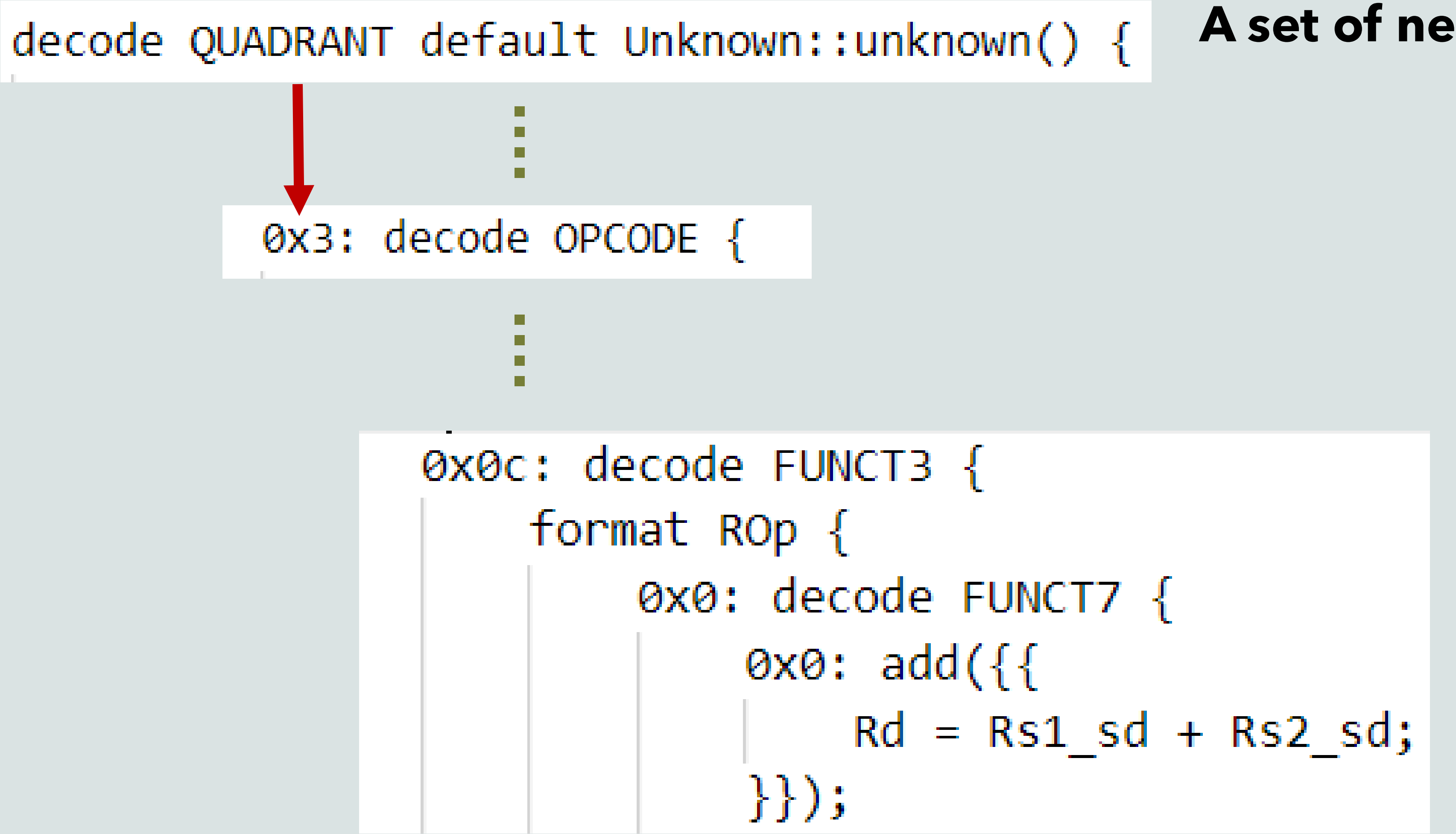
Name for a bitfield within a machine instruction



```
def bitfield QUADRANT <1:0>;
def bitfield OPCODE <6:2>;

// R-Type
def bitfield ALL    <31:0>;
def bitfield RD     <11:7>;
def bitfield FUNCT3 <14:12>;
def bitfield RS1    <19:15>;
def bitfield RS2    <24:20>;
def bitfield FUNCT7 <31:25>;

// Bit shifts
def bitfield SRTYPE <30>;
def bitfield SHAMT5 <24:20>;
def bitfield SHAMT6 <25:20>;
```

src/arch/riscv/isa/bitfields.isa

Decoding an ADD Instruction



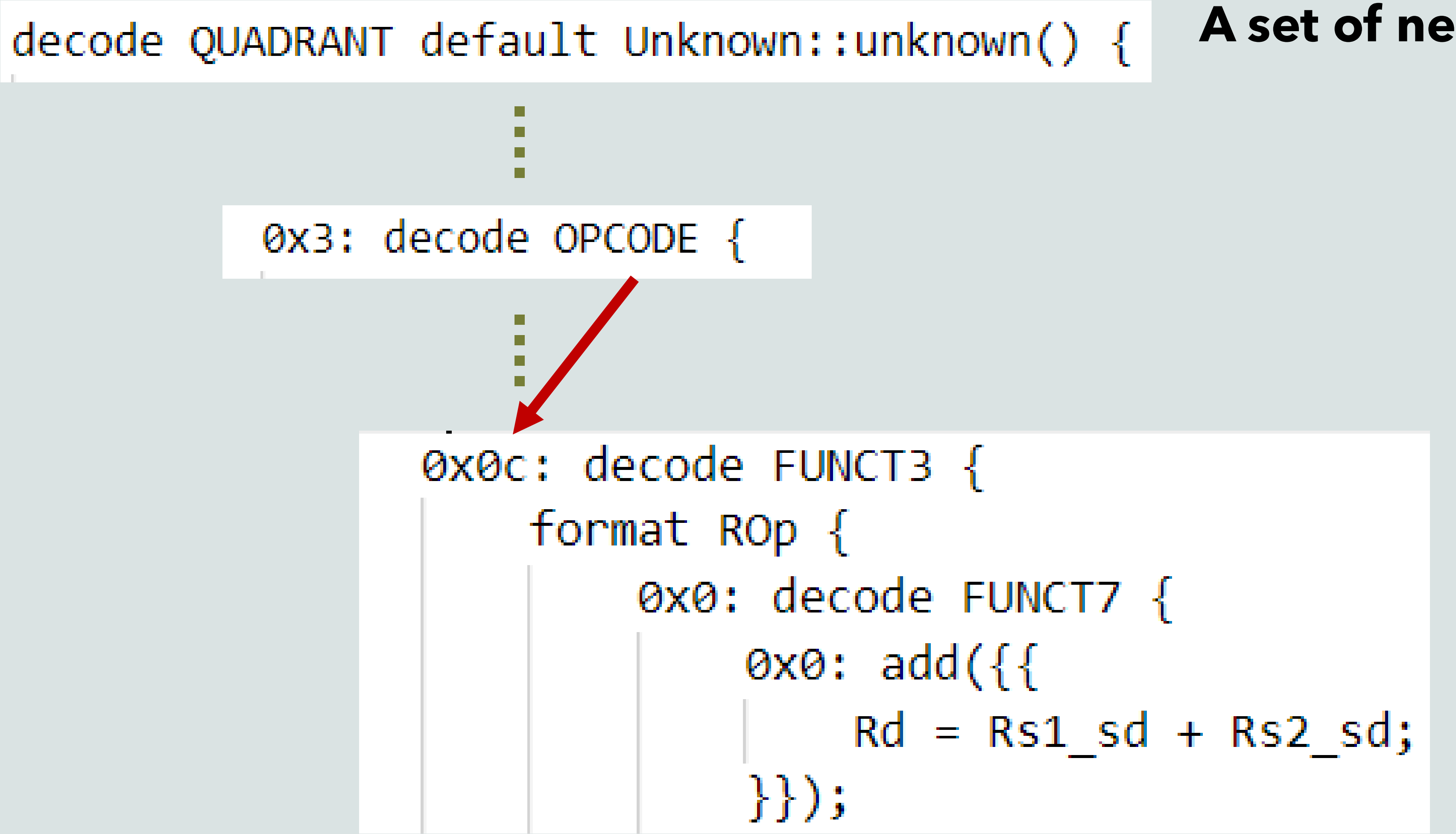
```
decode QUADRANT default Unknown::unknown() {
```

```
0x3: decode OPCODE {
```

```
0x0c: decode FUNCT3 {
    format ROp {
        0x0: decode FUNCT7 {
            0x0: add({{
                Rd = Rs1_sd + Rs2_sd;
            }});
```

A set of nested decode blocks!

Decoding an ADD Instruction



```
decode QUADRANT default Unknown::unknown() {
```

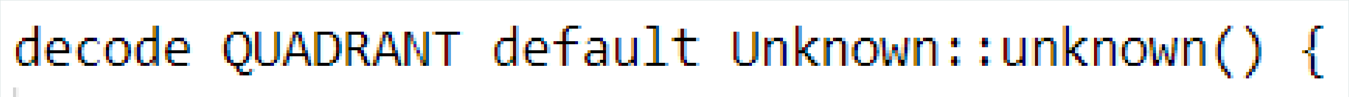
```
  0x3: decode OPCODE {
```

```
  0x0c: decode FUNCT3 {
      format ROp {
          0x0: decode FUNCT7 {
              0x0: add({{
                  Rd = Rs1_sd + Rs2_sd;
              }});
```

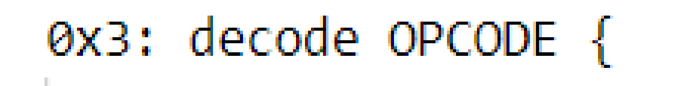
A set of nested decode blocks!

Decoding an ADD Instruction

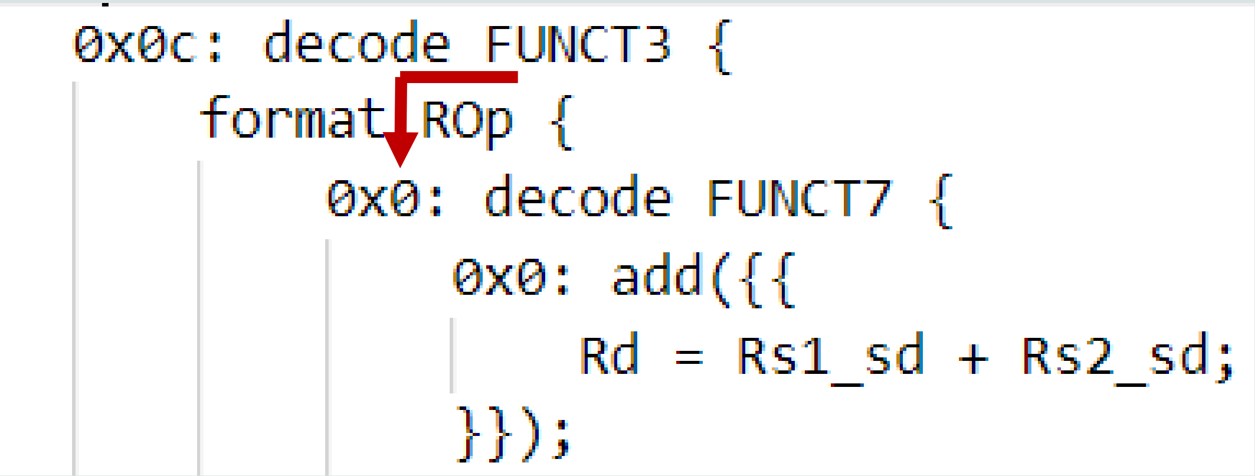
A set of nested decode blocks!



```
decode QUADRANT default Unknown::unknown() {
```



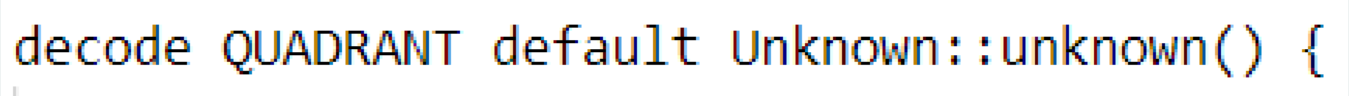
```
0x3: decode OPCODE {
```



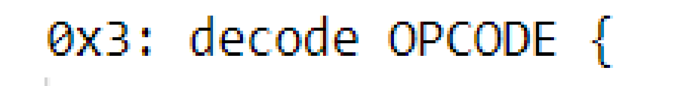
```
0x0c: decode FUNCT3 {
    format ROp {
        0x0: decode FUNCT7 {
            0x0: add({{
                Rd = Rs1_sd + Rs2_sd;
            }});
```

Decoding an ADD Instruction

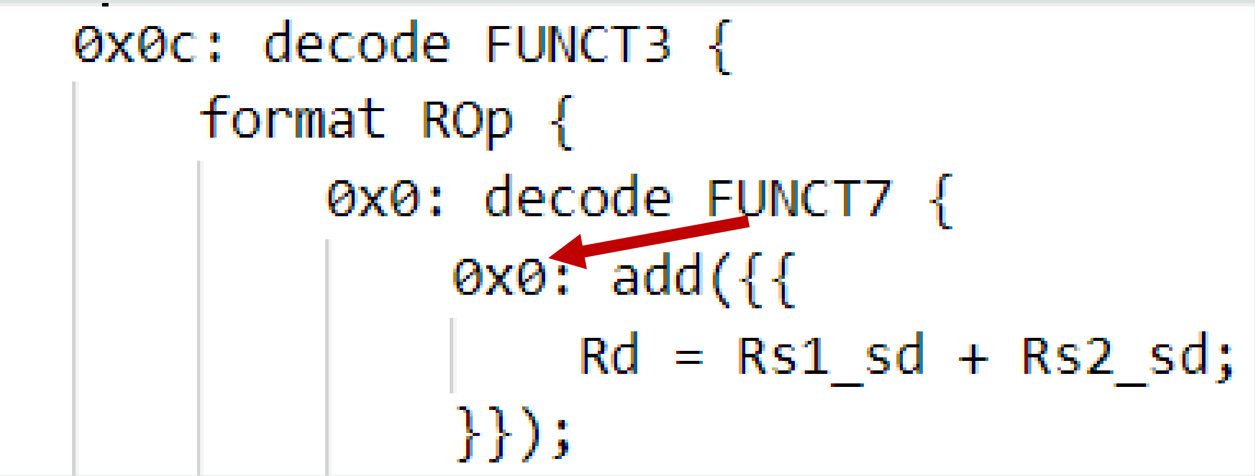
A set of nested decode blocks!



```
decode QUADRANT default Unknown::unknown() {
```

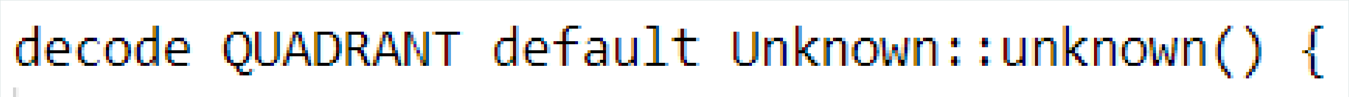


```
  0x3: decode OPCODE {
```



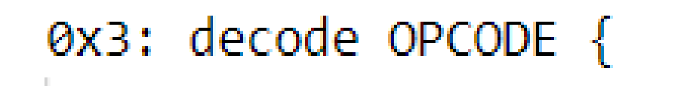
```
0x0c: decode FUNCT3 {
    format ROp {
        0x0: decode FUNCT7 {
            0x0: add({{
                Rd = Rs1_sd + Rs2_sd;
            }});
```

Decoding an ADD Instruction

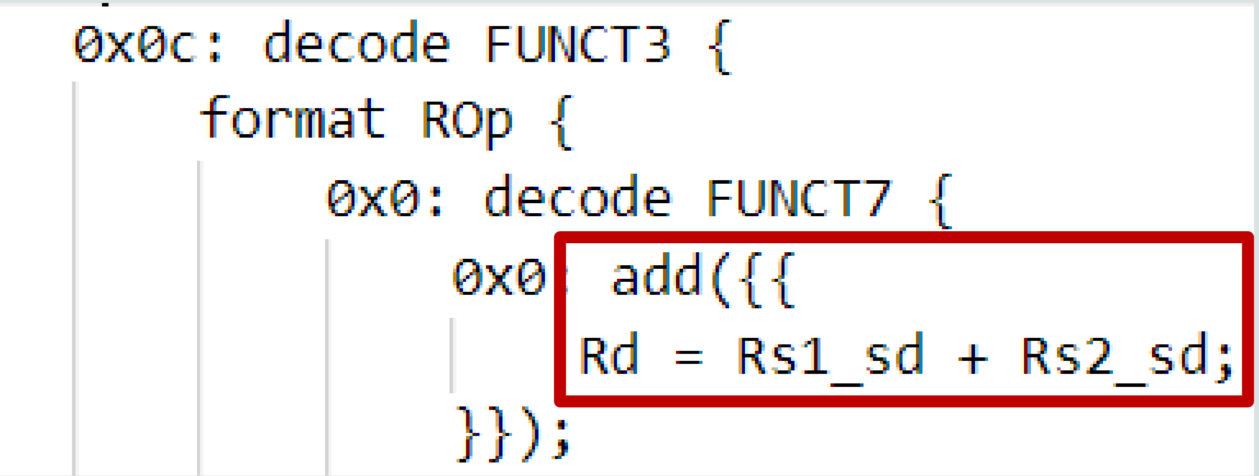


```
decode QUADRANT default Unknown::unknown() {
```

A set of nested decode blocks!

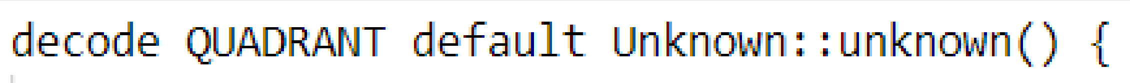


```
  0x3: decode OPCODE {
```

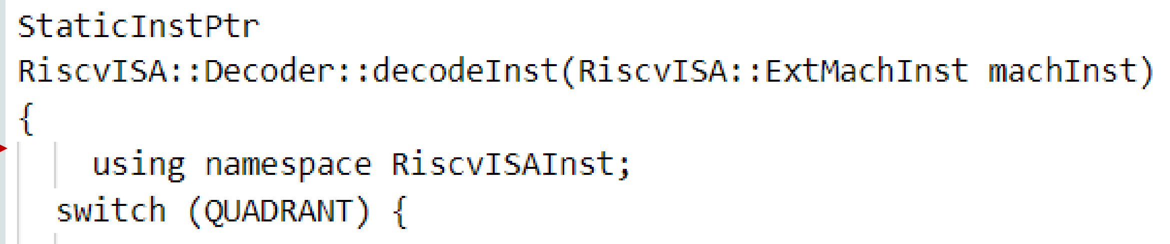


```
0x0c: decode FUNCT3 {
    format ROp {
        0x0: decode FUNCT7 {
            0x0: add({{
                Rd = Rs1_sd + Rs2_sd;
            }});
```

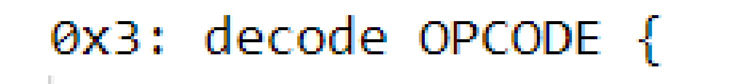
ISA DSL ---> Generated decode code (decode-method.cc.inc)



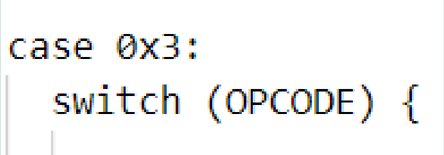
```
decode QUADRANT default Unknown::unknown() {
```



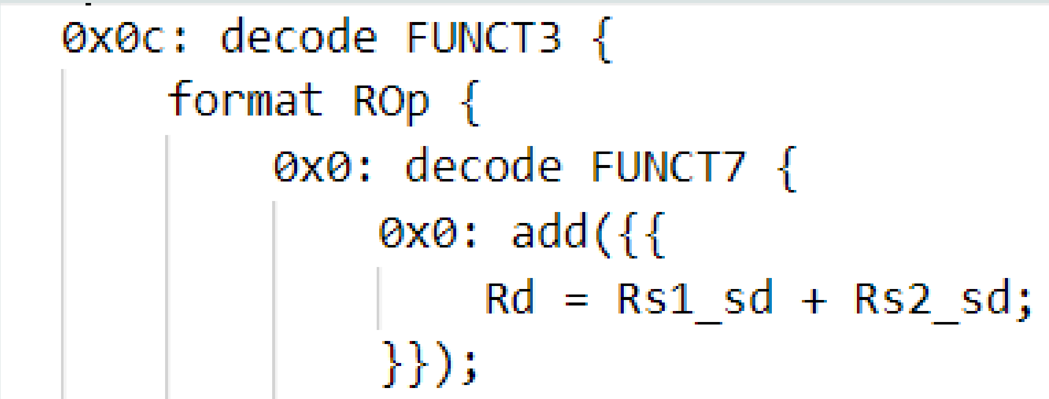
```
StaticInstPtr
RiscvISA::Decoder::decodeInst(RiscvISA::ExtMachInst machInst)
{
    using namespace RiscvISAInst;
  switch (QUADRANT) {
```



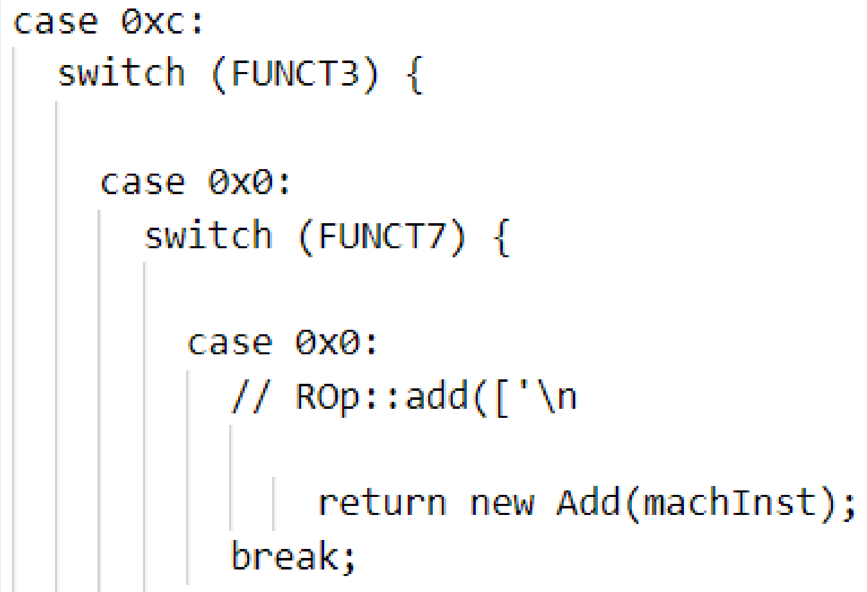
```
  0x3: decode OPCODE {
```



```
case 0x3:
  switch (OPCODE) {
```

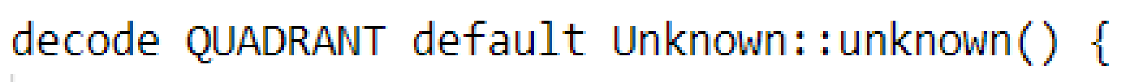


```
    0x0c: decode FUNCT3 {
          format ROp {
                0x0: decode FUNCT7 {
                      0x0: add({{
                            Rd = Rs1_sd + Rs2_sd;
                      }});
```

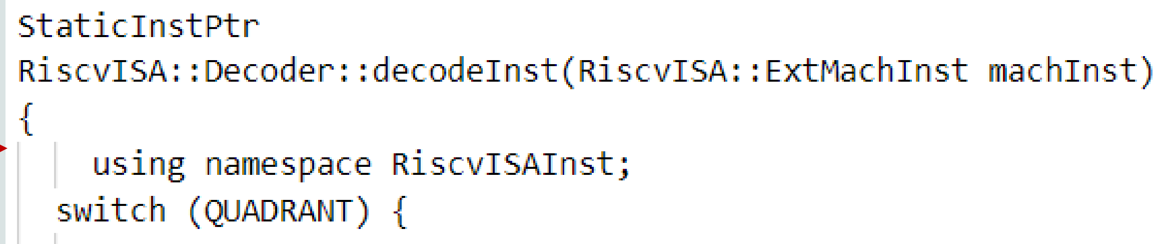


```
case 0xc:
  switch (FUNCT3) {

    case 0x0:
      switch (FUNCT7) {

        case 0x0:
          // ROp::add(['\n

            return new Add(machInst);
          break;
```

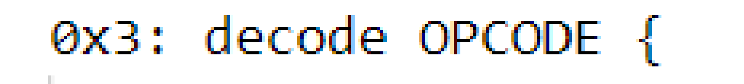
ISA DSL ---> Generated decode code (decode-method.cc.inc)



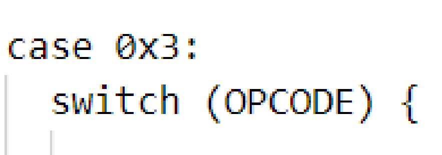
```
decode QUADRANT default Unknown::unknown() {
```



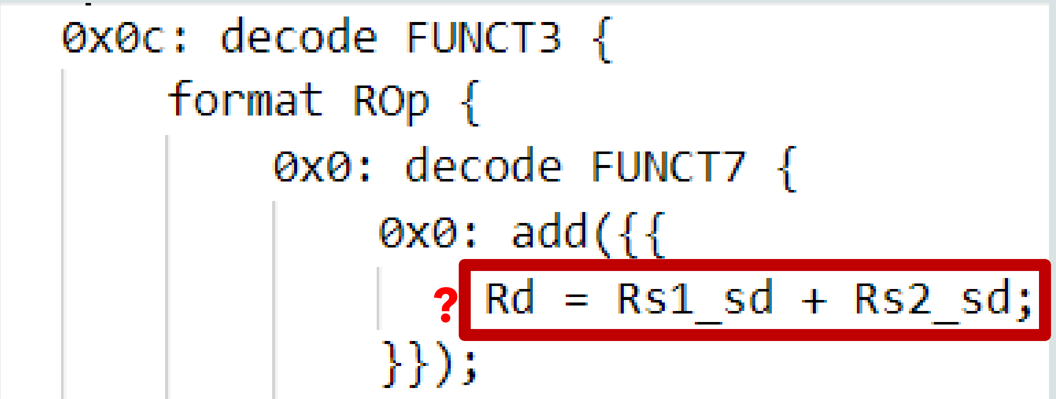
```
StaticInstPtr
RiscvISA::Decoder::decodeInst(RiscvISA::ExtMachInst machInst)
{
    using namespace RiscvISAInst;
  switch (QUADRANT) {
```



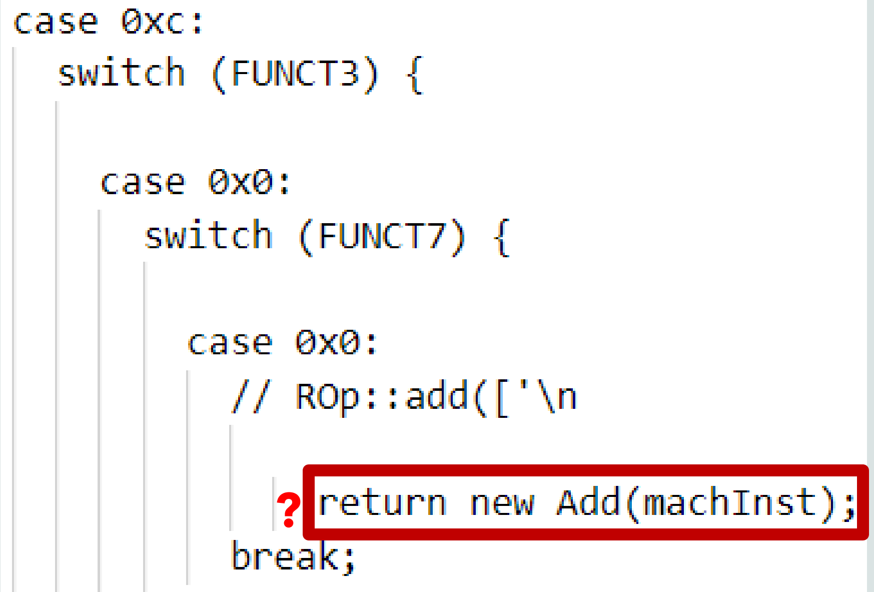
```
  0x3: decode OPCODE {
```



```
case 0x3:
  switch (OPCODE) {
```

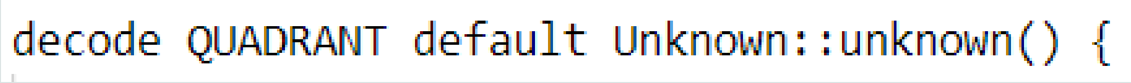


```
    0x0c: decode FUNCT3 {
          format ROp {
              0x0: decode FUNCT7 {
                  0x0: add({{
                   ? Rd = Rs1_sd + Rs2_sd;
                  }});
```

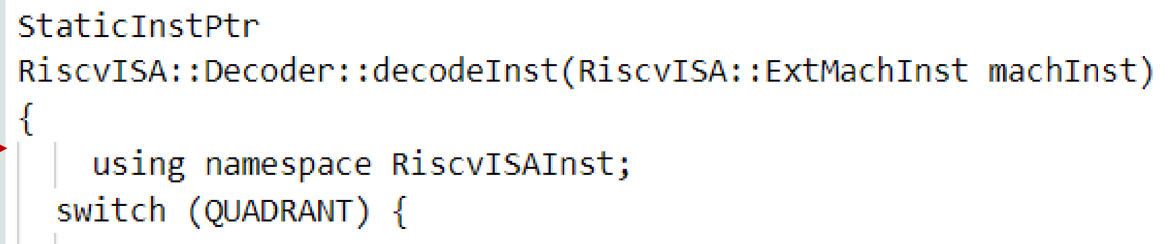


```
case 0xc:
  switch (FUNCT3) {

    case 0x0:
      switch (FUNCT7) {

        case 0x0:
          // ROp::add(['\n

          ? return new Add(machInst);
          break;
```

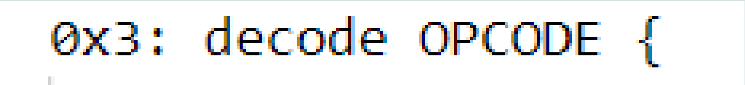
ISA DSL ---> Generated decode code (decode-method.cc.inc)



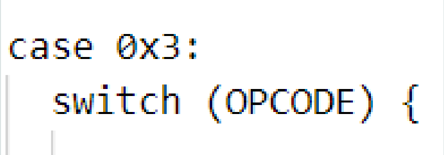
```
decode QUADRANT default Unknown::unknown() {
```



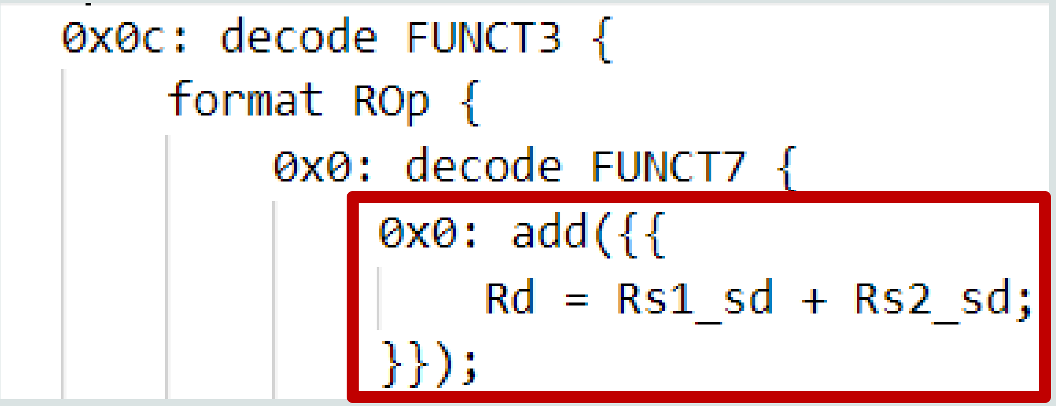
```
StaticInstPtr
RiscvISA::Decoder::decodeInst(RiscvISA::ExtMachInst machInst)
{
    using namespace RiscvISAInst;
  switch (QUADRANT) {
```



```
 0x3: decode OPCODE {
```

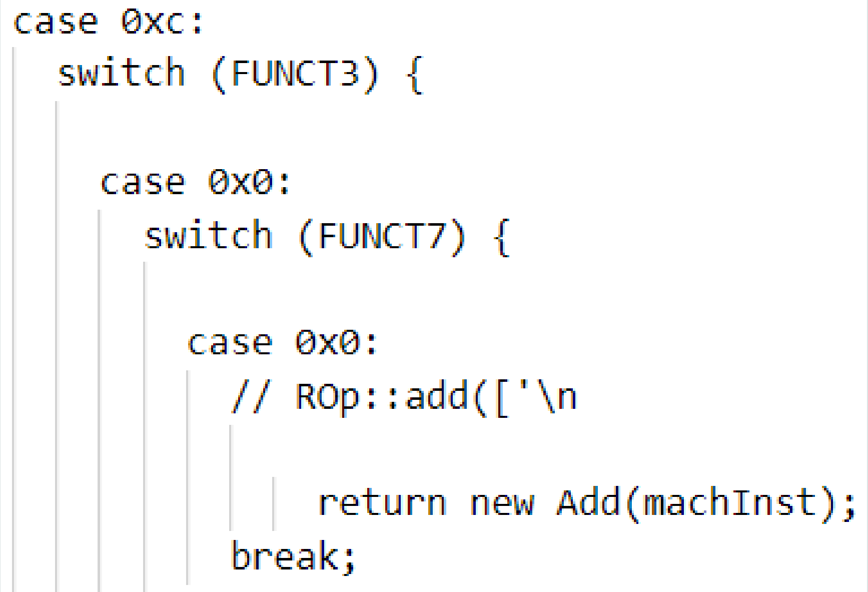


```
case 0x3:
  switch (OPCODE) {
```



```
  0x0c: decode FUNCT3 {
      format ROp {
          0x0: decode FUNCT7 {
              0x0: add({{
                  Rd = Rs1_sd + Rs2_sd;
              }});
```

Each instruction definition invokes a function call



```
case 0xc:
  switch (FUNCT3) {

    case 0x0:
      switch (FUNCT7) {

        case 0x0:
          // ROp::add(['\n

            return new Add(machInst);
          break;
```

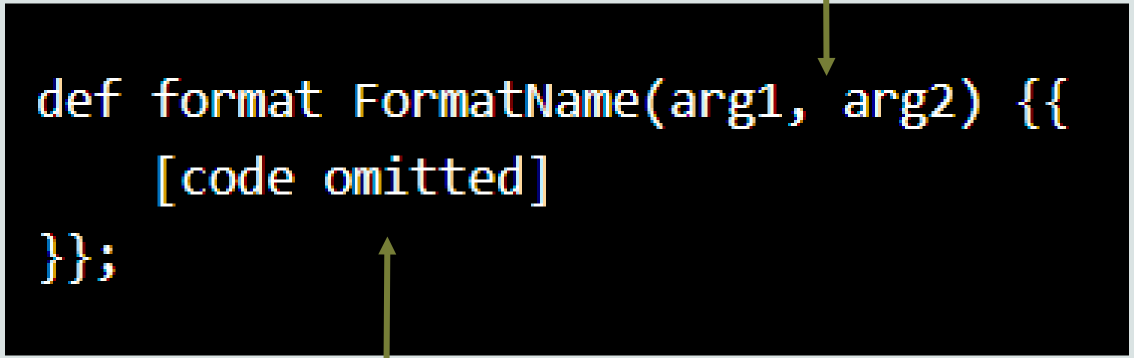
Declaration Section

- Instruction formats, supporting elements for decode block

Declaration Section

- Instruction formats, supporting elements for decode block
- **Format definition**

from instruction definition in decode block

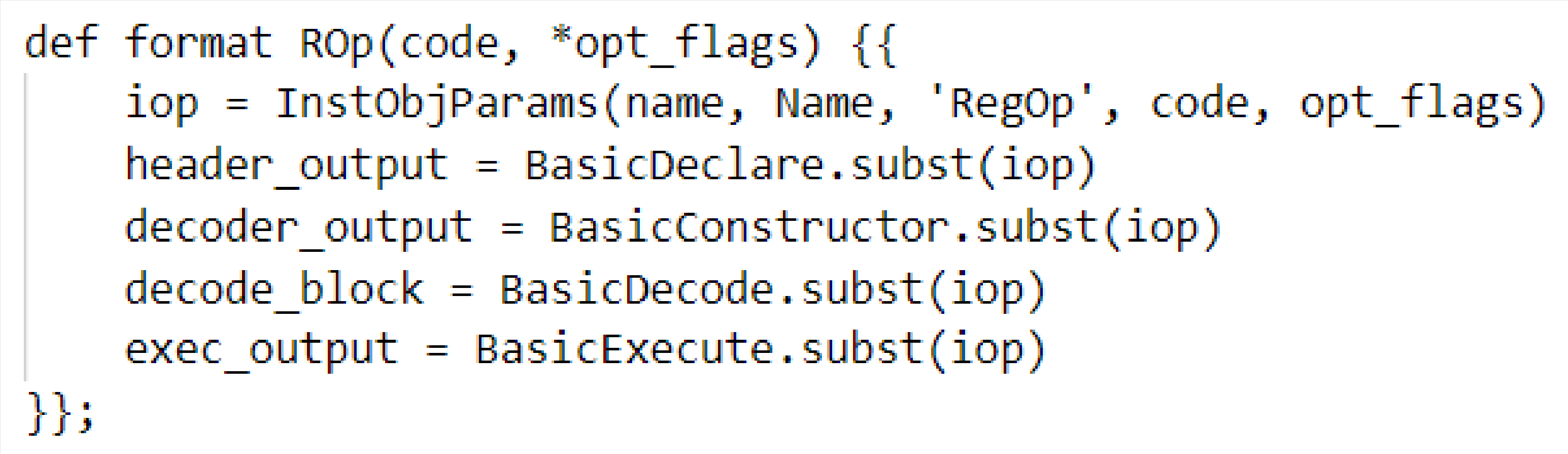


```
def format FormatName(arg1, arg2) {{
    [code omitted]
}};
```

Header output
Decoder output
Execute output
Decode block

Code literal (string constant, python code block)
Usually created by a specialized template!

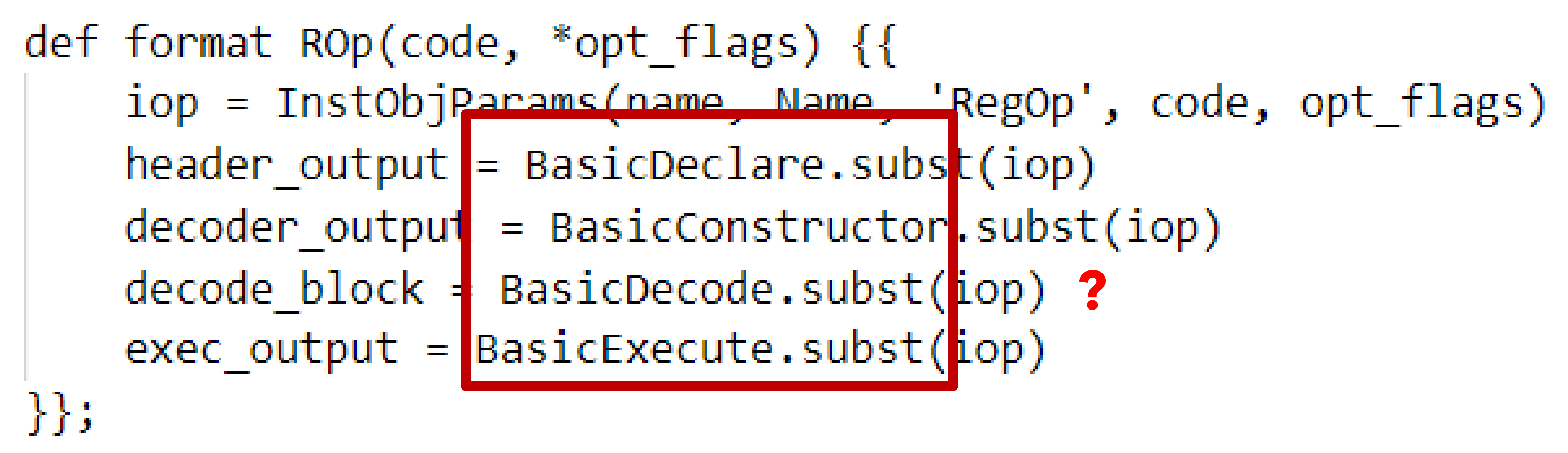
ROp Format used by ADD Instruction



```
def format ROp(code, *opt_flags) {{
    iop = InstObjParams(name, Name, 'RegOp', code, opt_flags)
    header_output = BasicDeclare.subst(iop)
    decoder_output = BasicConstructor.subst(iop)
    decode_block = BasicDecode.subst(iop)
    exec_output = BasicExecute.subst(iop)
}};
```

src/arch/riscv/isa/formats/standard.isa

ROp Format used by ADD Instruction

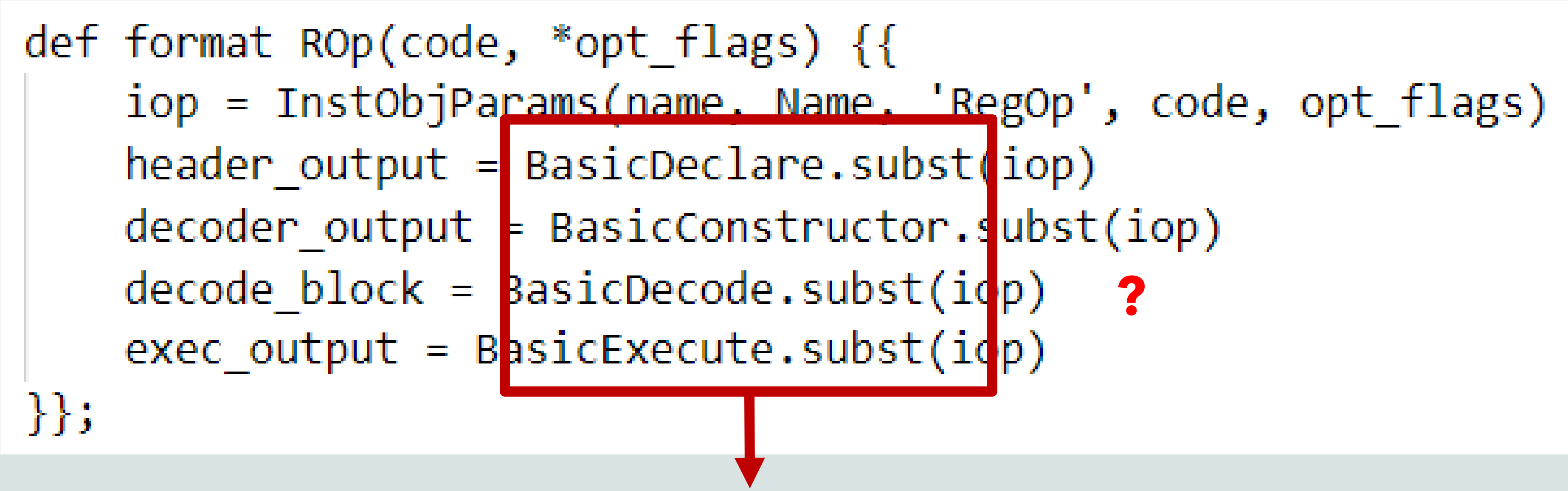


```
def format ROp(code, *opt_flags) {{
    iop = InstObjParams(name, Name, 'RegOp', code, opt_flags)
    header_output = BasicDeclare.subst(iop)
    decoder_output = BasicConstructor.subst(iop)
    decode_block = BasicDecode.subst(iop) ?
    exec_output = BasicExecute.subst(iop)
}};
```

src/arch/riscv/isa/formats/standard.isa

ROp Format used by ADD Instruction

String assignment to four special variables



```
def format ROp(code, *opt_flags) {{
    iop = InstObjParams(name, Name, 'RegOp', code, opt_flags)
    header_output = BasicDeclare.subst(iop)
    decoder_output = BasicConstructor.subst(iop)
    decode_block = BasicDecode.subst(iop)
    exec_output = BasicExecute.subst(iop)
}};
```

?

Template Blocks

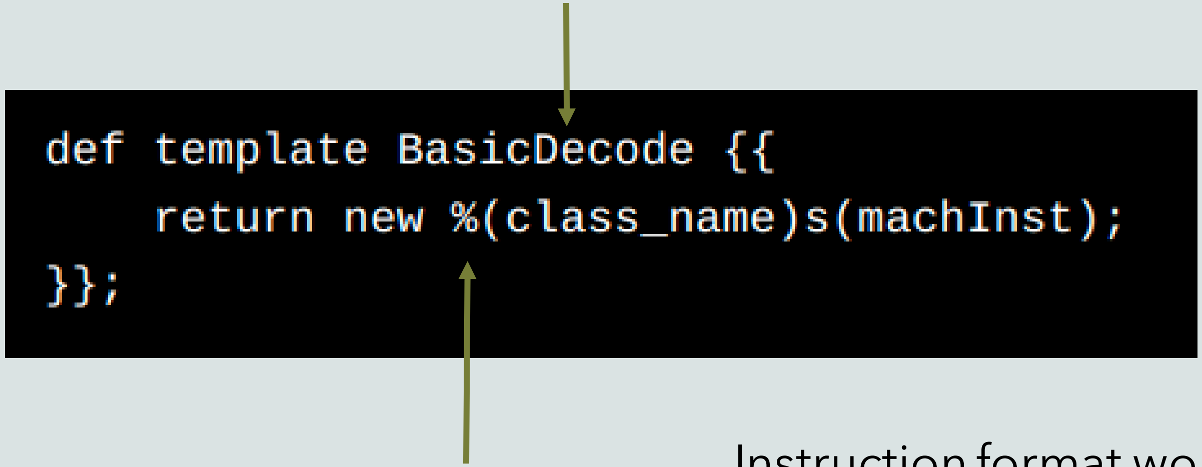
Template definitions

The code pieces of format block are generated through templates

BasicDecode Template

Template names start with a capital letter

Returning a string



```
def template BasicDecode {{
    return new %(class_name)s(machInst);
}};
```

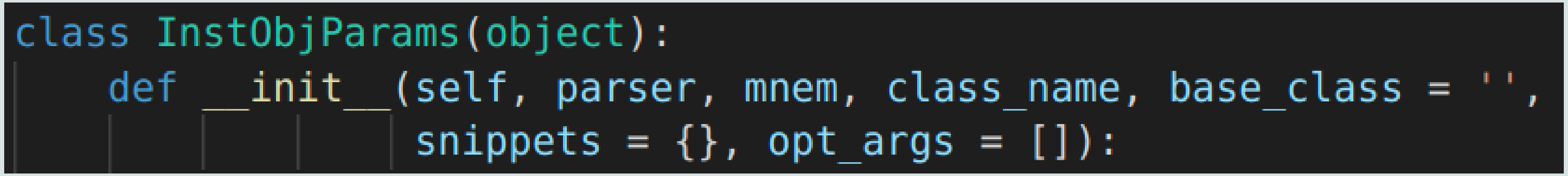
Code literal

Instruction format would specialize this template, using an instance of **InstObjParams**

InstObjParams Class

InstObjParams encapsulates the full set of parameters to be substituted into a template

isa_parser.py

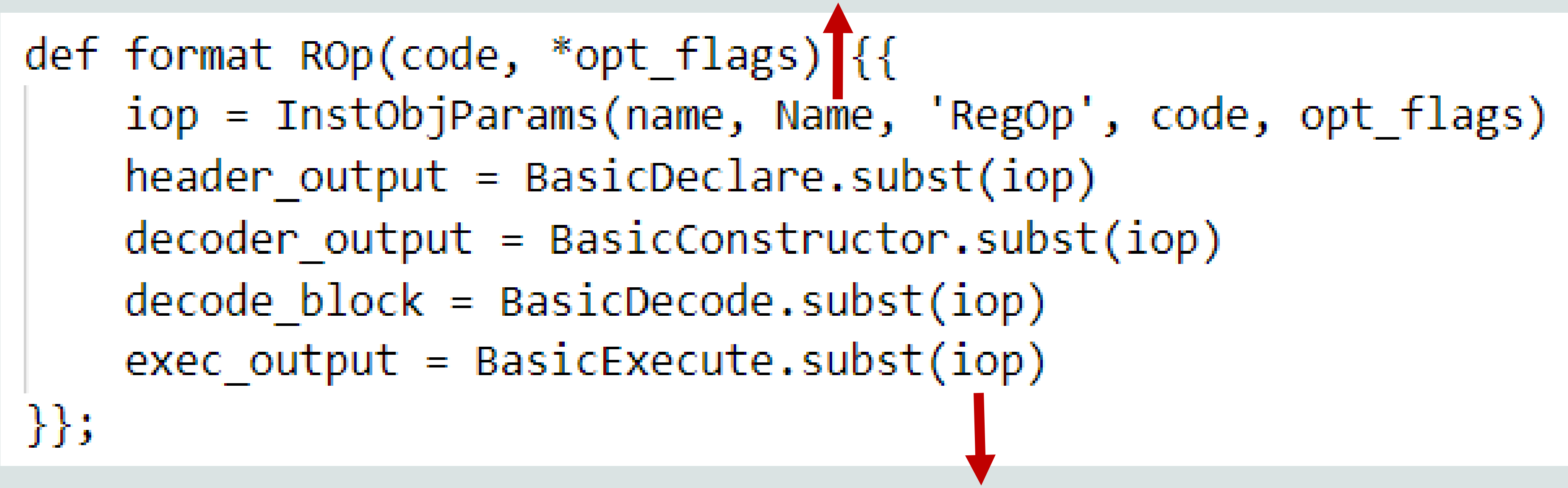


```python
class InstObjParams(object):
    def __init__(self, parser, mnem, class_name, base_class = '',
                 snippets = {}, opt_args = []):
```

Code block object

ROp Format used by ADD Instruction

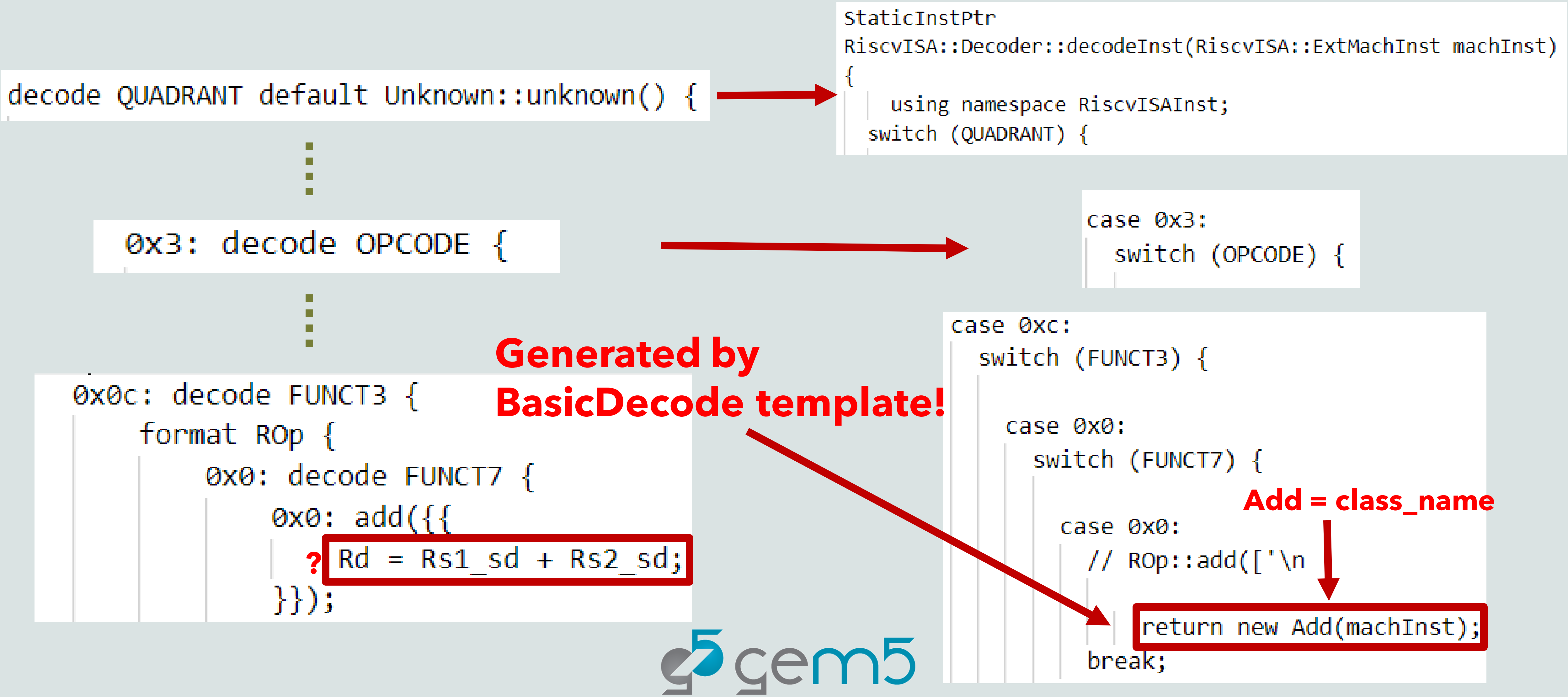
Instruction Mnemonic (class_name = Add)



```
def format ROp(code, *opt_flags) {{
    iop = InstObjParams(name, Name, 'RegOp', code, opt_flags)
    header_output = BasicDeclare.subst(iop)
    decoder_output = BasicConstructor.subst(iop)
    decode_block = BasicDecode.subst(iop)
    exec_output = BasicExecute.subst(iop)
}};
```

Object's attributes map to the substitution symbols in the template

ISA DSL ---> Generated decode code (decode-method.cc.inc)



```
decode QUADRANT default Unknown::unknown() {
```

```
StaticInstPtr
RiscvISA::Decoder::decodeInst(RiscvISA::ExtMachInst machInst)
{
    using namespace RiscvISAInst;
  switch (QUADRANT) {
```

```
  0x3: decode OPCODE {
```

```
case 0x3:
  switch (OPCODE) {
```

```
  0x0c: decode FUNCT3 {
      format ROp {
          0x0: decode FUNCT7 {
              0x0: add({{
               ? Rd = Rs1_sd + Rs2_sd;
              }});
```

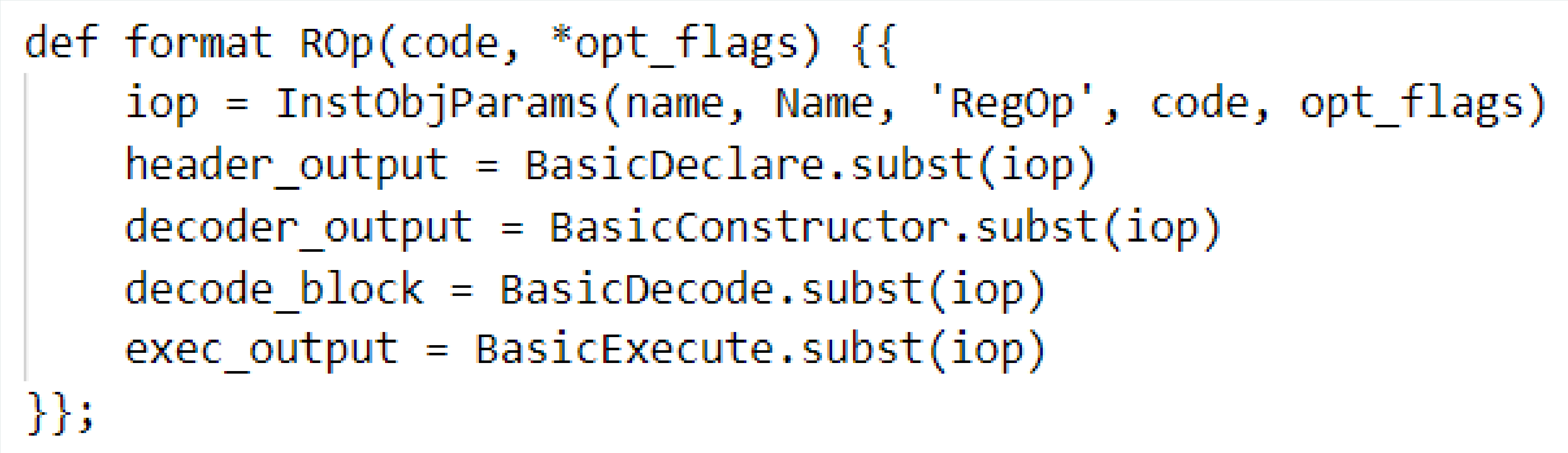
Generated by BasicDecode template!

```
case 0xc:
  switch (FUNCT3) {

    case 0x0:
      switch (FUNCT7) {

        case 0x0:
          // ROp::add(['\n
            return new Add(machInst);
          break;
```

Add = class_name

ROp Format used by ADD Instruction

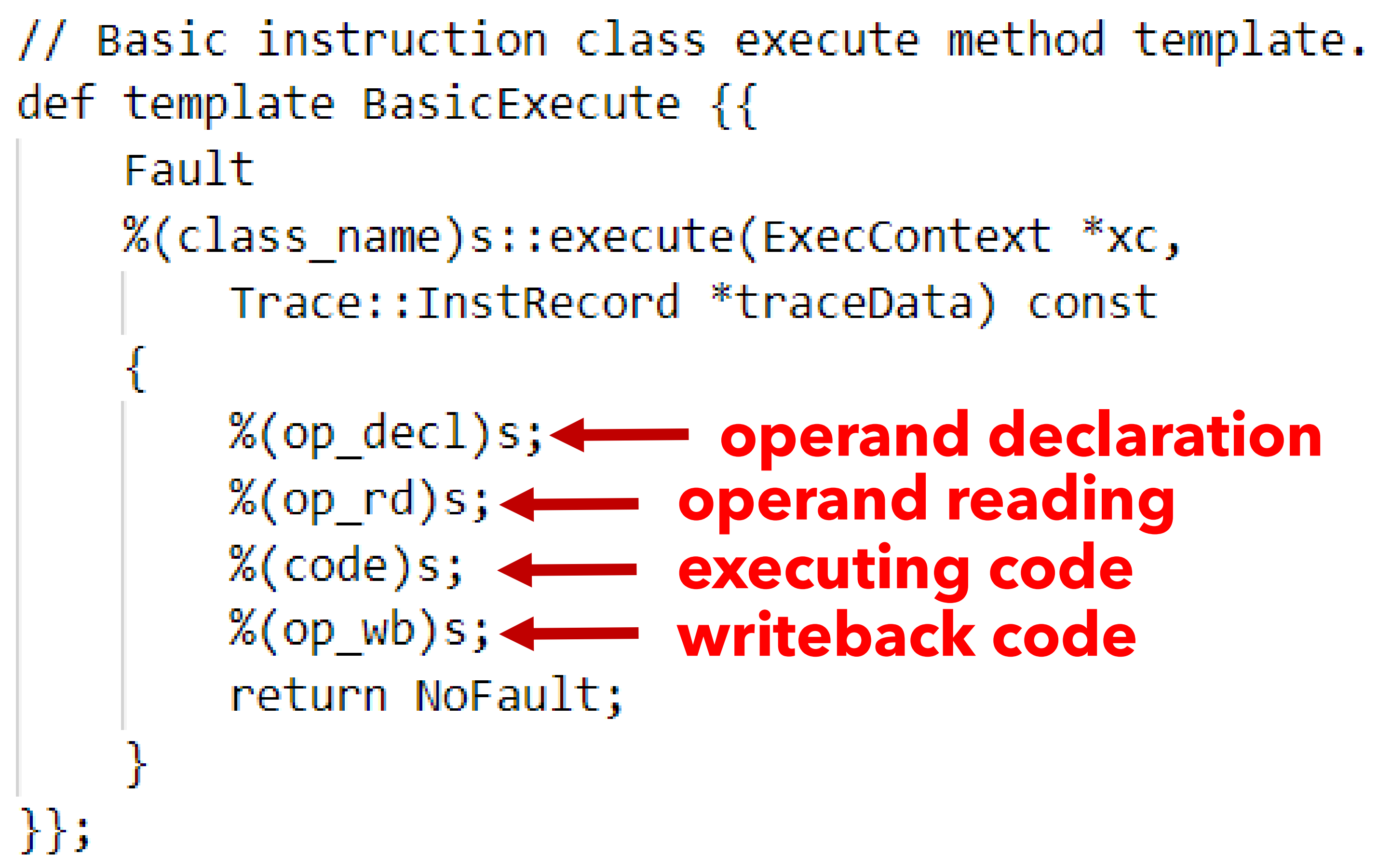
format also generates exec block!



```
def format ROp(code, *opt_flags) {{
    iop = InstObjParams(name, Name, 'RegOp', code, opt_flags)
    header_output = BasicDeclare.subst(iop)
    decoder_output = BasicConstructor.subst(iop)
    decode_block = BasicDecode.subst(iop)
    exec_output = BasicExecute.subst(iop)
}};
```

src/arch/riscv/isa/formats/standard.isa

BasicExecute Template



```
// Basic instruction class execute method template.
def template BasicExecute {{
    Fault
    %(class_name)s::execute(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        %(op_decl)s;
        %(op_rd)s;
        %(code)s;
        %(op_wb)s;
        return NoFault;
    }
}};
```

operand declaration

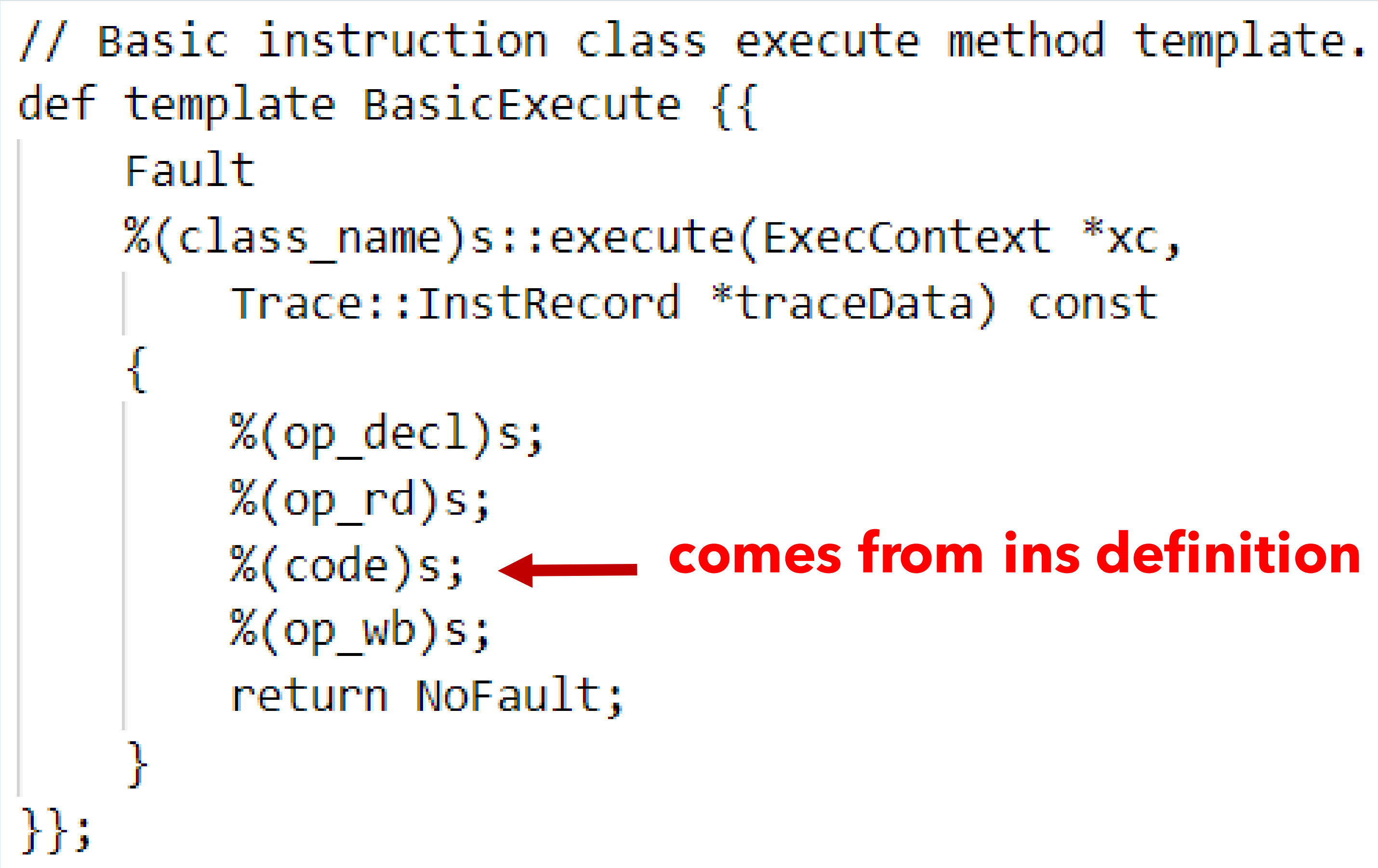
operand reading

executing code

writeback code

src/arch/riscv/isa/formats/basic.isa

BasicExecute Template

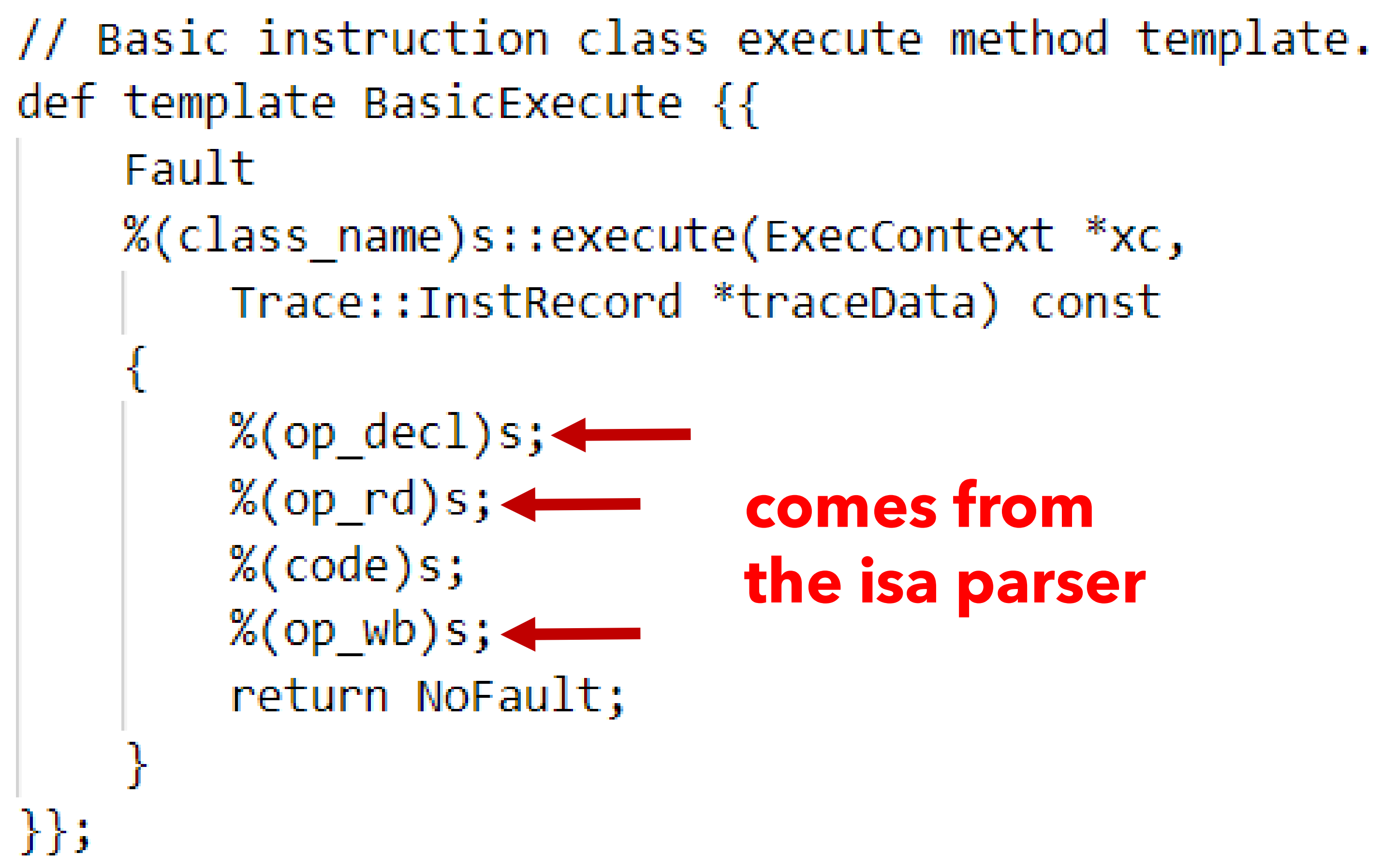


```
// Basic instruction class execute method template.
def template BasicExecute {{
    Fault
    %(class_name)s::execute(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        %(op_decl)s;
        %(op_rd)s;
        %(code)s;
        %(op_wb)s;
        return NoFault;
    }
}};
```

comes from ins definition

src/arch/riscv/isa/formats/basic.isa

BasicExecute Template

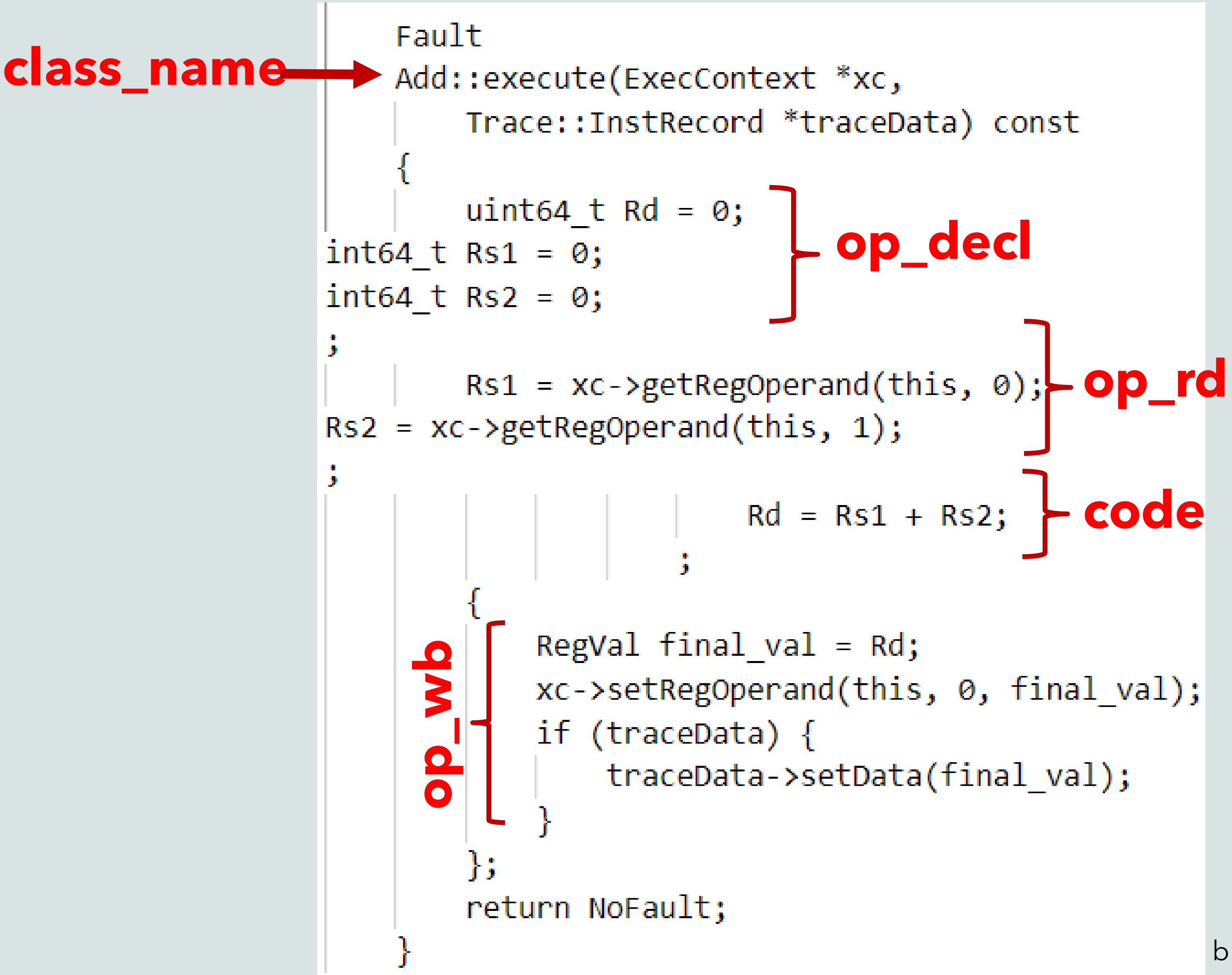


```
// Basic instruction class execute method template.
def template BasicExecute {{
    Fault
    %(class_name)s::execute(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        %(op_decl)s;
        %(op_rd)s;
        %(code)s;
        %(op_wb)s;
        return NoFault;
    }
}};
```

comes from the isa parser

src/arch/riscv/isa/formats/basic.isa

ISA DSL --> Generated Execute Code



```
    Fault
    Add::execute(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        uint64_t Rd = 0;
int64_t Rs1 = 0;
int64_t Rs2 = 0;
;
        Rs1 = xc->getRegOperand(this, 0);
Rs2 = xc->getRegOperand(this, 1);
;
                        Rd = Rs1 + Rs2;
                    ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

class_name

op_decl

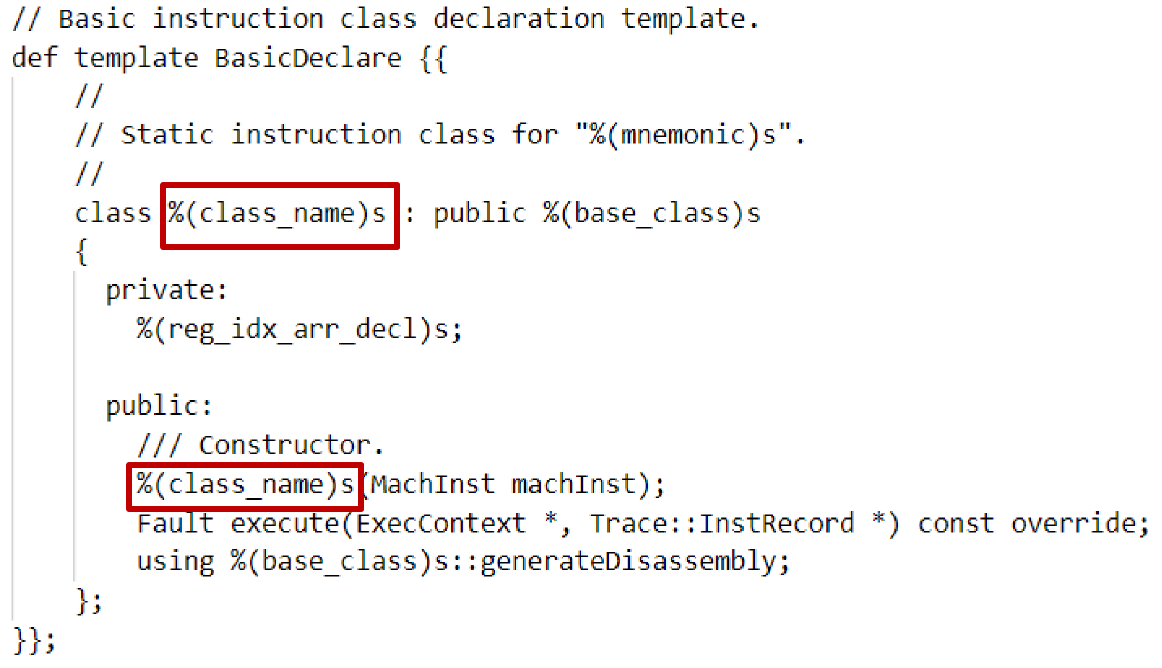
op_rd

code

op_wb

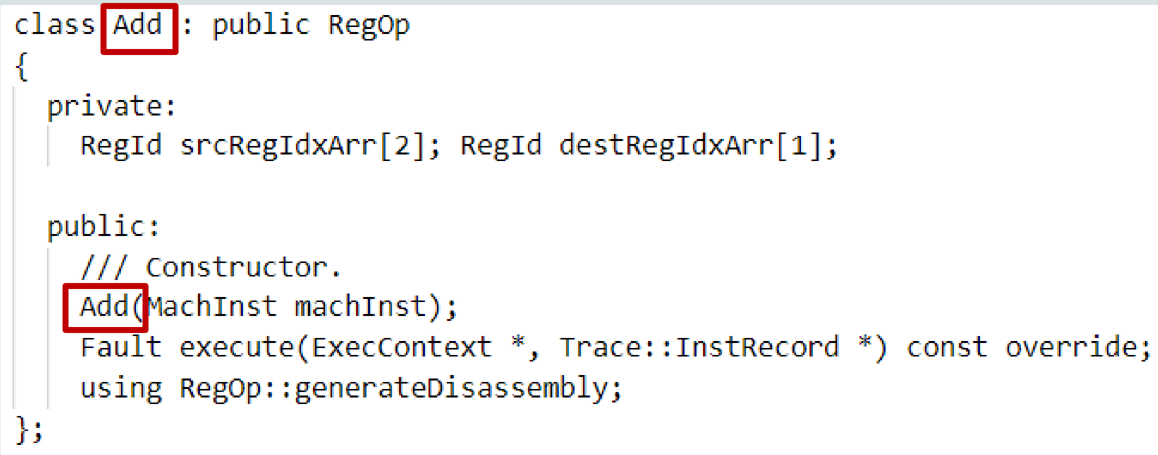
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

BasicDeclare Template



```
// Basic instruction class declaration template.
def template BasicDeclare {{
    //
    // Static instruction class for "%(mnemonic)s".
    //
    class %(class_name)s : public %(base_class)s
    {
      private:
        %(reg_idx_arr_decl)s;

      public:
        /// Constructor.
        %(class_name)s(MachInst machInst);
        Fault execute(ExecContext *, Trace::InstRecord *) const override;
        using %(base_class)s::generateDisassembly;
    };
}};
```

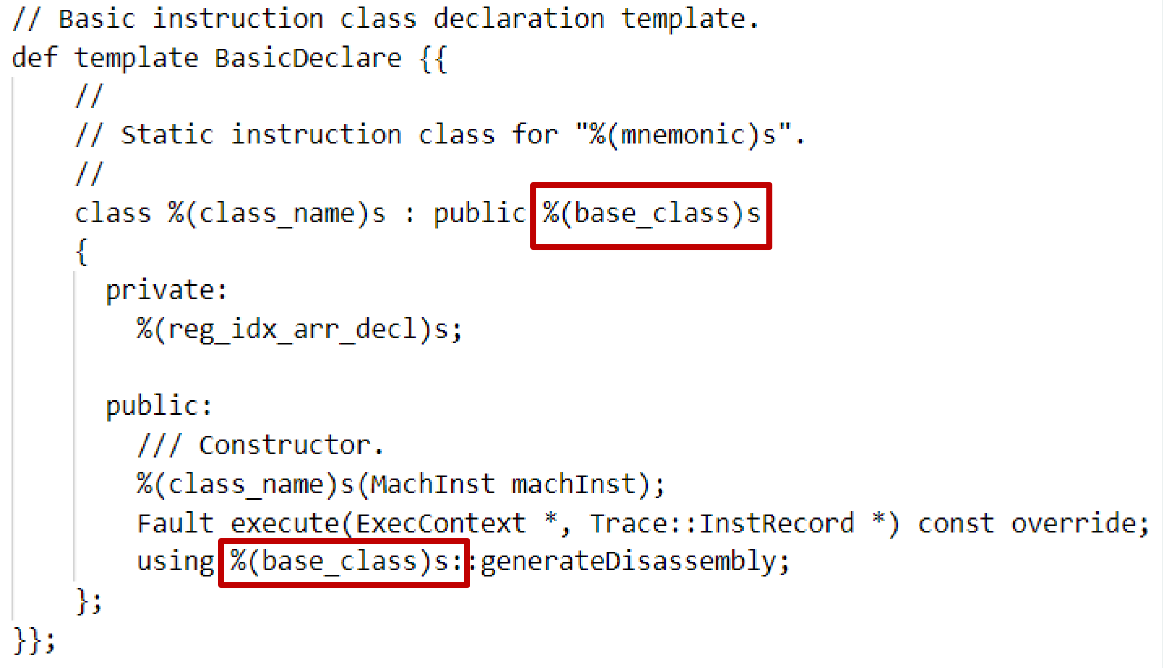
src/arch/riscv/isa/formats/basic.isa



```
class Add : public RegOp
{
  private:
    RegId srcRegIdxArr[2]; RegId destRegIdxArr[1];

  public:
    /// Constructor.
    Add(MachInst machInst);
    Fault execute(ExecContext *, Trace::InstRecord *) const override;
    using RegOp::generateDisassembly;
};
```

build/RISCV/arch/riscv/generated/decoder-ns.hh.inc

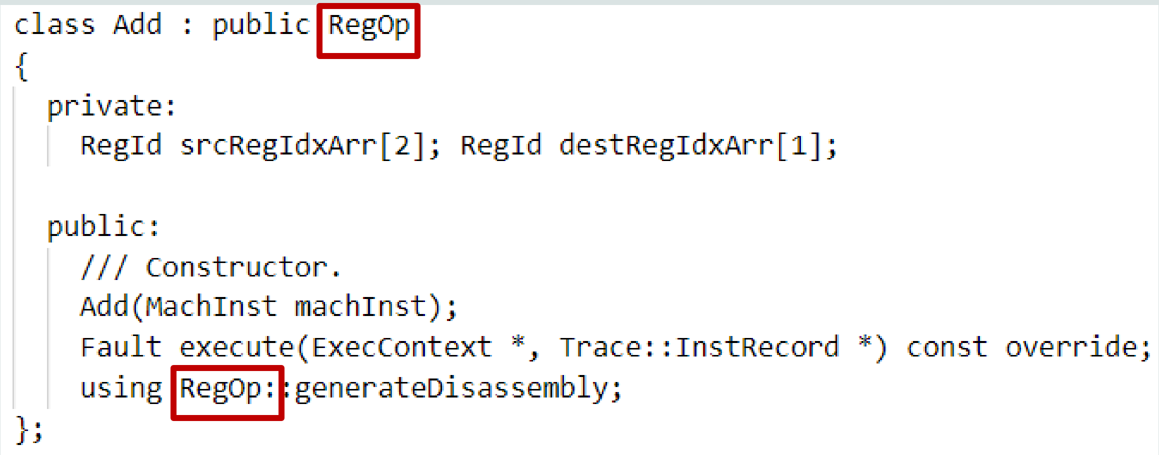
BasicDeclare Template



```
// Basic instruction class declaration template.
def template BasicDeclare {{
    //
    // Static instruction class for "%(mnemonic)s".
    //
    class %(class_name)s : public %(base_class)s
    {
      private:
        %(reg_idx_arr_decl)s;

      public:
        /// Constructor.
        %(class_name)s(MachInst machInst);
        Fault execute(ExecContext *, Trace::InstRecord *) const override;
        using %(base_class)s::generateDisassembly;
    };
}};
```

src/arch/riscv/isa/formats/basic.isa

→

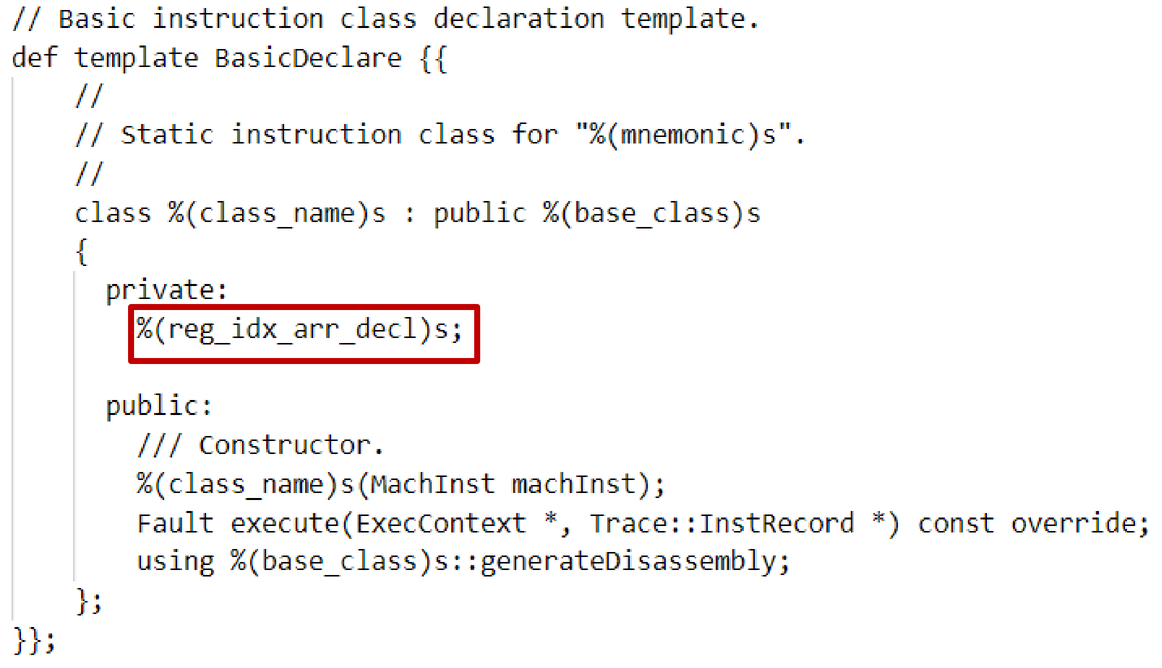


```
class Add : public RegOp
{
  private:
    RegId srcRegIdxArr[2]; RegId destRegIdxArr[1];

  public:
    /// Constructor.
    Add(MachInst machInst);
    Fault execute(ExecContext *, Trace::InstRecord *) const override;
    using RegOp::generateDisassembly;
};
```

build/RISCV/arch/riscv/generated/decoder-ns.hh.inc

gem5

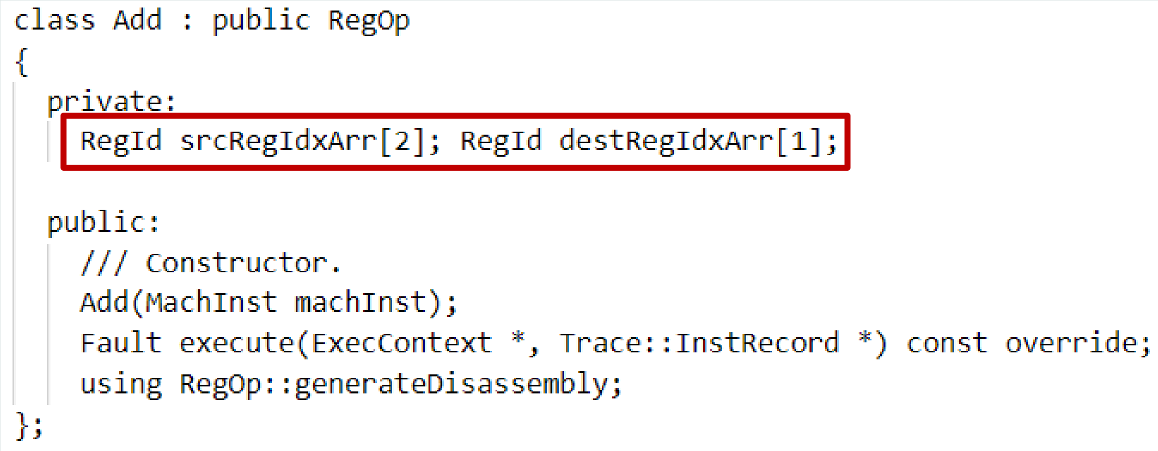
BasicDeclare Template



```
// Basic instruction class declaration template.
def template BasicDeclare {{
    //
    // Static instruction class for "%(mnemonic)s".
    //
    class %(class_name)s : public %(base_class)s
    {
      private:
        %(reg_idx_arr_decl)s;

      public:
        /// Constructor.
        %(class_name)s(MachInst machInst);
        Fault execute(ExecContext *, Trace::InstRecord *) const override;
        using %(base_class)s::generateDisassembly;
    };
}};
```

src/arch/riscv/isa/formats/basic.isa

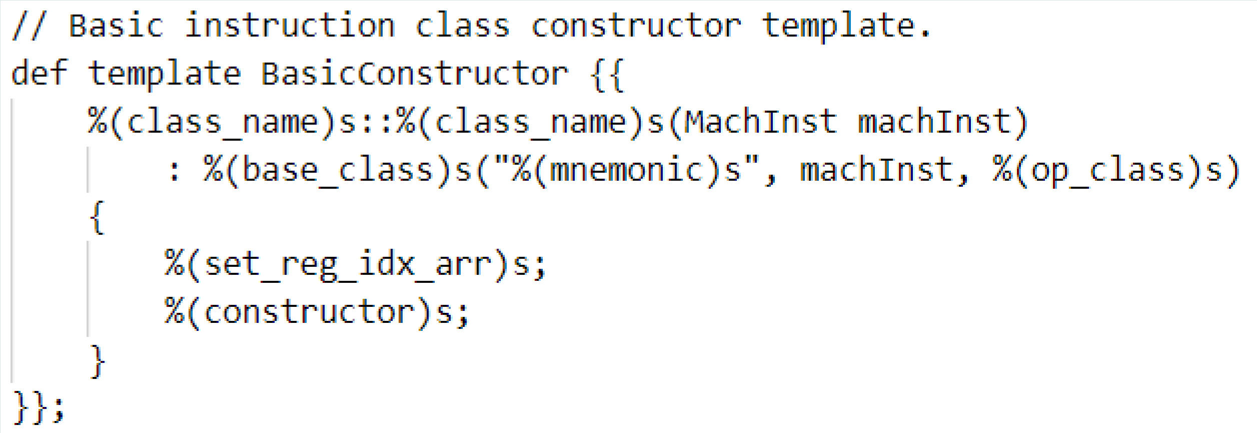
→



```
class Add : public RegOp
{
  private:
    RegId srcRegIdxArr[2]; RegId destRegIdxArr[1];

  public:
    /// Constructor.
    Add(MachInst machInst);
    Fault execute(ExecContext *, Trace::InstRecord *) const override;
    using RegOp::generateDisassembly;
};
```

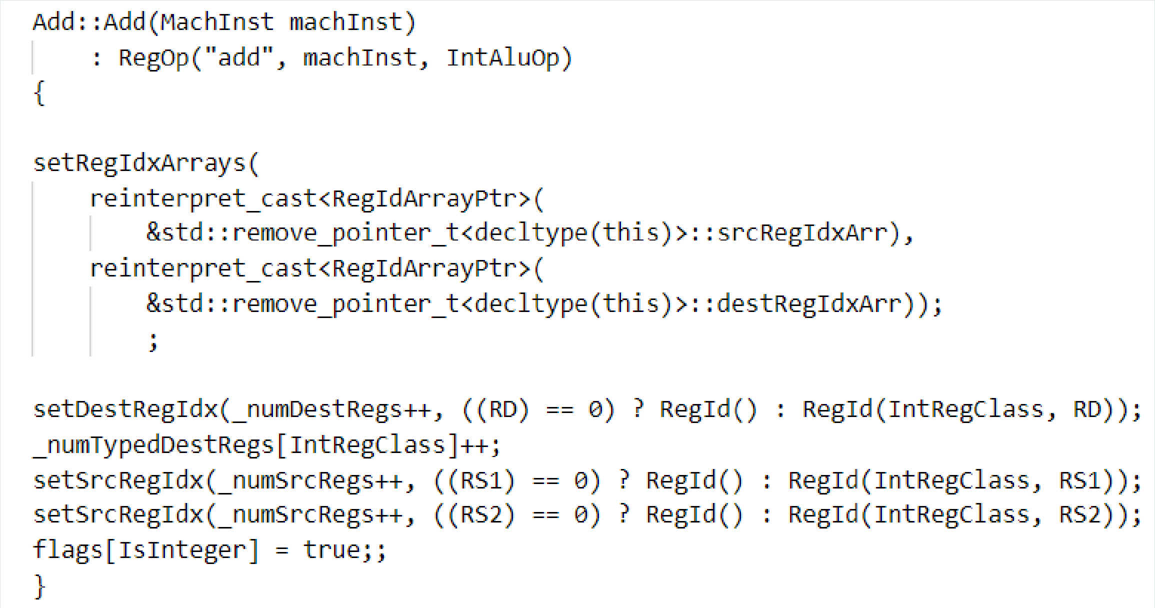
build/RISCV/arch/riscv/generated/decoder-ns.hh.inc

BasicConstructor Template



```
// Basic instruction class constructor template.
def template BasicConstructor {{
    %(class_name)s::%(class_name)s(MachInst machInst)
        : %(base_class)s("%(mnemonic)s", machInst, %(op_class)s)
    {
        %(set_reg_idx_arr)s;
        %(constructor)s;
    }
}};
```

src/arch/riscv/isa/formats/basic.isa



```
Add::Add(MachInst machInst)
    : RegOp("add", machInst, IntAluOp)
{

setRegIdxArrays(
    reinterpret_cast<RegIdArrayPtr>(
        &std::remove_pointer_t<decltype(this)>::srcRegIdxArr),
    reinterpret_cast<RegIdArrayPtr>(
        &std::remove_pointer_t<decltype(this)>::destRegIdxArr));
        ;

setDestRegIdx(_numDestRegs++, ((RD) == 0) ? RegId() : RegId(IntRegClass, RD));
_numTypedDestRegs[IntRegClass]++;
setSrcRegIdx(_numSrcRegs++, ((RS1) == 0) ? RegId() : RegId(IntRegClass, RS1));
setSrcRegIdx(_numSrcRegs++, ((RS2) == 0) ? RegId() : RegId(IntRegClass, RS2));
flags[IsInteger] = true;;
}
```

build/RISCV/arch/riscv/generated/decoder-ns.cc.inc

Differences for Memory Operations

exec_output for memory Operations

- Generated through three different templates

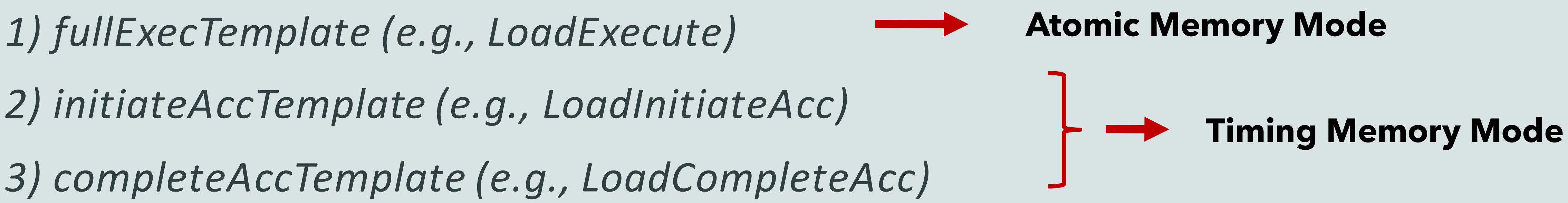
1) fullExecTemplate (e.g., LoadExecute)

2) initiateAccTemplate (e.g., LoadInitiateAcc)

3) completeAccTemplate (e.g., LoadCompleteAcc)

exec_output for memory Operations

- Generated through three different templates



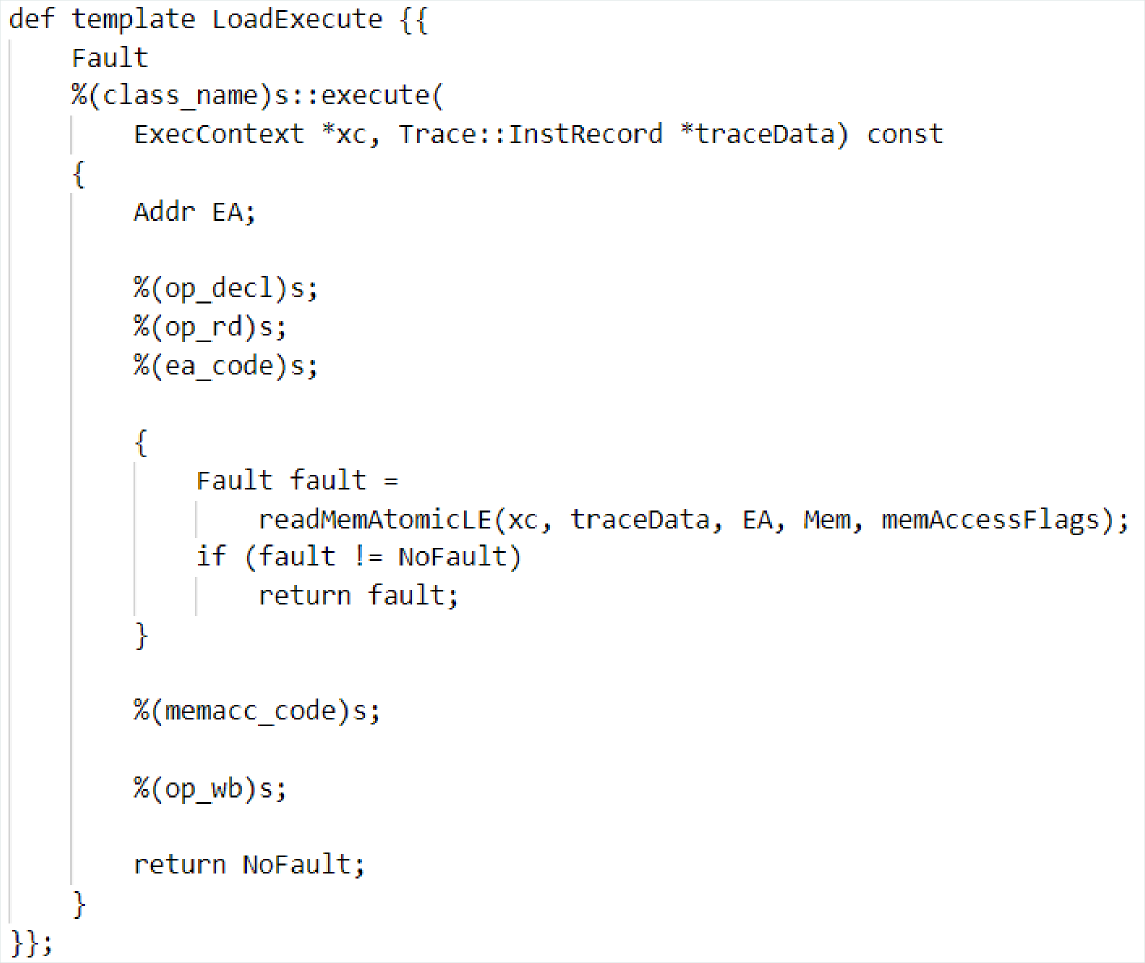
1) fullExecTemplate (e.g., LoadExecute) → **Atomic Memory Mode**

2) initiateAccTemplate (e.g., LoadInitiateAcc)

3) completeAccTemplate (e.g., LoadCompleteAcc)

} → **Timing Memory Mode**

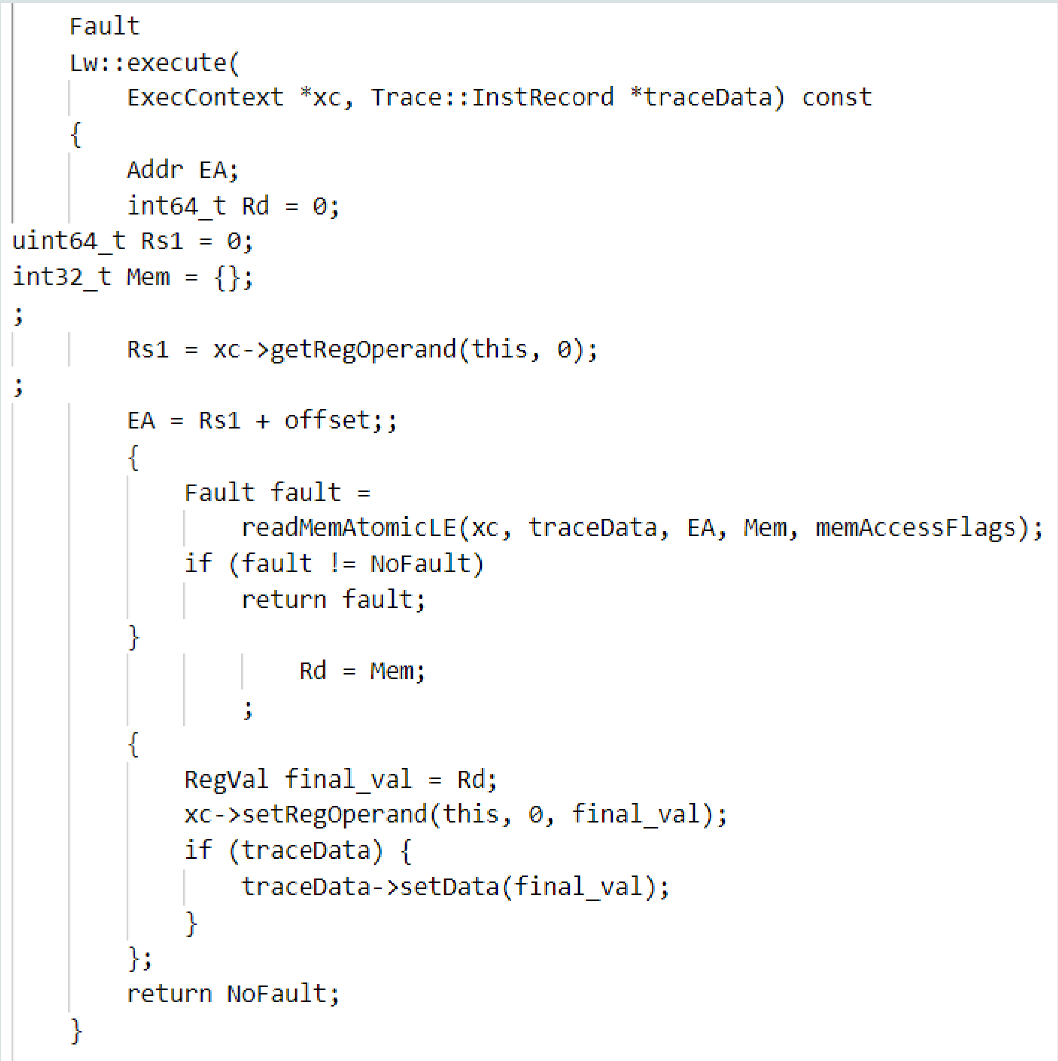
LoadExecute (Atomic)



```
def template LoadExecute {{
    Fault
    %(class_name)s::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }

        %(memacc_code)s;

        %(op_wb)s;

        return NoFault;
    }
}};
```

src/arch/riscv/isa/formats/mem.isa

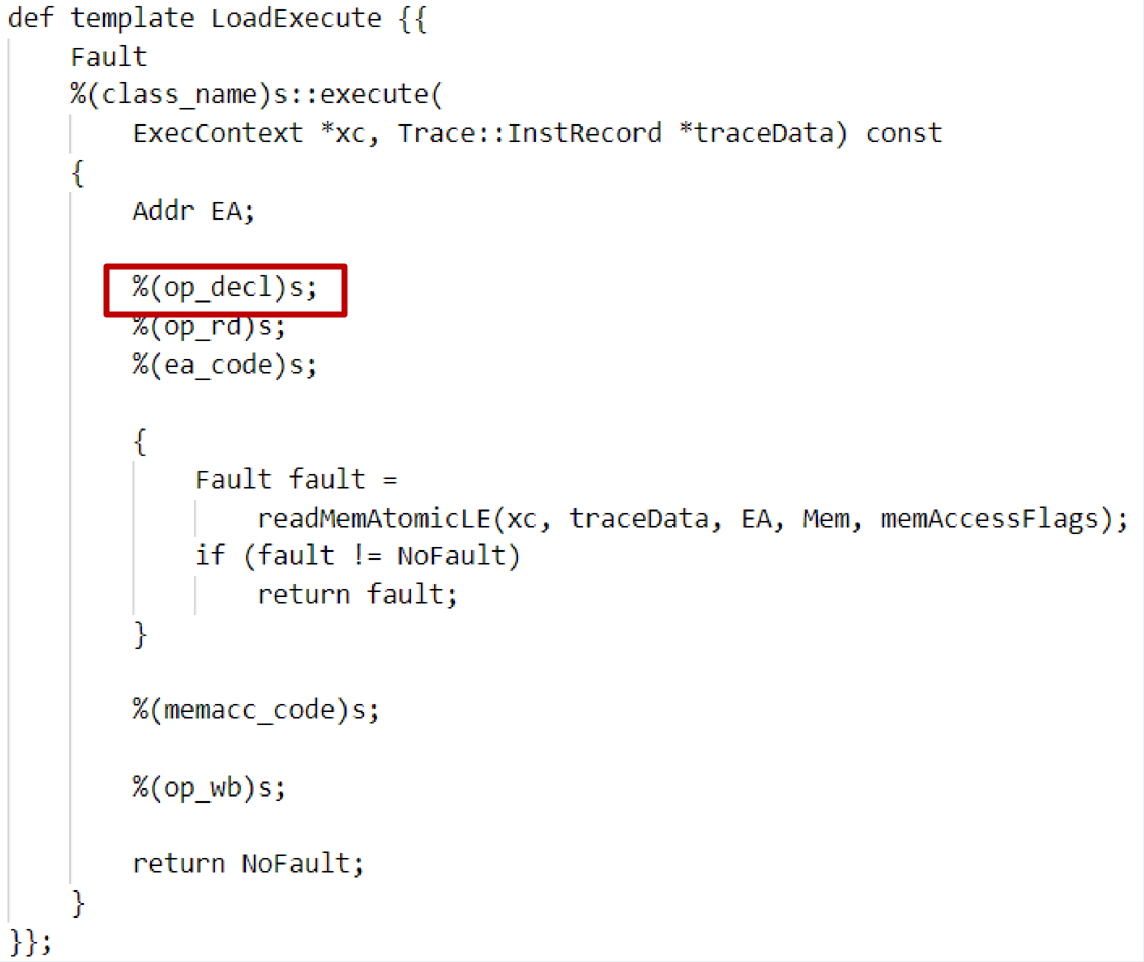
→



```
    Fault
    Lw::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;
        int64_t Rd = 0;
uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;
        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }
                    Rd = Mem;
                ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

build/RISCV/arch/riscv/generated/exec-ns.cc.inc

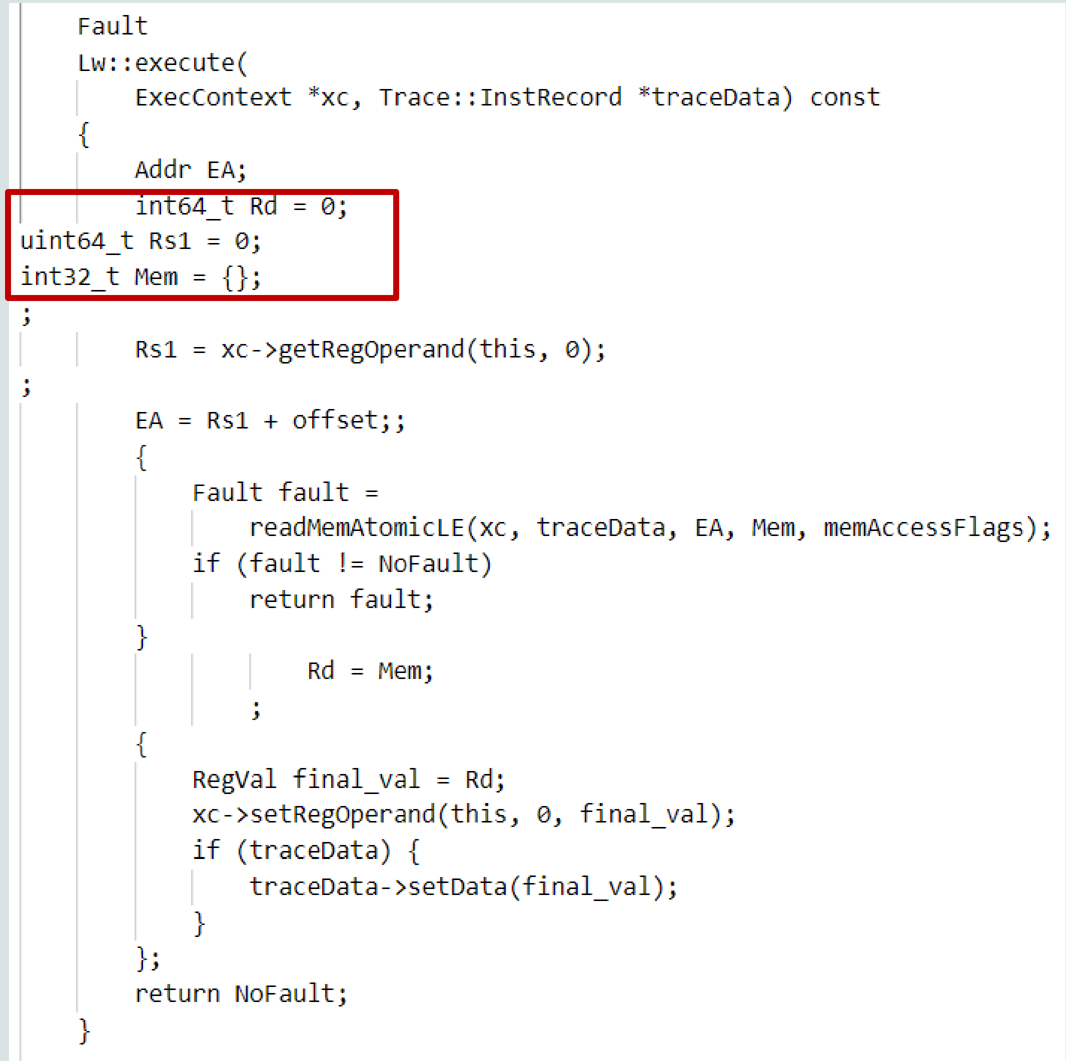
LoadExecute (Atomic)



```
def template LoadExecute {{
    Fault
    %(class_name)s::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }

        %(memacc_code)s;

        %(op_wb)s;

        return NoFault;
    }
}};
```

src/arch/riscv/isa/formats/mem.isa

→

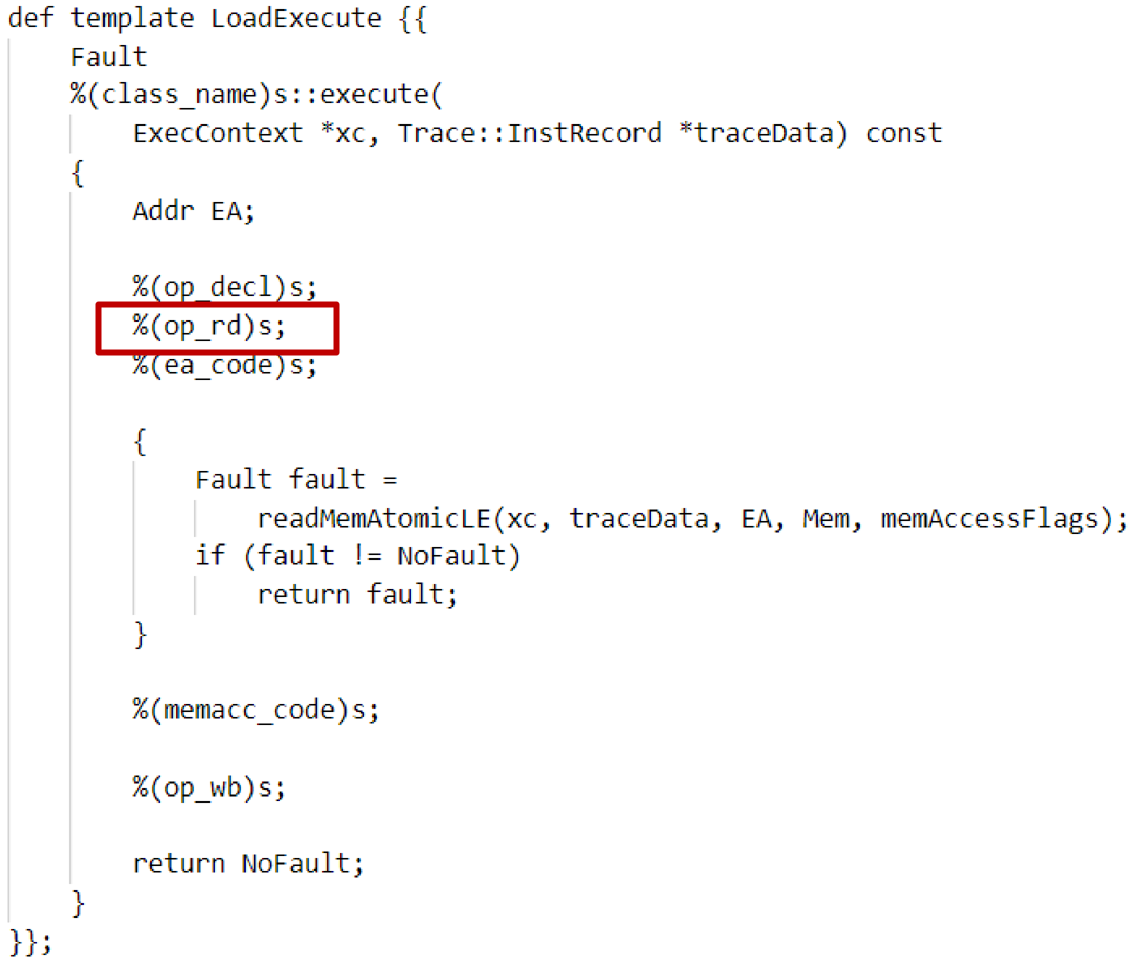


```
    Fault
    Lw::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;
        int64_t Rd = 0;
uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;
        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }
                    Rd = Mem;
                ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

build/RISCV/arch/riscv/generated/exec-ns.cc.inc

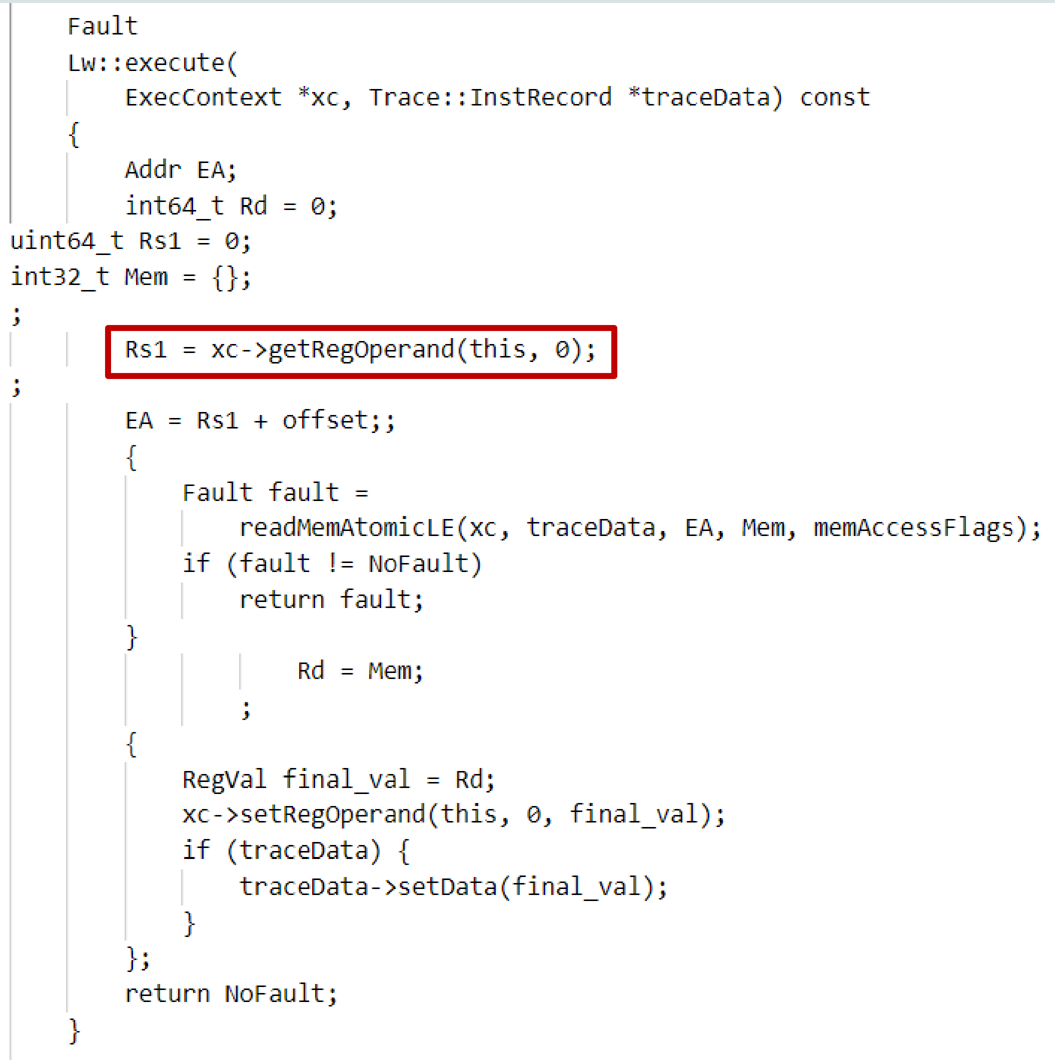
gem5

LoadExecute (Atomic)



```
def template LoadExecute {{
    Fault
    %(class_name)s::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }

        %(memacc_code)s;

        %(op_wb)s;

        return NoFault;
    }
}};
```

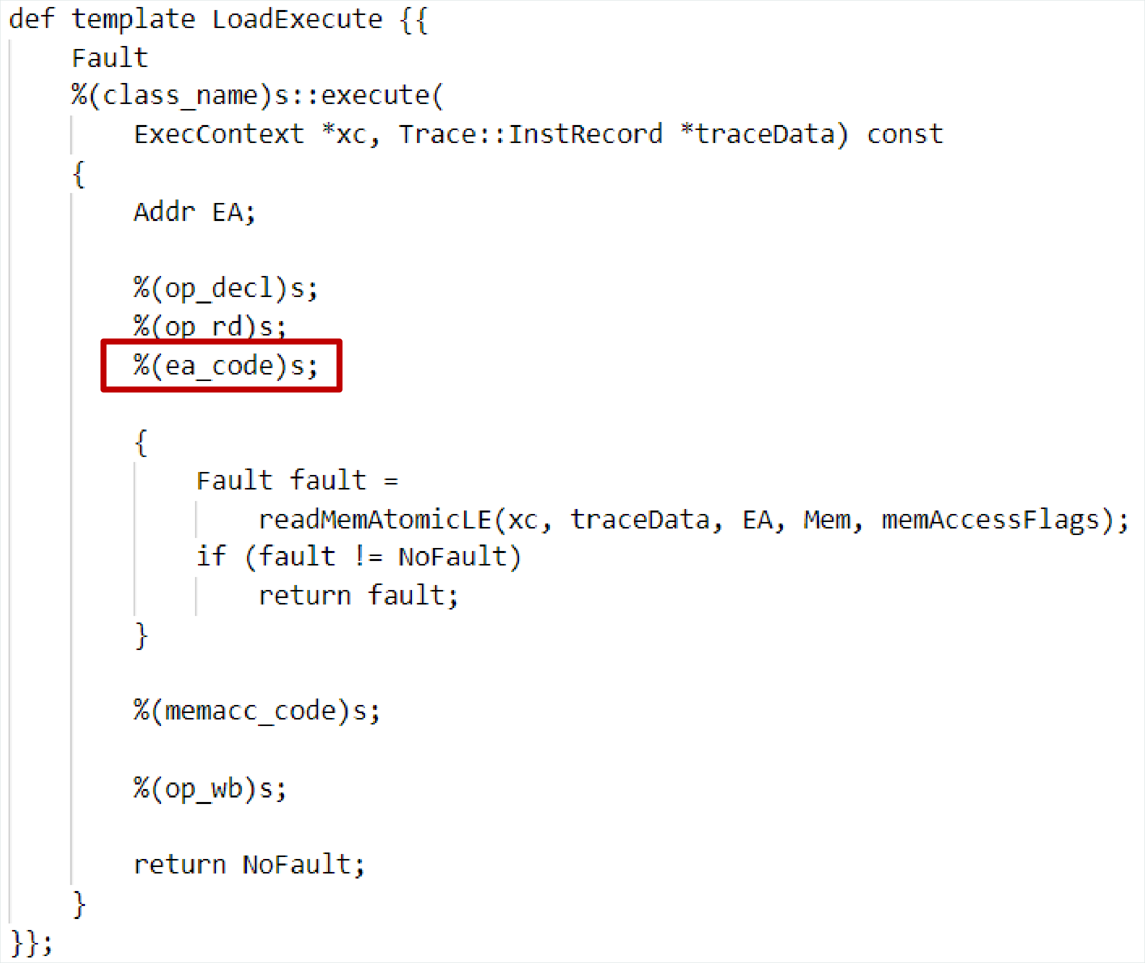
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;
        int64_t Rd = 0;
uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;
        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }
                    Rd = Mem;
                ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

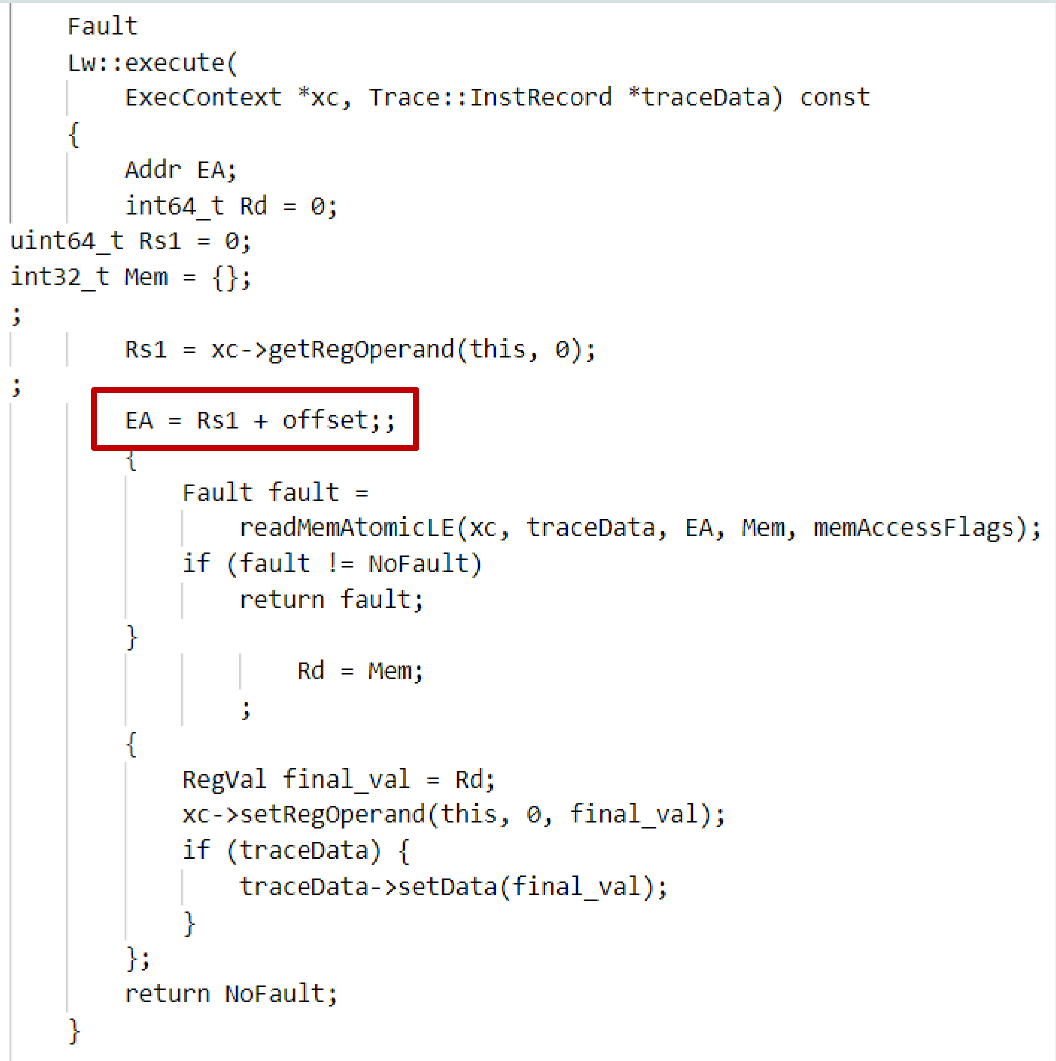
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadExecute (Atomic)



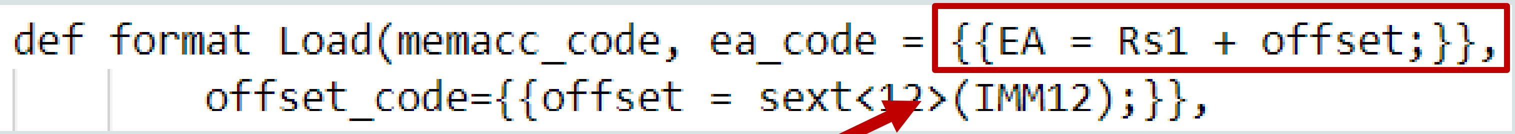
```
def template LoadExecute {{
    Fault
    %(class_name)s::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }

        %(memacc_code)s;

        %(op_wb)s;

        return NoFault;
    }
}};
```

src/arch/riscv/isa/formats/mem.isa

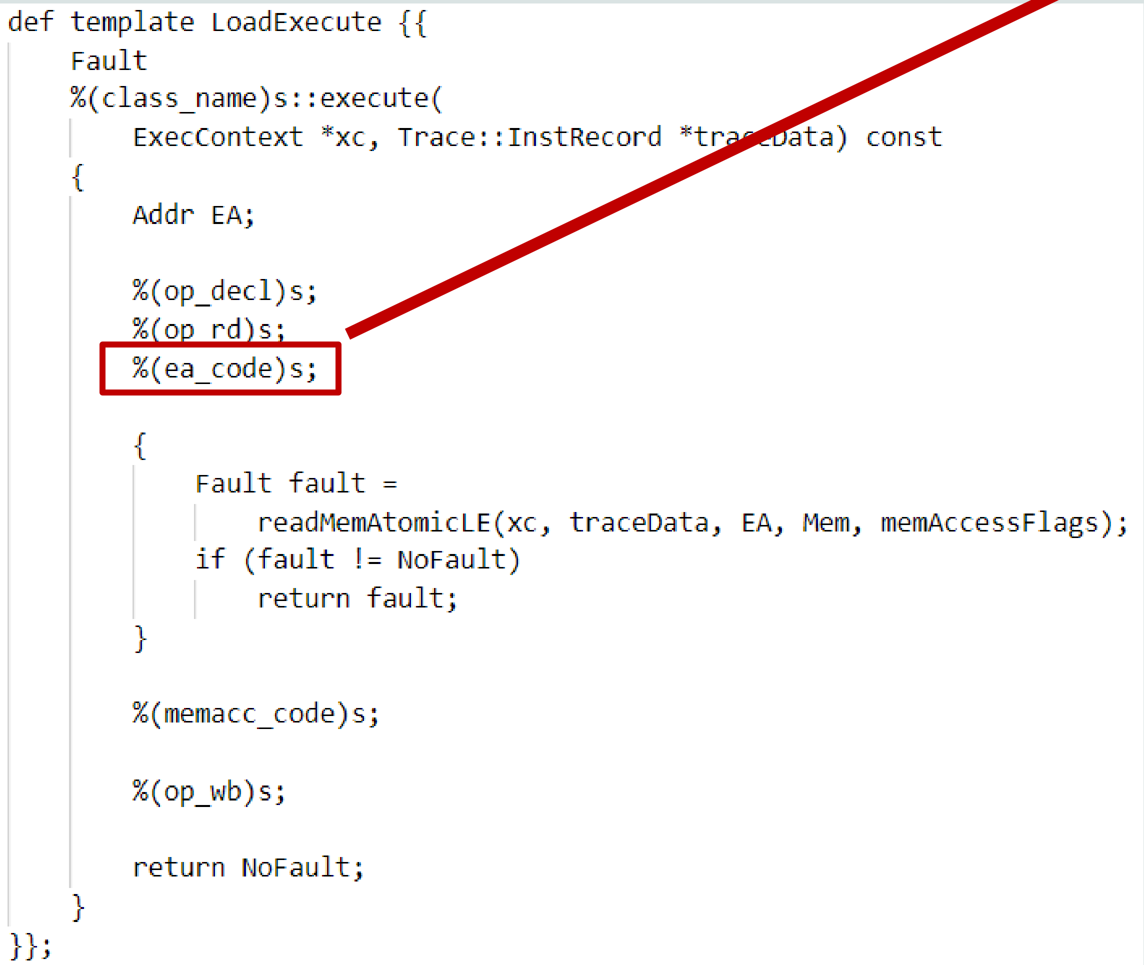


```
    Fault
    Lw::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;
        int64_t Rd = 0;
uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;
        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }
                    Rd = Mem;
                ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

build/RISCV/arch/riscv/generated/exec-ns.cc.inc

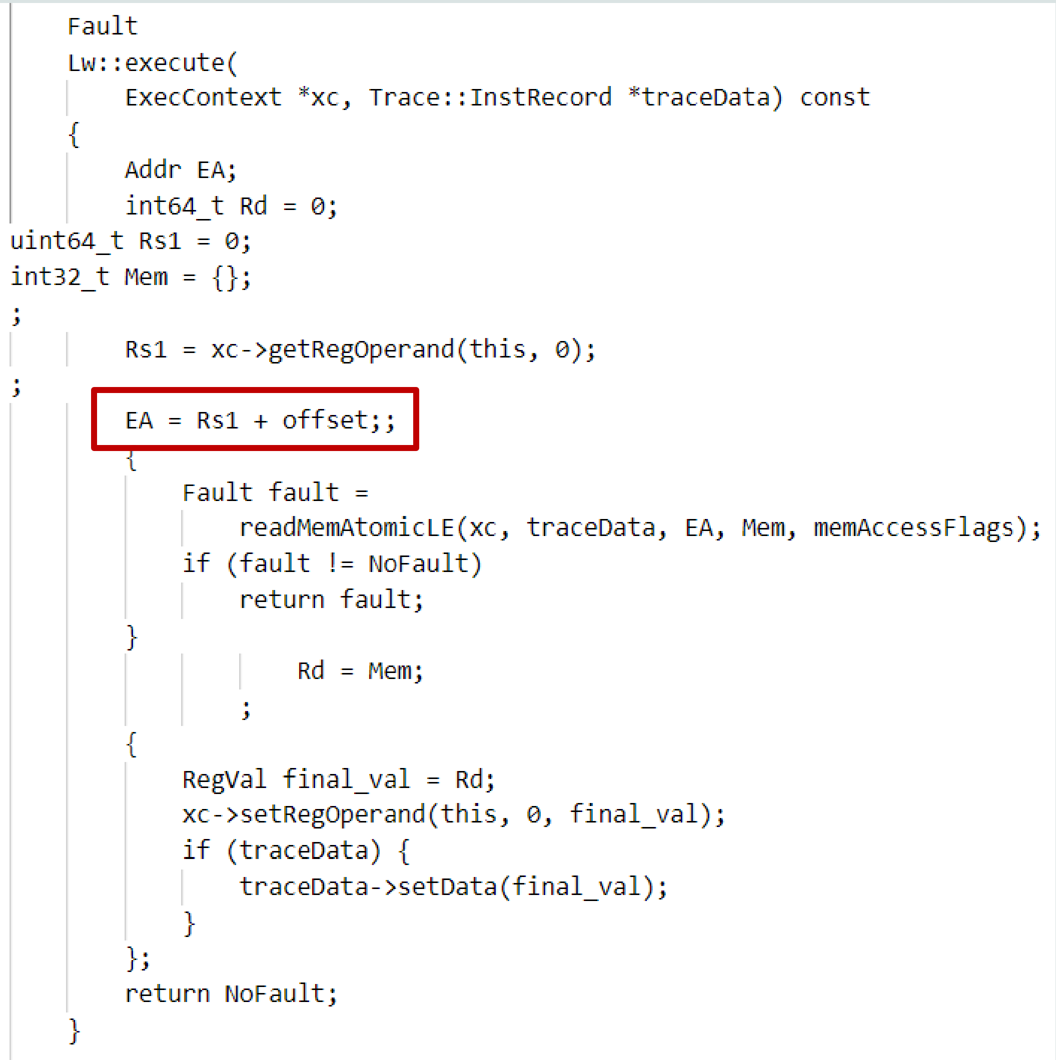


```
def format Load(memacc_code, ea_code = {{EA = Rs1 + offset;}},
        offset_code={{offset = sext<12>(IMM12);}},
```



```
def template LoadExecute {{
    Fault
    %(class_name)s::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }

        %(memacc_code)s;

        %(op_wb)s;

        return NoFault;
    }
}};
```

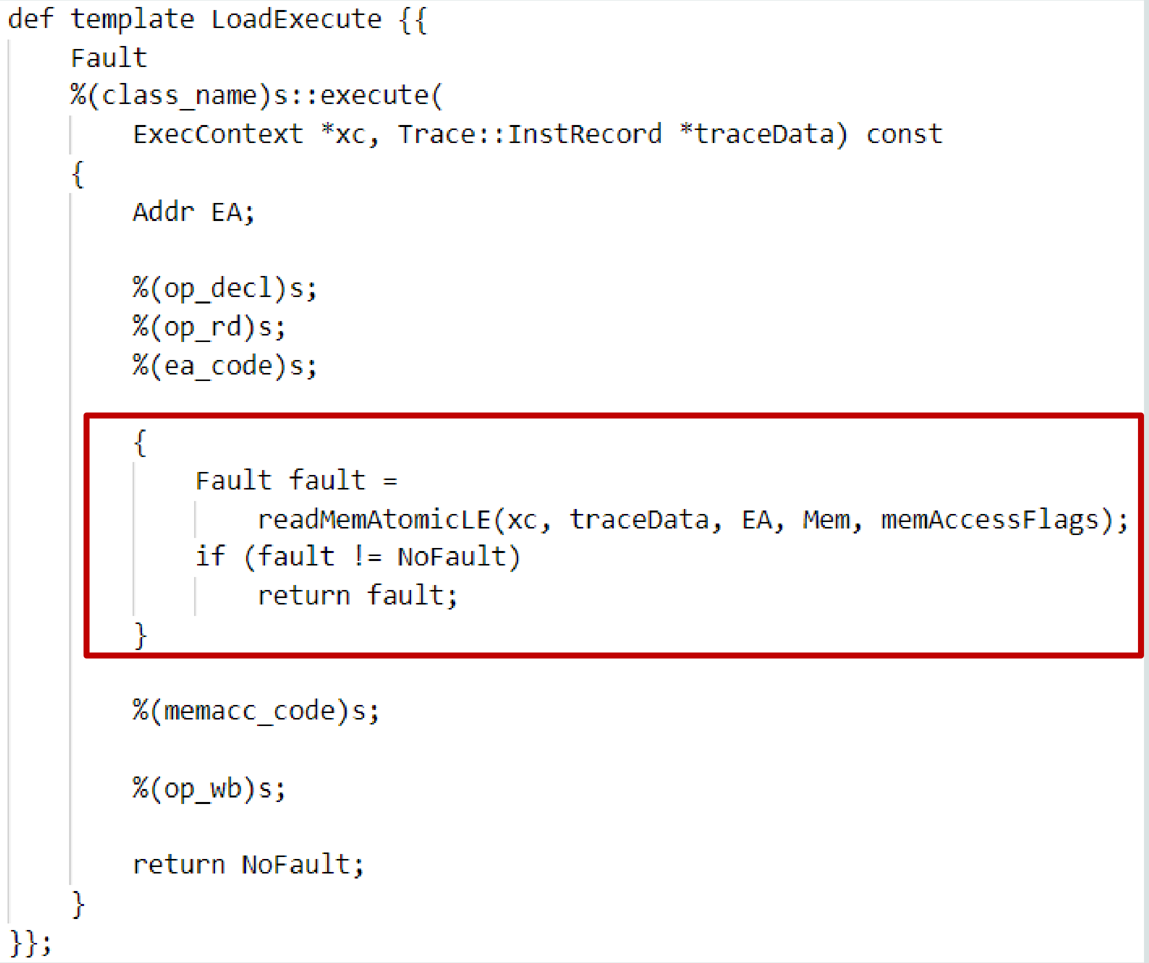
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;
        int64_t Rd = 0;
uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;
        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }
                    Rd = Mem;
                ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

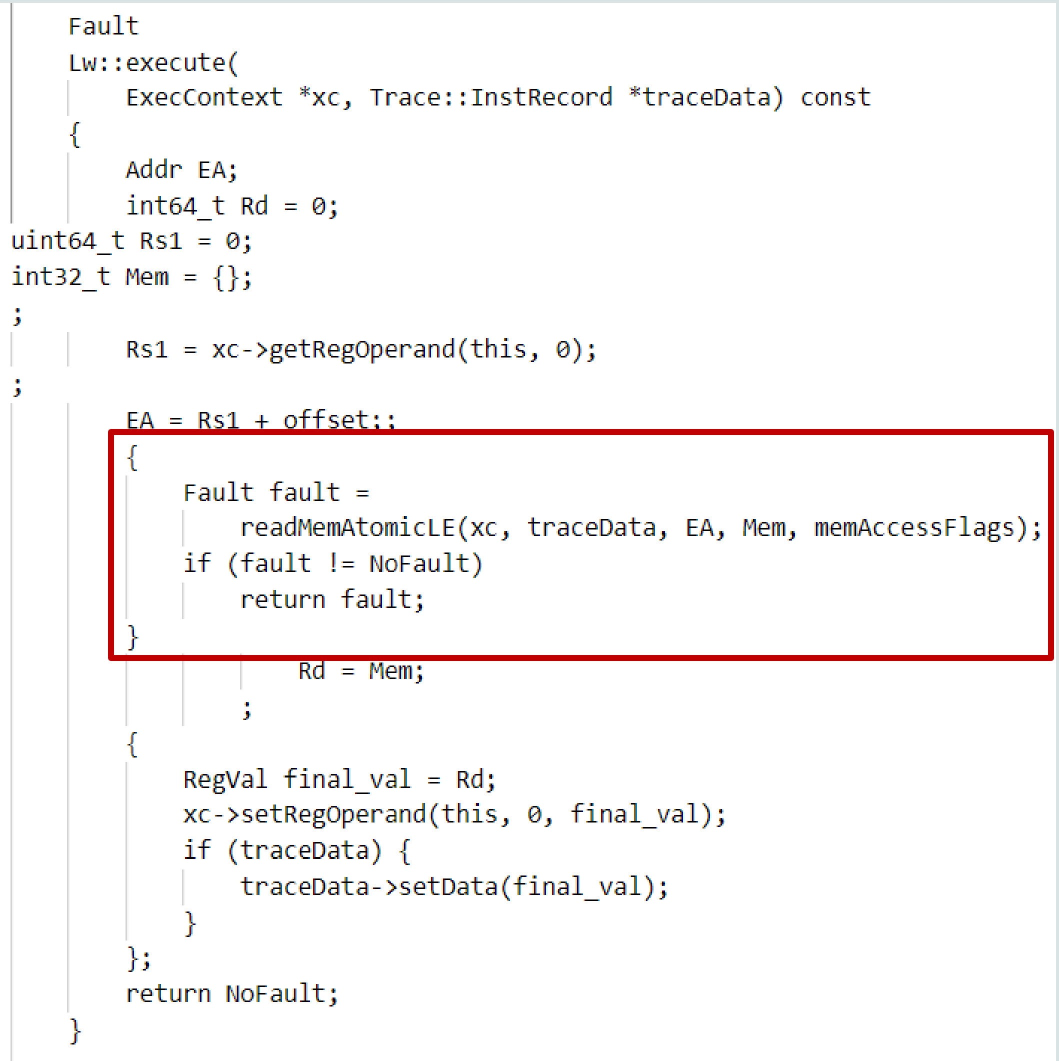
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadExecute (Atomic)



```
def template LoadExecute {{
    Fault
    %(class_name)s::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }

        %(memacc_code)s;

        %(op_wb)s;

        return NoFault;
    }
}};
```

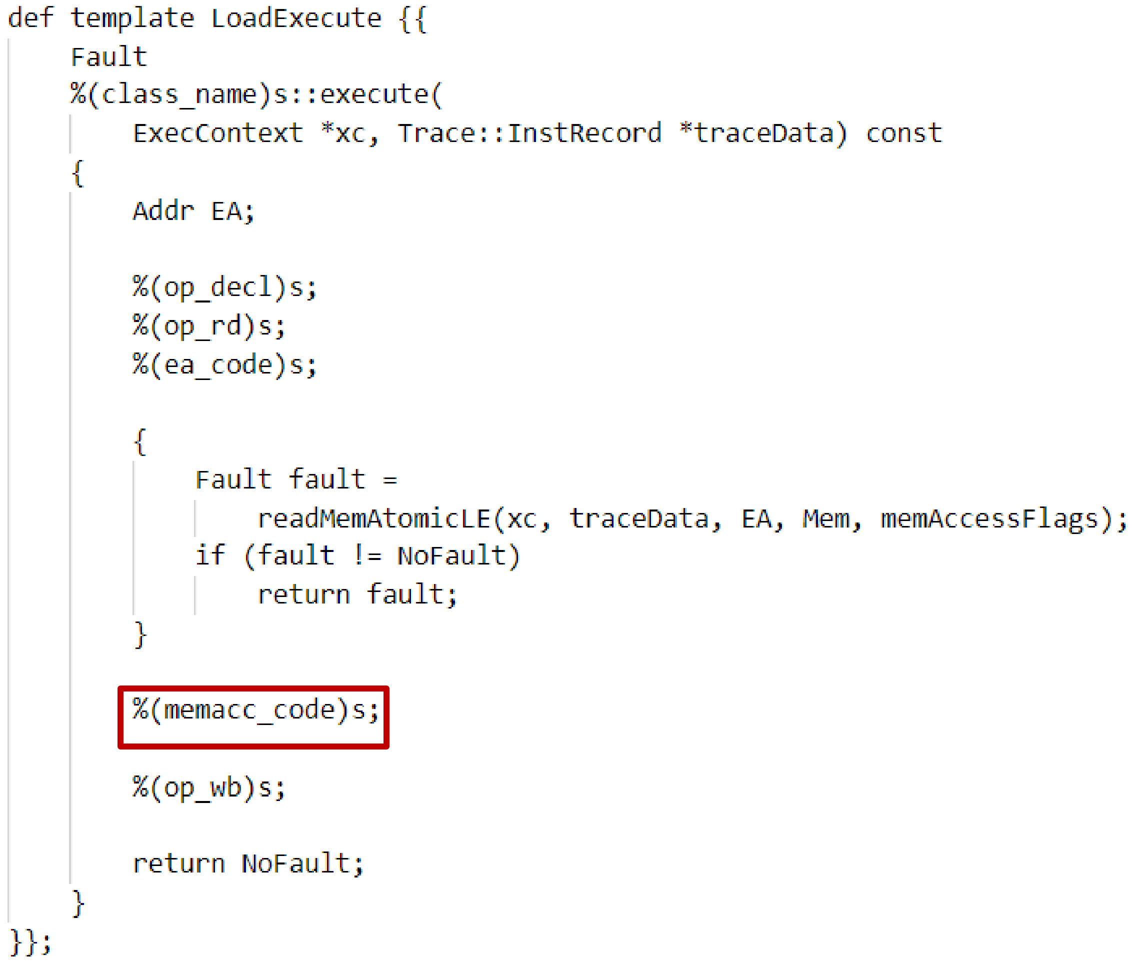
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;
        int64_t Rd = 0;
uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;
        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }
                    Rd = Mem;
                ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

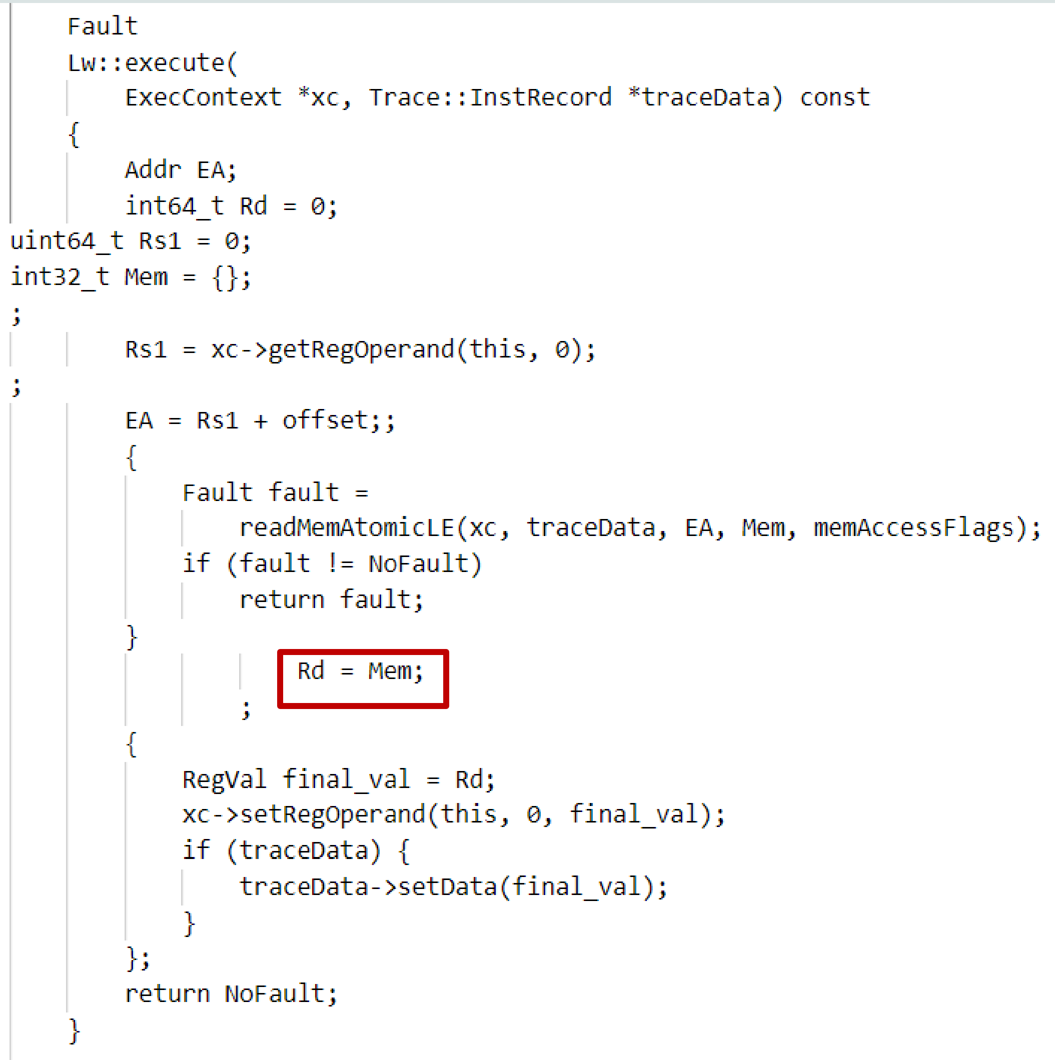
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadExecute (Atomic)



```
def template LoadExecute {{
    Fault
    %(class_name)s::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }

        %(memacc_code)s;

        %(op_wb)s;

        return NoFault;
    }
}};
```

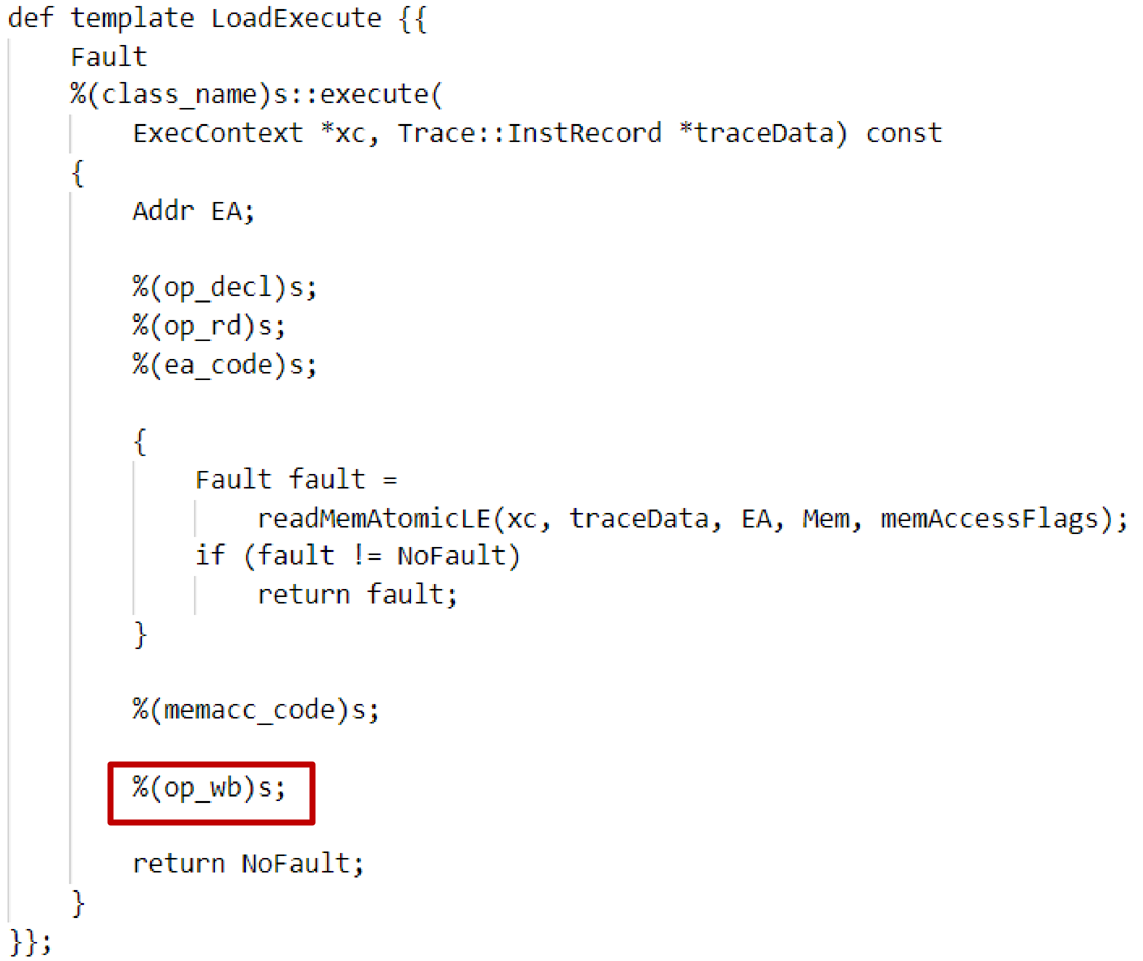
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;
        int64_t Rd = 0;
uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;
        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }
                    Rd = Mem;
                ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

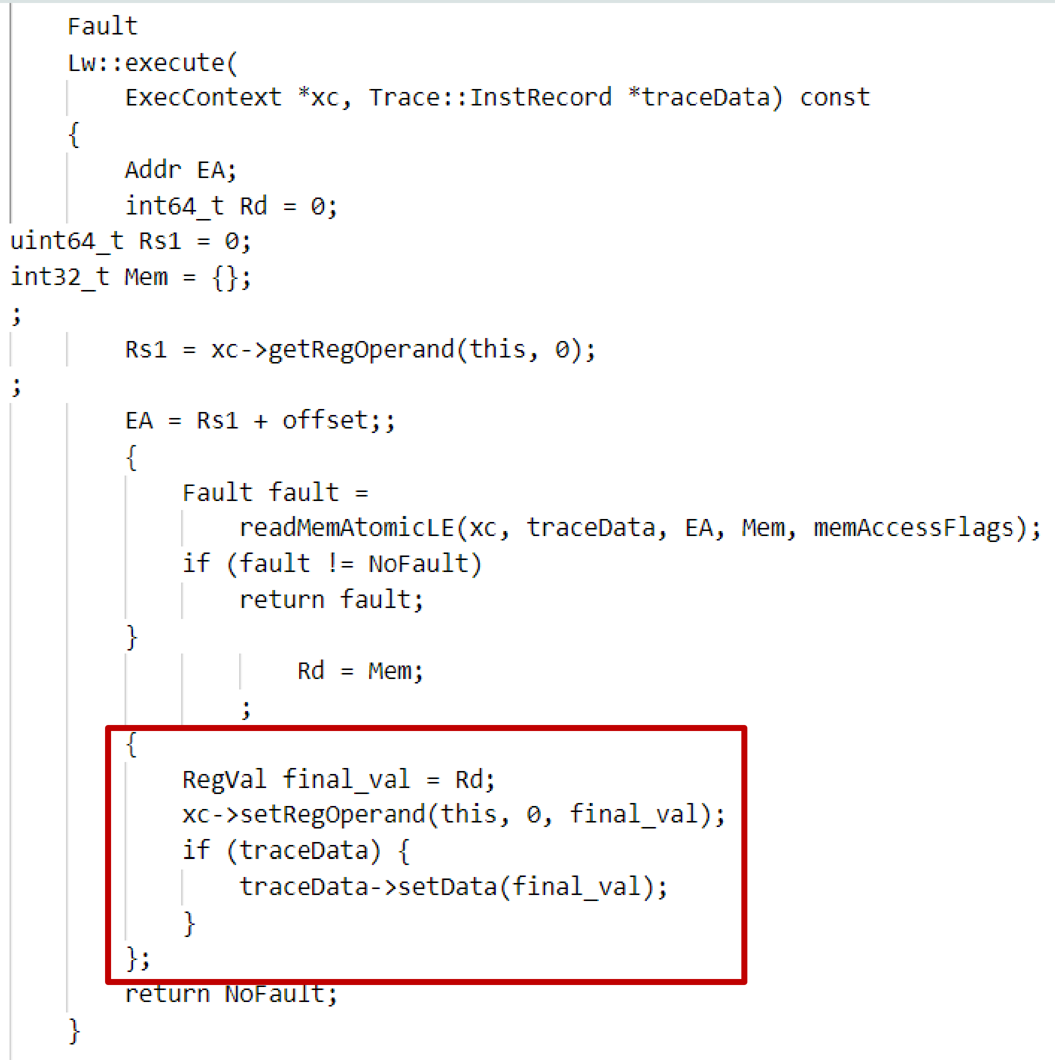
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadExecute (Atomic)



```
def template LoadExecute {{
    Fault
    %(class_name)s::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }

        %(memacc_code)s;

        %(op_wb)s;

        return NoFault;
    }
}};
```

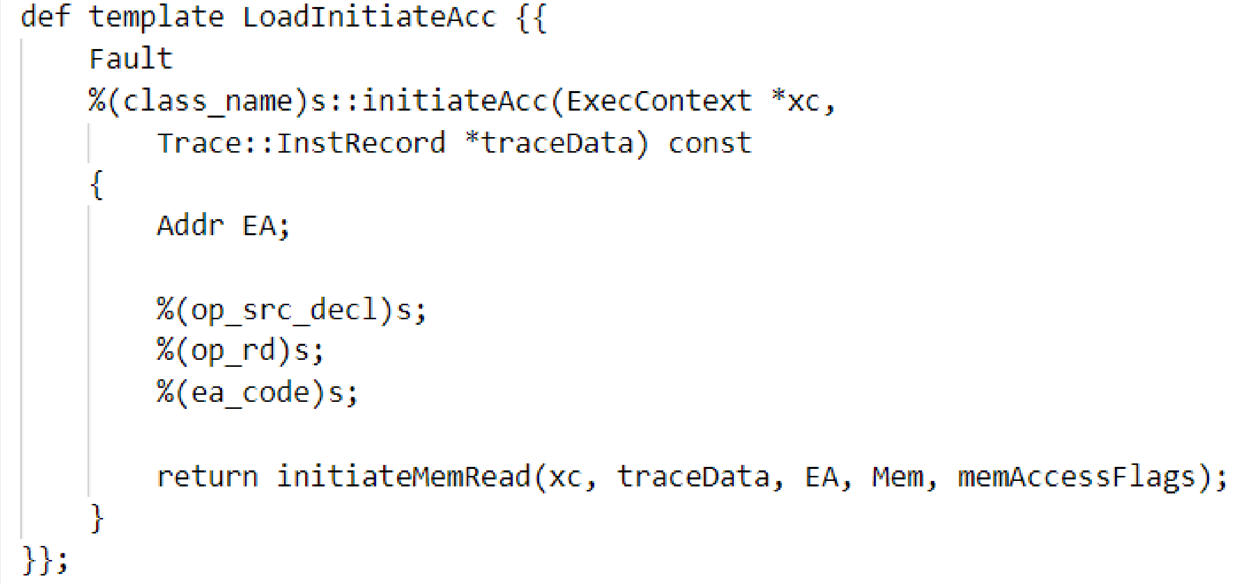
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::execute(
        ExecContext *xc, Trace::InstRecord *traceData) const
    {
        Addr EA;
        int64_t Rd = 0;
uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;
        {
            Fault fault =
                readMemAtomicLE(xc, traceData, EA, Mem, memAccessFlags);
            if (fault != NoFault)
                return fault;
        }
                    Rd = Mem;
                ;
        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };
        return NoFault;
    }
```

build/RISCV/arch/riscv/generated/exec-ns.cc.inc

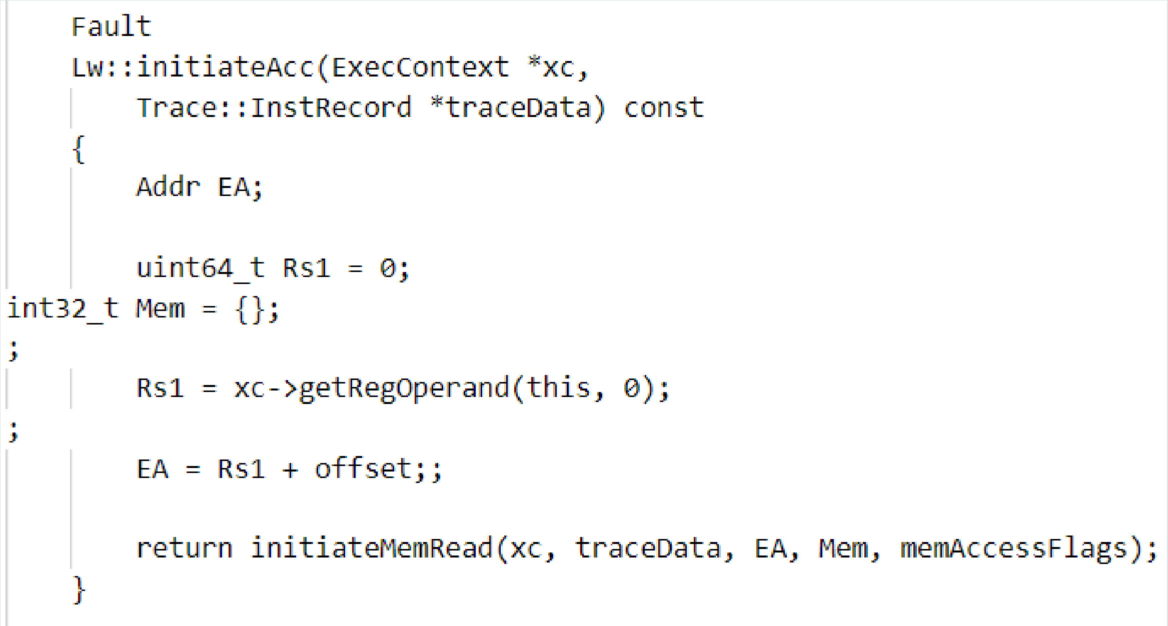
LoadInitiateAcc (Timing)



```
def template LoadInitiateAcc {{
    Fault
    %(class_name)s::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_src_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
}};
```

src/arch/riscv/isa/formats/mem.isa

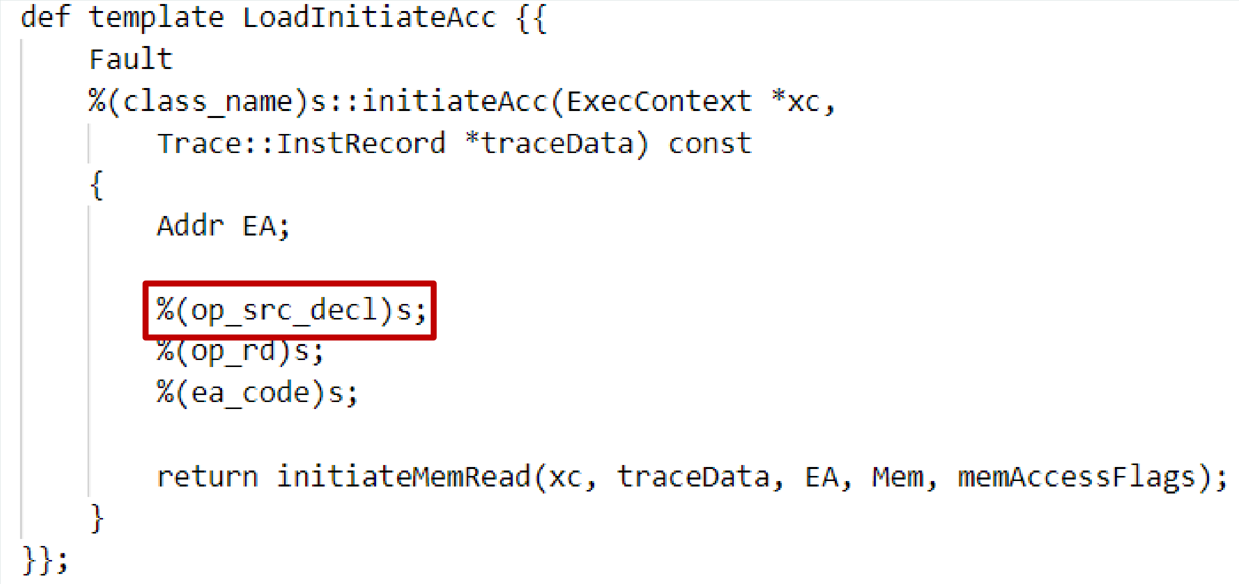
→



```
    Fault
    Lw::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
```

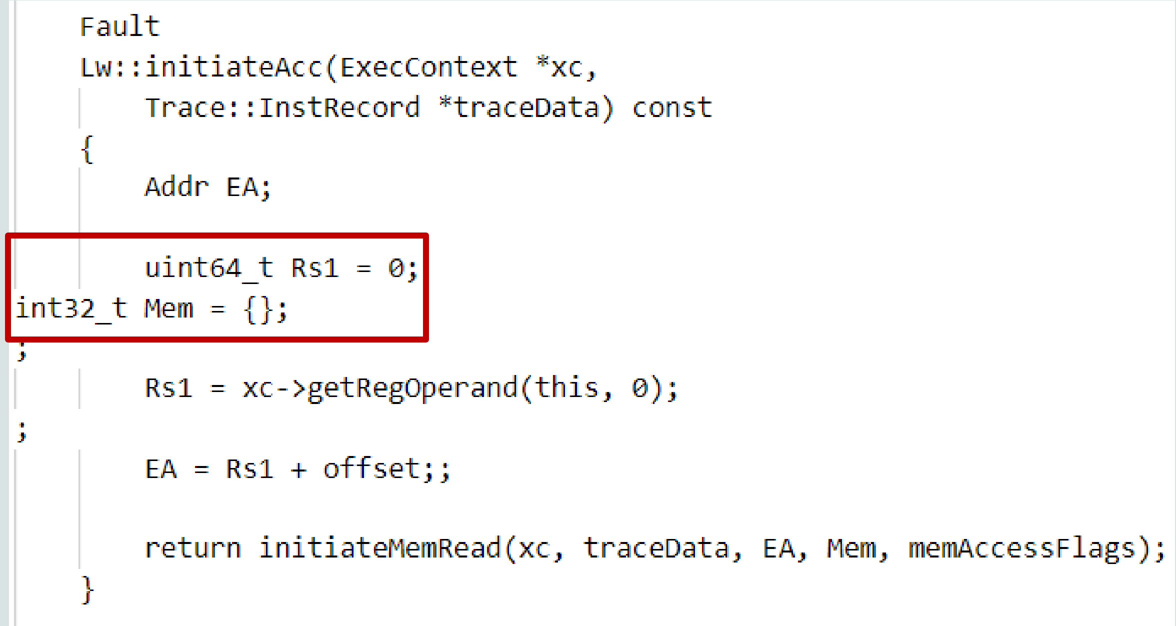
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadInitiateAcc (Timing)



```
def template LoadInitiateAcc {{
    Fault
    %(class_name)s::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_src_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
}};
```

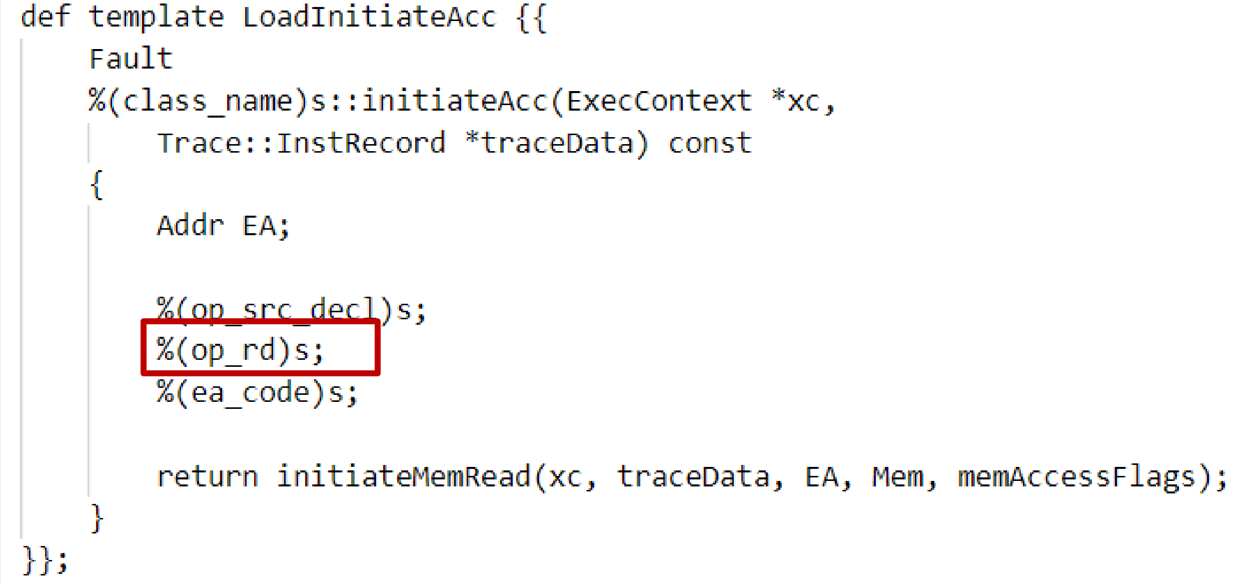
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
```

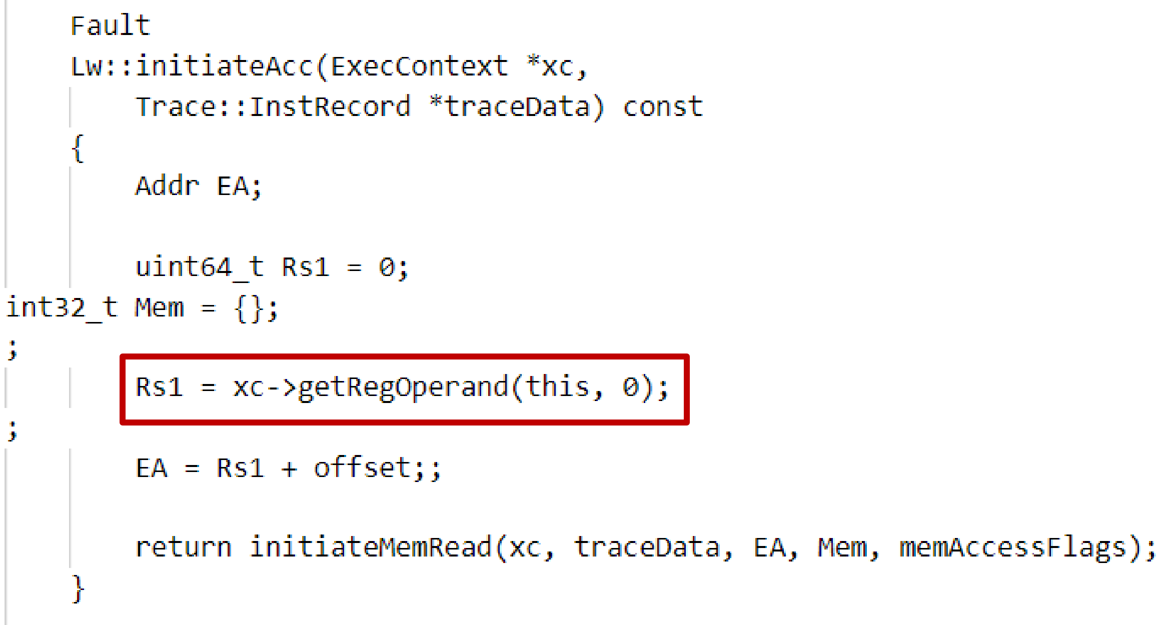
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadInitiateAcc (Timing)



```
def template LoadInitiateAcc {{
    Fault
    %(class_name)s::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_src_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
}};
```

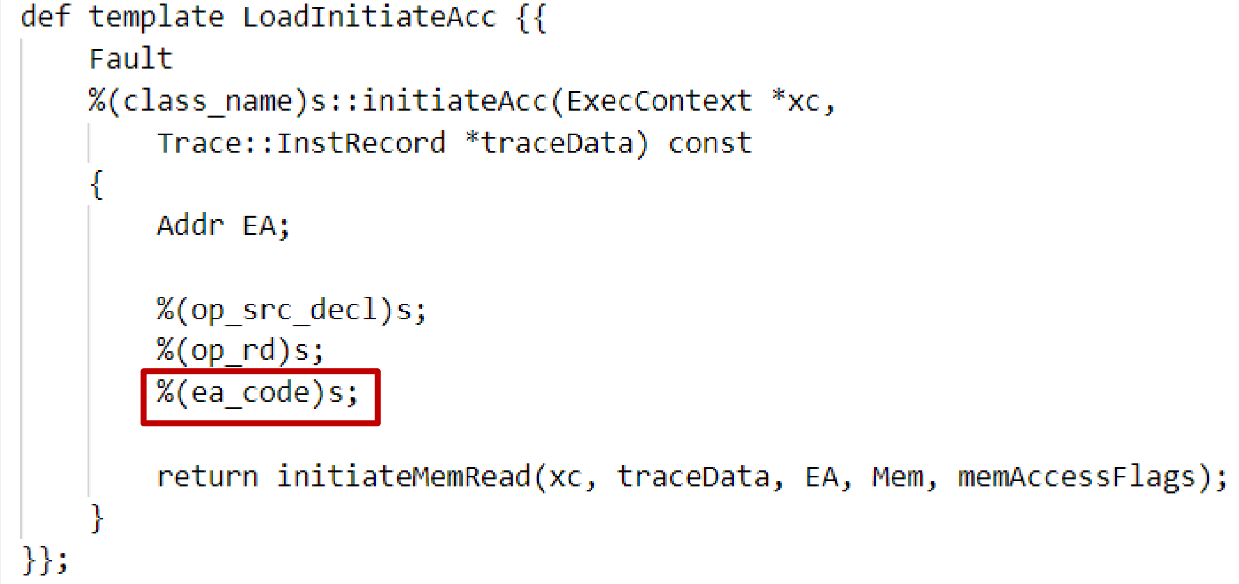
src/arch/riscv/isa/formats/mem.isa



```
  Fault
  Lw::initiateAcc(ExecContext *xc,
    Trace::InstRecord *traceData) const
  {
    Addr EA;

    uint64_t Rs1 = 0;
int32_t Mem = {};
;
    Rs1 = xc->getRegOperand(this, 0);
;
    EA = Rs1 + offset;;

    return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
  }
```

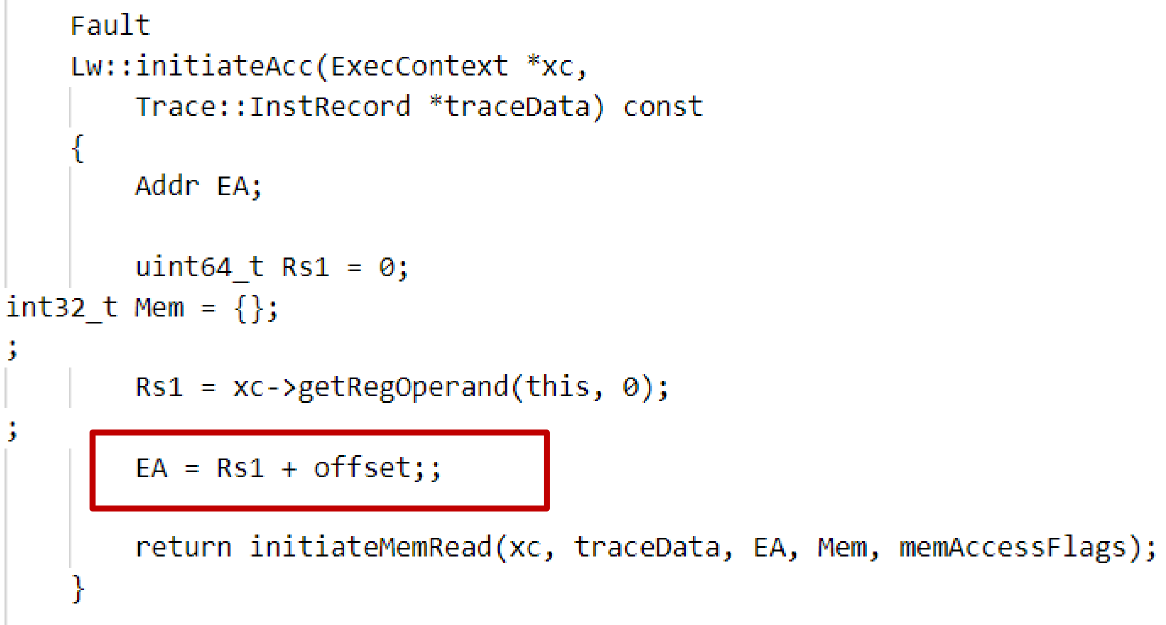
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadInitiateAcc (Timing)



```
def template LoadInitiateAcc {{
    Fault
    %(class_name)s::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_src_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
}};
```

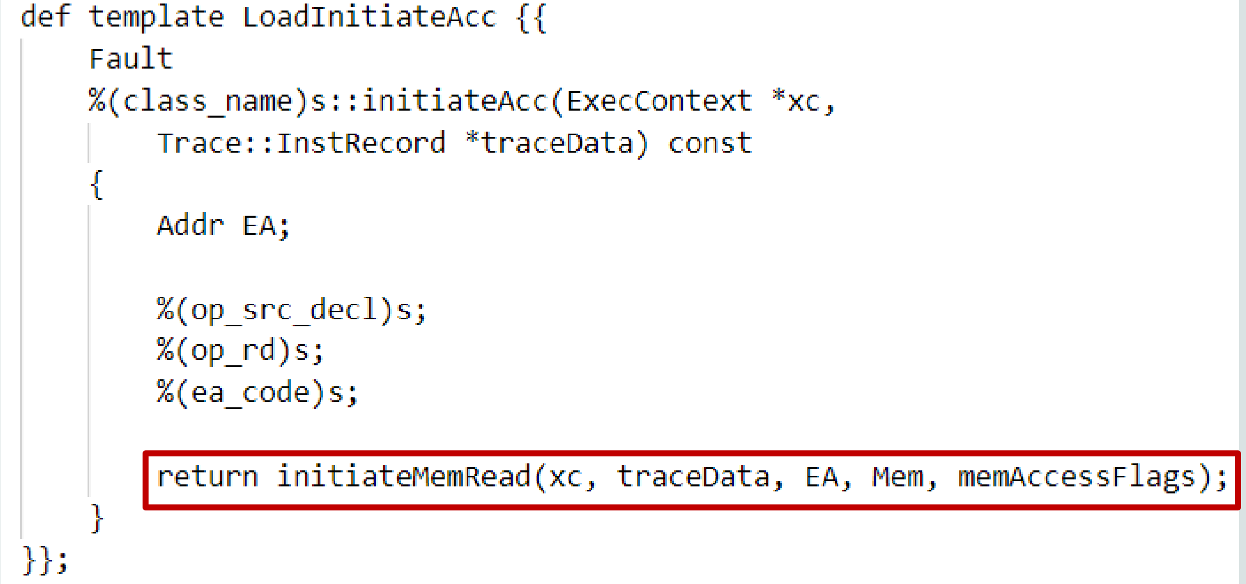
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
```

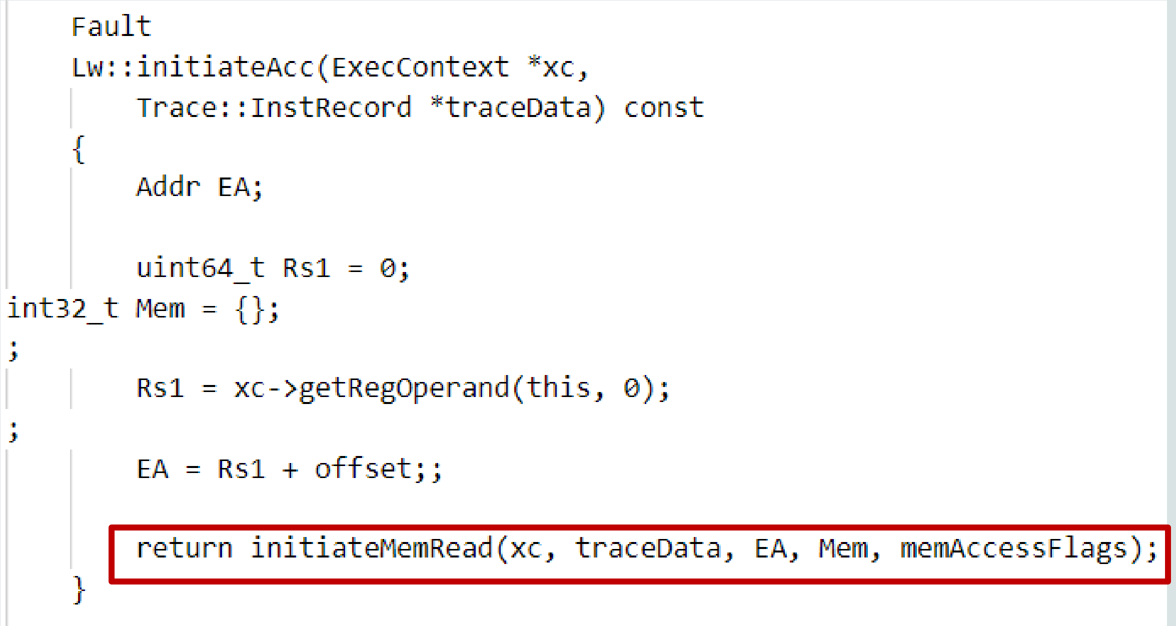
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadInitiateAcc (Timing)



```
def template LoadInitiateAcc {{
    Fault
    %(class_name)s::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        %(op_src_decl)s;
        %(op_rd)s;
        %(ea_code)s;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
}};
```

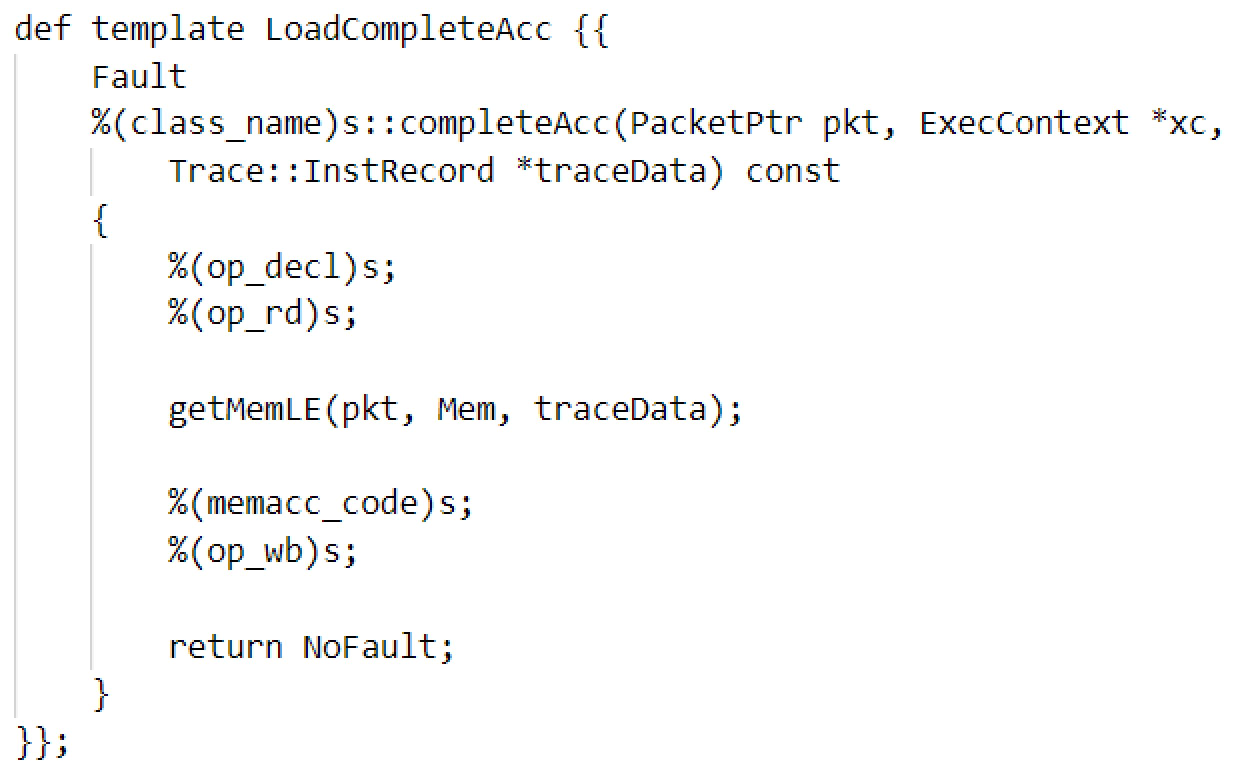
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::initiateAcc(ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        Addr EA;

        uint64_t Rs1 = 0;
int32_t Mem = {};
;
        Rs1 = xc->getRegOperand(this, 0);
;
        EA = Rs1 + offset;;

        return initiateMemRead(xc, traceData, EA, Mem, memAccessFlags);
    }
```

build/RISCV/arch/riscv/generated/exec-ns.cc.inc

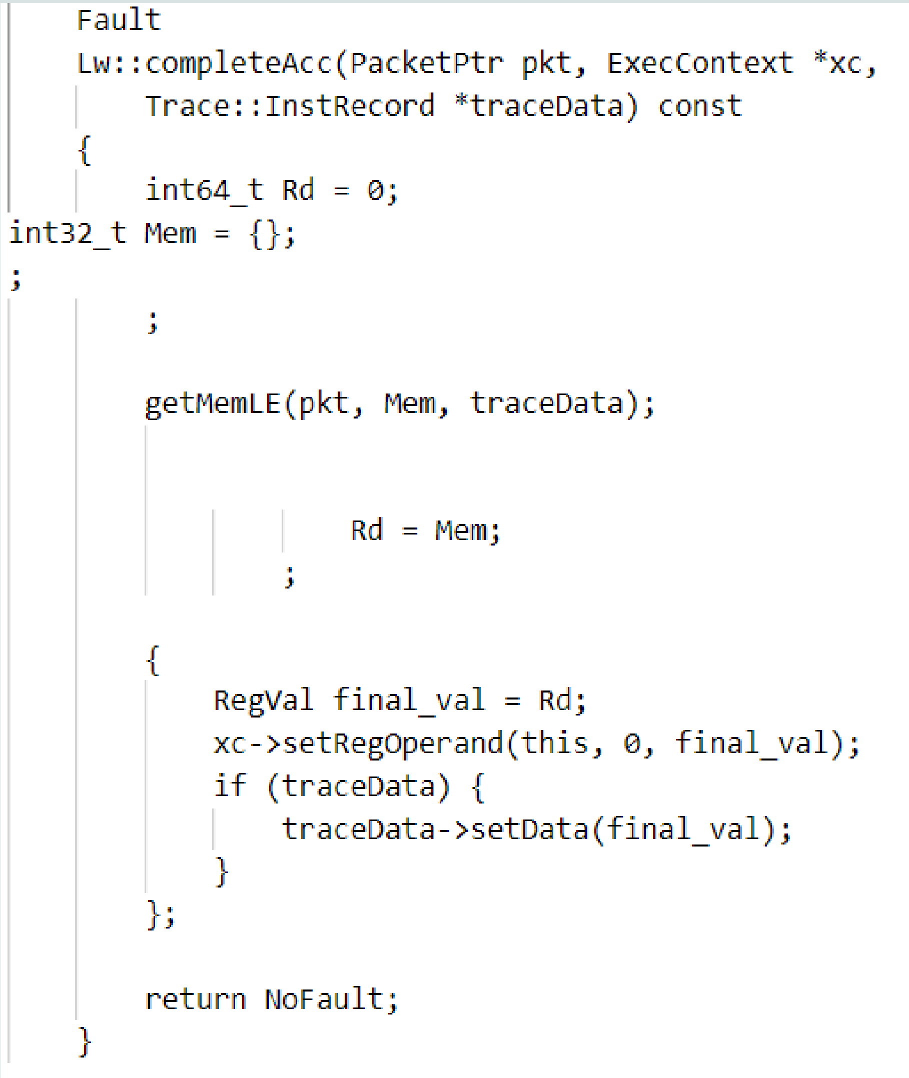
LoadCompleteAcc (Timing)



```
def template LoadCompleteAcc {{
    Fault
    %(class_name)s::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        %(op_decl)s;
        %(op_rd)s;

        getMemLE(pkt, Mem, traceData);

        %(memacc_code)s;
        %(op_wb)s;

        return NoFault;
    }
}};
```

src/arch/riscv/isa/formats/mem.isa

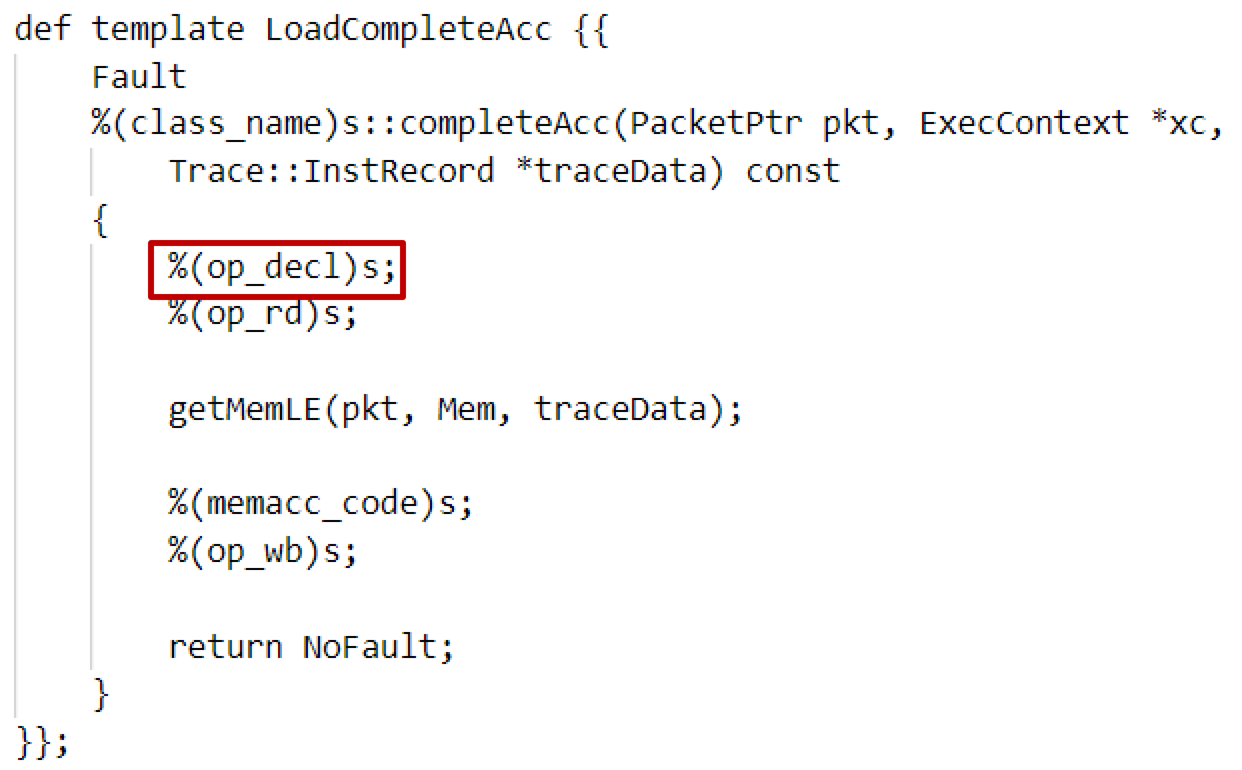
→



```
    Fault
    Lw::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        int64_t Rd = 0;
int32_t Mem = {};
;
        ;

        getMemLE(pkt, Mem, traceData);


                    Rd = Mem;
                ;

        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };

        return NoFault;
    }
```

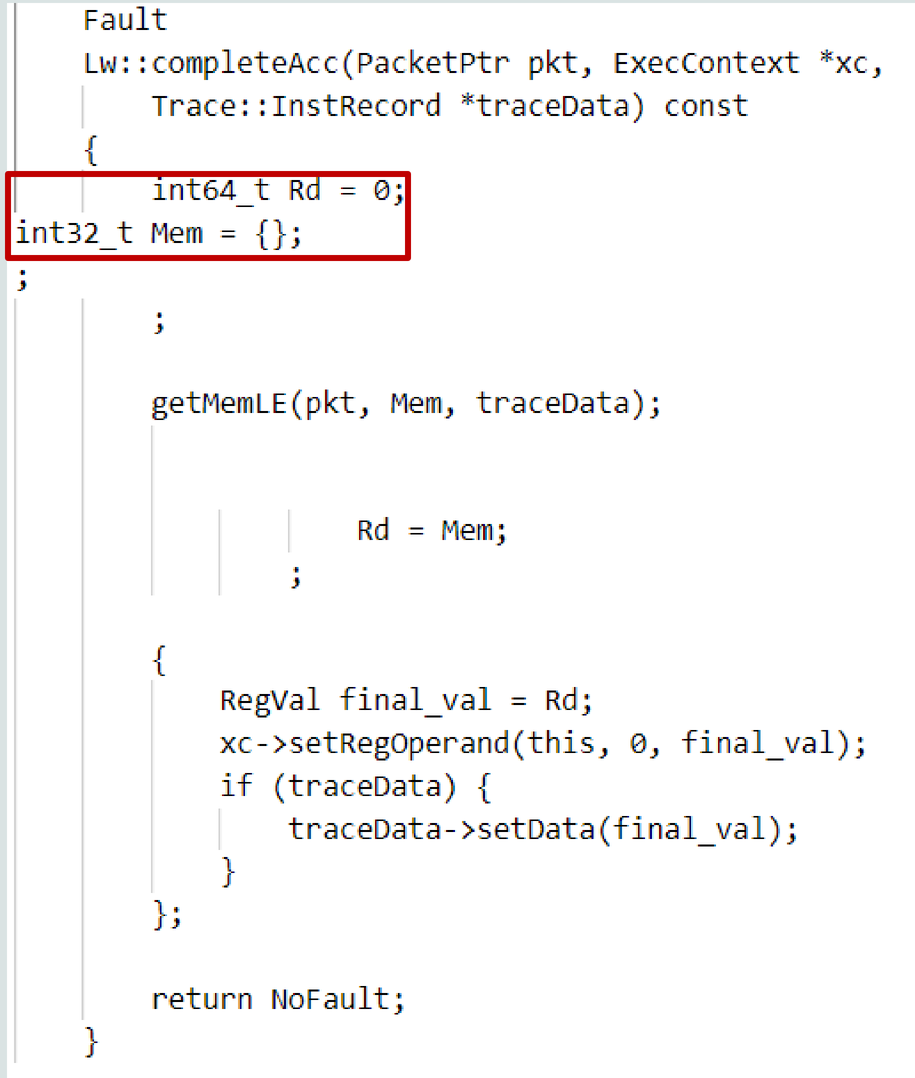
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadCompleteAcc (Timing)



```
def template LoadCompleteAcc {{
    Fault
    %(class_name)s::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        %(op_decl)s;
        %(op_rd)s;

        getMemLE(pkt, Mem, traceData);

        %(memacc_code)s;
        %(op_wb)s;

        return NoFault;
    }
}};
```

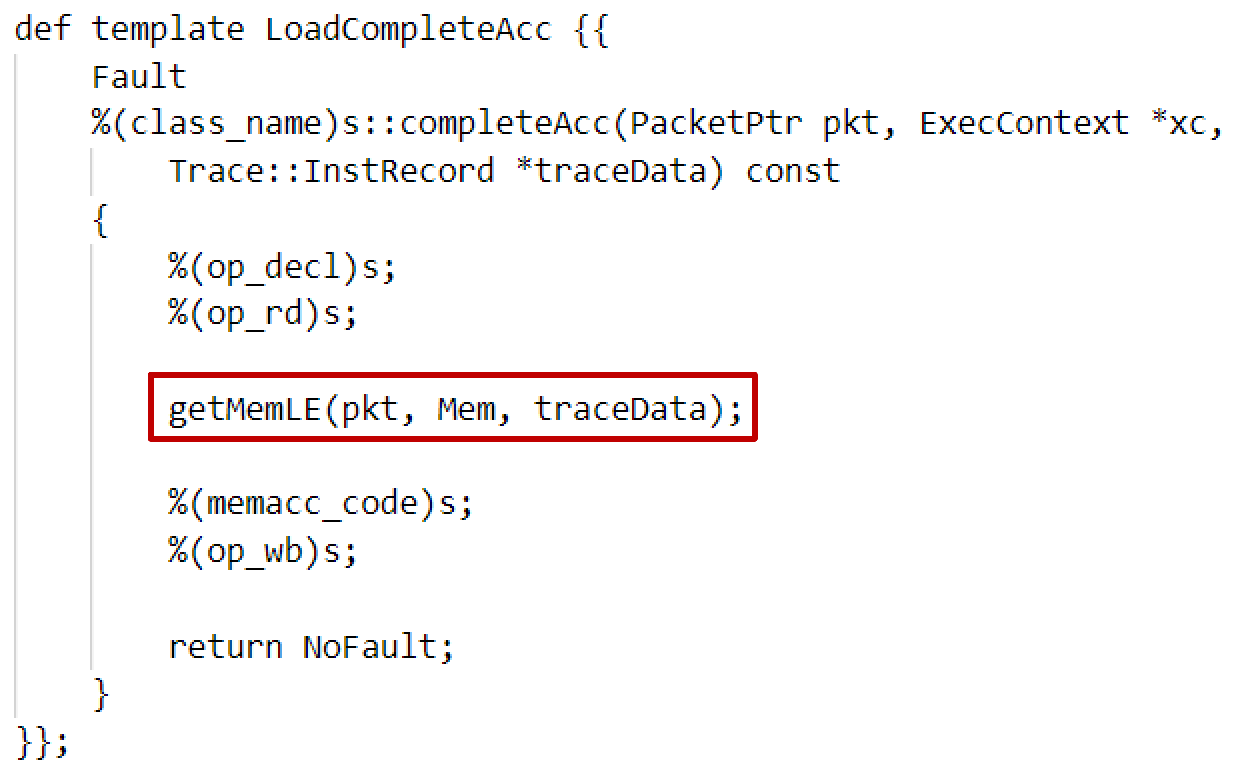
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        int64_t Rd = 0;
int32_t Mem = {};
;
        ;

        getMemLE(pkt, Mem, traceData);


                    Rd = Mem;
                ;

        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };

        return NoFault;
    }
```

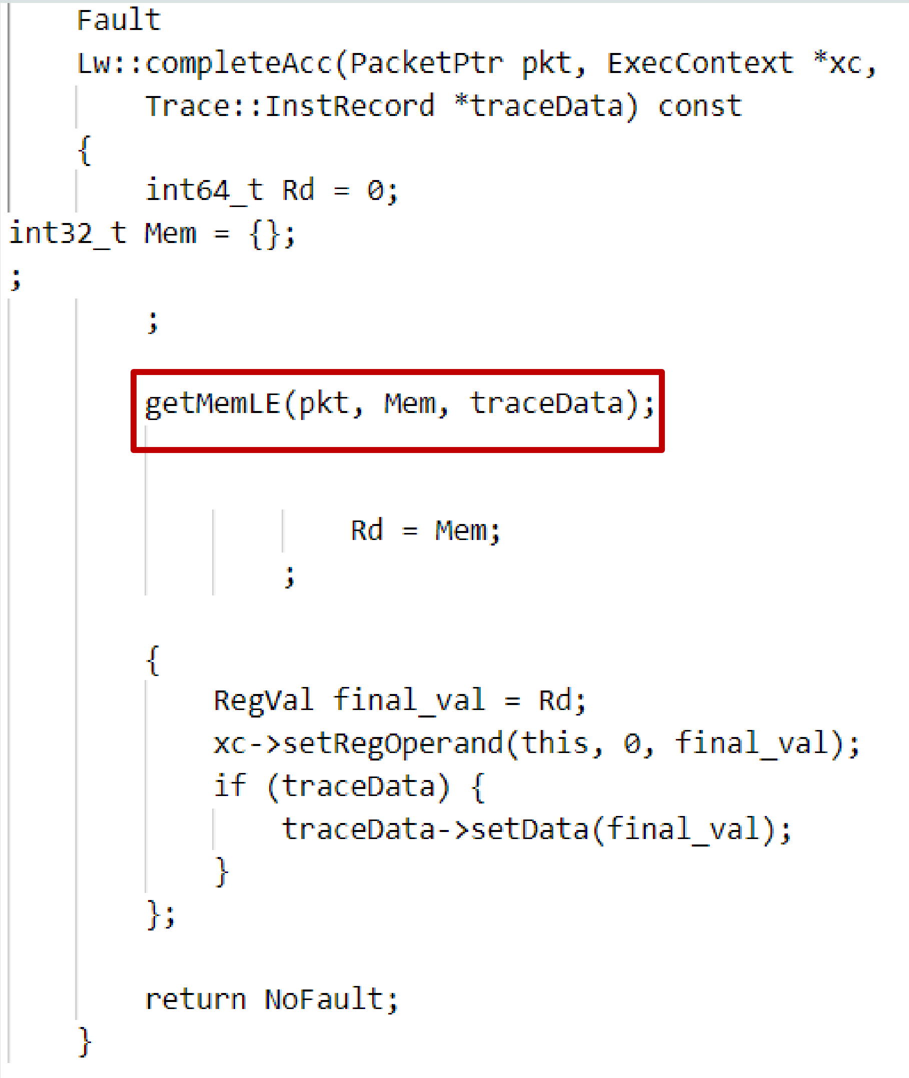
build/RISCV/arch/riscv/generated/exec-ns.cc.inc

LoadCompleteAcc (Timing)



```
def template LoadCompleteAcc {{
    Fault
    %(class_name)s::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        %(op_decl)s;
        %(op_rd)s;

        getMemLE(pkt, Mem, traceData);

        %(memacc_code)s;
        %(op_wb)s;

        return NoFault;
    }
}};
```

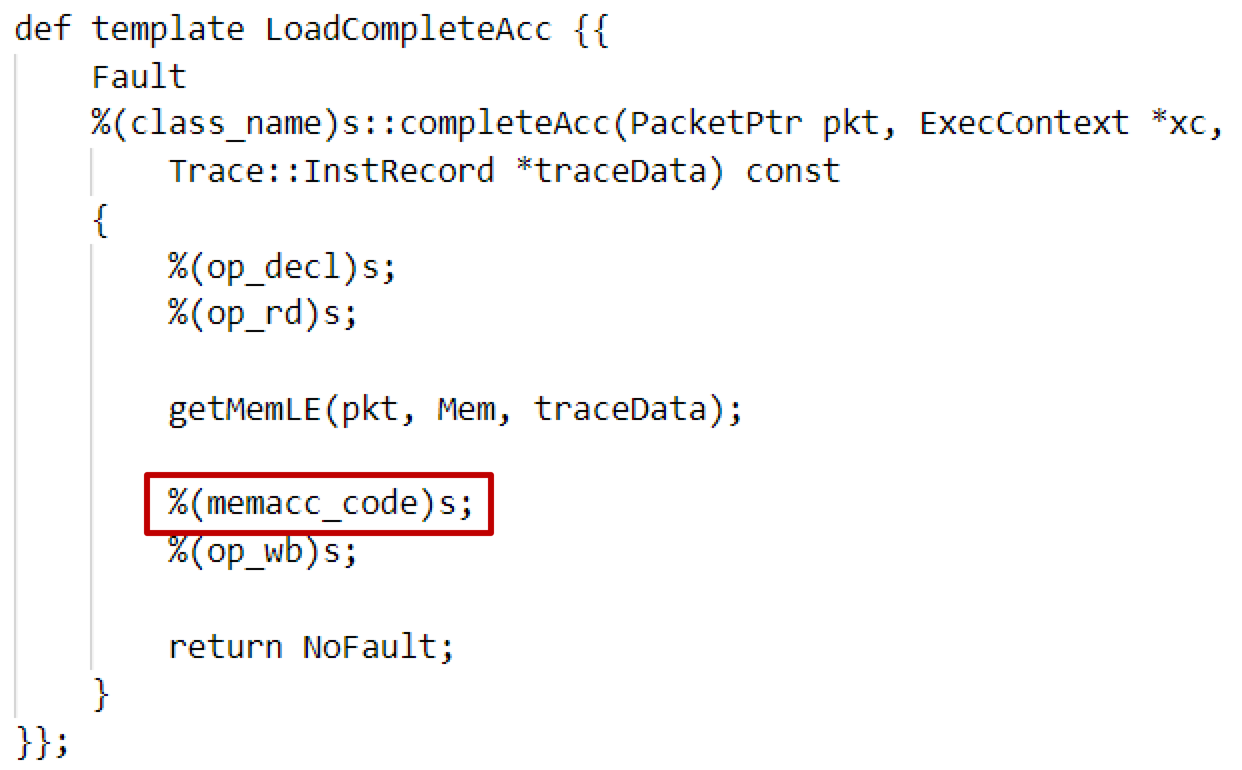
src/arch/riscv/isa/formats/mem.isa



```
    Fault
    Lw::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        int64_t Rd = 0;
int32_t Mem = {};
;
        ;

        getMemLE(pkt, Mem, traceData);


                    Rd = Mem;
                ;

        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };

        return NoFault;
    }
```

build/RISCV/arch/riscv/generated/exec-ns.cc.inc

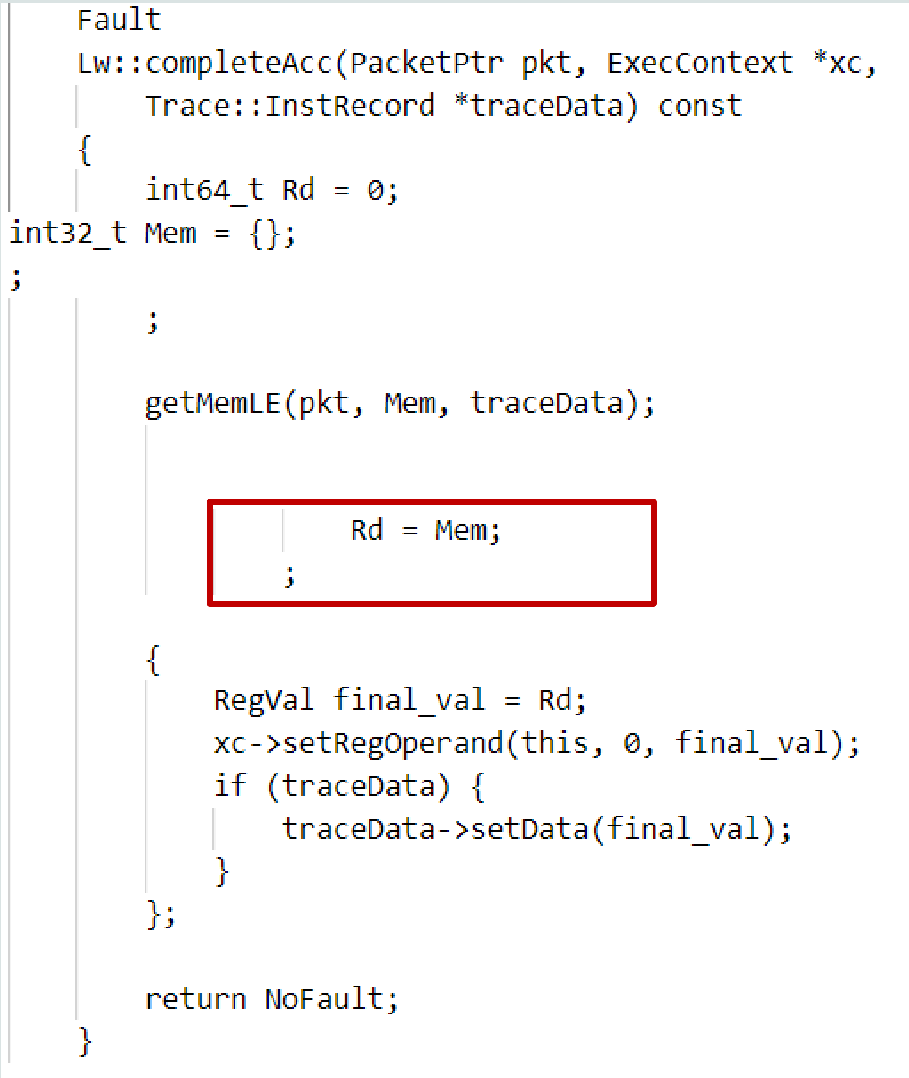
LoadCompleteAcc (Timing)



```
def template LoadCompleteAcc {{
    Fault
    %(class_name)s::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        %(op_decl)s;
        %(op_rd)s;

        getMemLE(pkt, Mem, traceData);

        %(memacc_code)s;
        %(op_wb)s;

        return NoFault;
    }
}};
```

src/arch/riscv/isa/formats/mem.isa

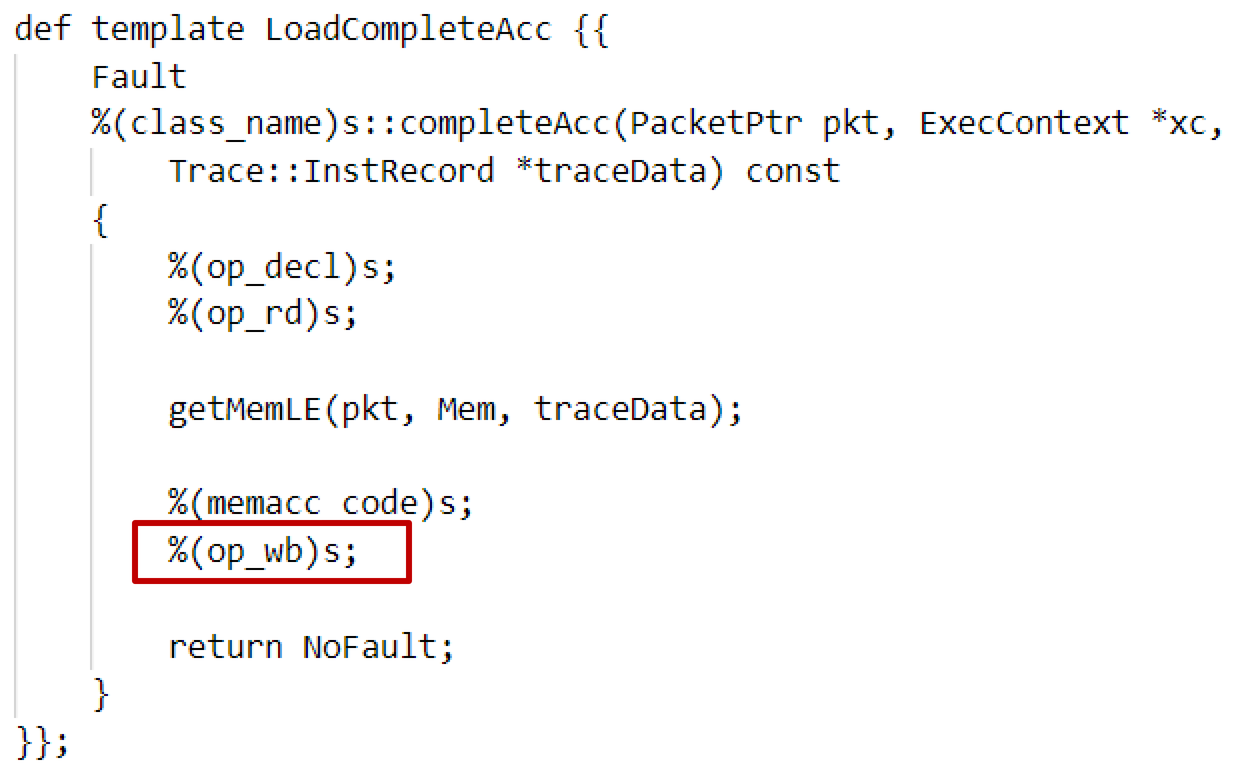
→



```
    Fault
    Lw::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        int64_t Rd = 0;
int32_t Mem = {};
;
        ;

        getMemLE(pkt, Mem, traceData);


                    Rd = Mem;
                ;

        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };

        return NoFault;
    }
```

build/RISCV/arch/riscv/generated/exec-ns.cc.inc

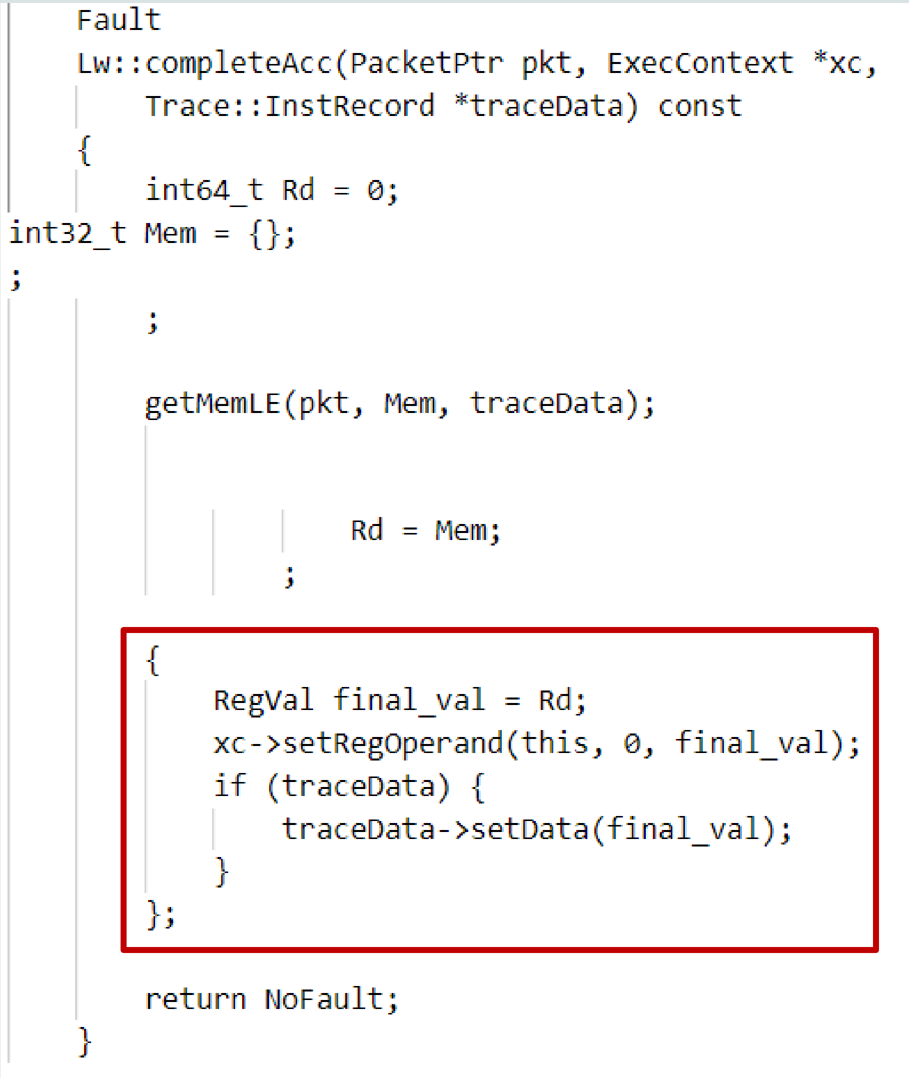
LoadCompleteAcc (Timing)



```
def template LoadCompleteAcc {{
    Fault
    %(class_name)s::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        %(op_decl)s;
        %(op_rd)s;

        getMemLE(pkt, Mem, traceData);

        %(memacc_code)s;
        %(op_wb)s;

        return NoFault;
    }
}};
```

src/arch/riscv/isa/formats/mem.isa

→



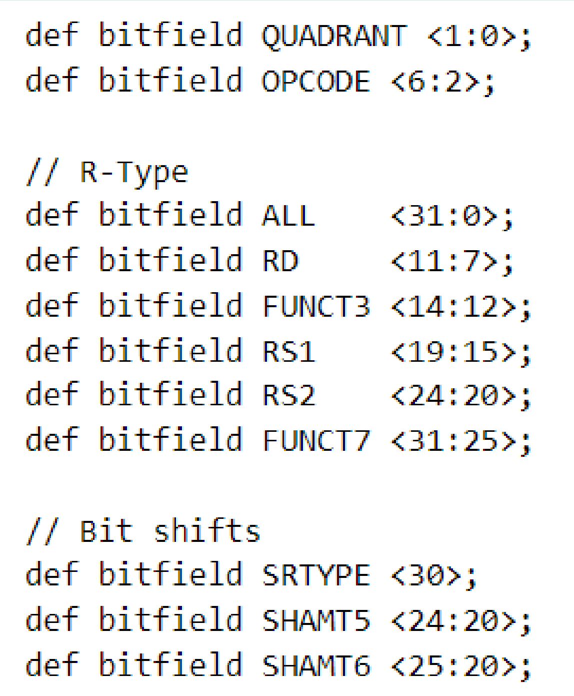
```
    Fault
    Lw::completeAcc(PacketPtr pkt, ExecContext *xc,
        Trace::InstRecord *traceData) const
    {
        int64_t Rd = 0;
int32_t Mem = {};
;
        ;

        getMemLE(pkt, Mem, traceData);


                    Rd = Mem;
                ;

        {
            RegVal final_val = Rd;
            xc->setRegOperand(this, 0, final_val);
            if (traceData) {
                traceData->setData(final_val);
            }
        };

        return NoFault;
    }
```

build/RISCV/arch/riscv/generated/exec-ns.cc.inc

gem5

Bitfield definition

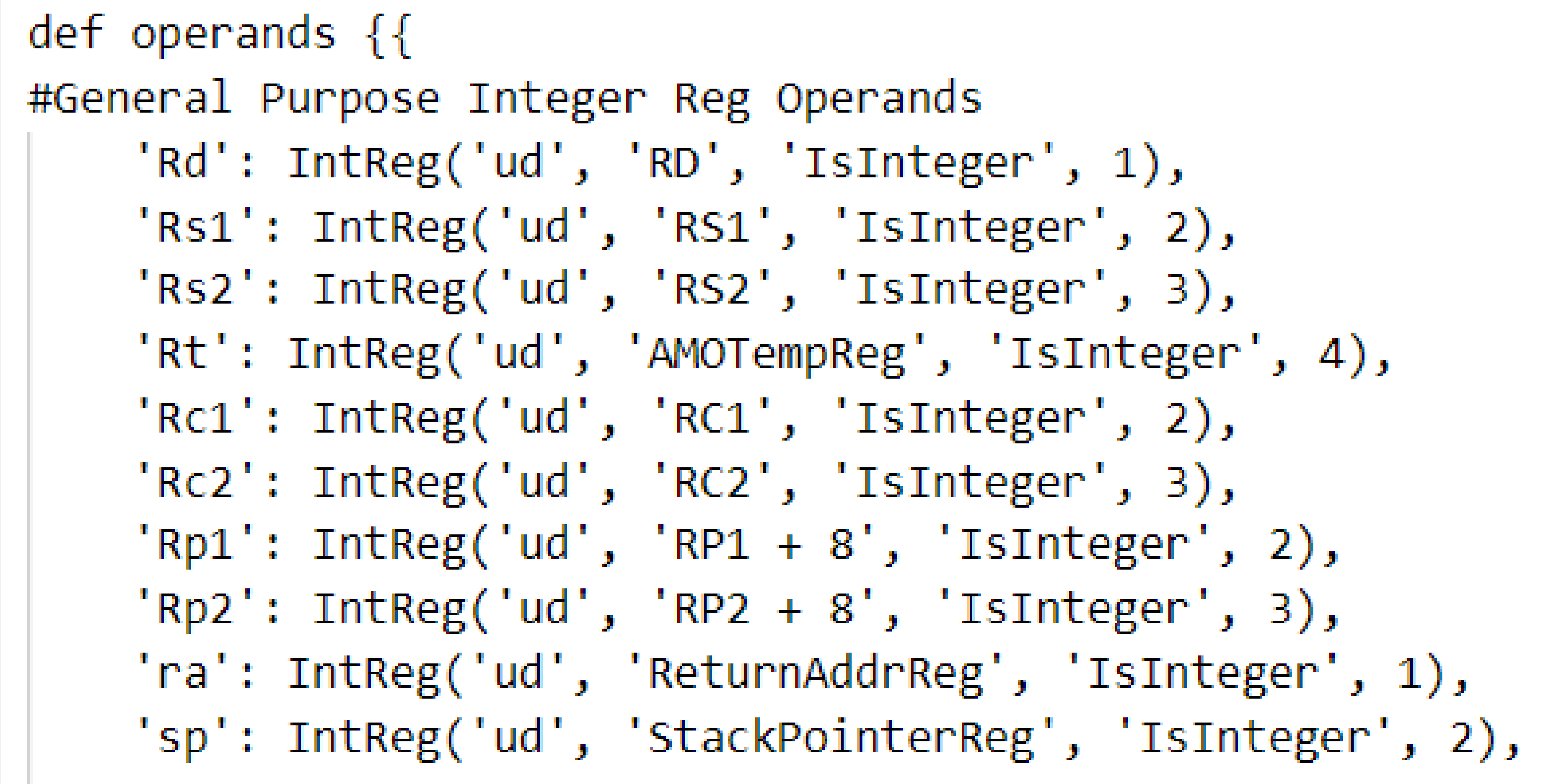
Name for a bitfield within a machine instruction



```
def bitfield QUADRANT <1:0>;
def bitfield OPCODE <6:2>;

// R-Type
def bitfield ALL    <31:0>;
def bitfield RD     <11:7>;
def bitfield FUNCT3 <14:12>;
def bitfield RS1    <19:15>;
def bitfield RS2    <24:20>;
def bitfield FUNCT7 <31:25>;

// Bit shifts
def bitfield SRTYPE <30>;
def bitfield SHAMT5 <24:20>;
def bitfield SHAMT6 <25:20>;
```

src/arch/riscv/isa/bitfields.isa

Instruction Operands

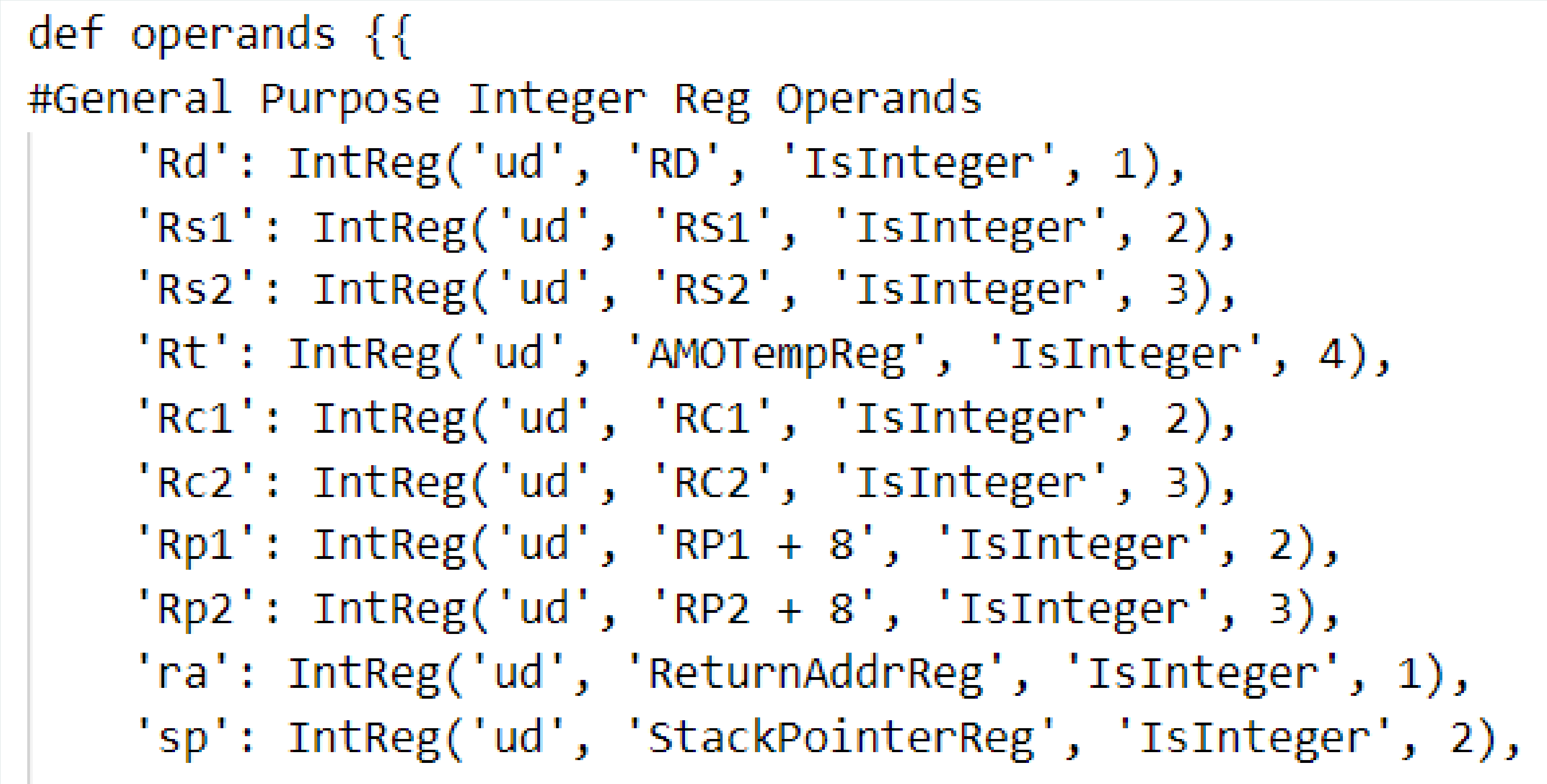


```
def operands {{
#General Purpose Integer Reg Operands
    'Rd': IntReg('ud', 'RD', 'IsInteger', 1),
    'Rs1': IntReg('ud', 'RS1', 'IsInteger', 2),
    'Rs2': IntReg('ud', 'RS2', 'IsInteger', 3),
    'Rt': IntReg('ud', 'AMOTempReg', 'IsInteger', 4),
    'Rc1': IntReg('ud', 'RC1', 'IsInteger', 2),
    'Rc2': IntReg('ud', 'RC2', 'IsInteger', 3),
    'Rp1': IntReg('ud', 'RP1 + 8', 'IsInteger', 2),
    'Rp2': IntReg('ud', 'RP2 + 8', 'IsInteger', 3),
    'ra': IntReg('ud', 'ReturnAddrReg', 'IsInteger', 1),
    'sp': IntReg('ud', 'StackPointerReg', 'IsInteger', 2),
```

src/arch/riscv/isa/operands.isa

Instruction Operands

Maps operands to five element tuples

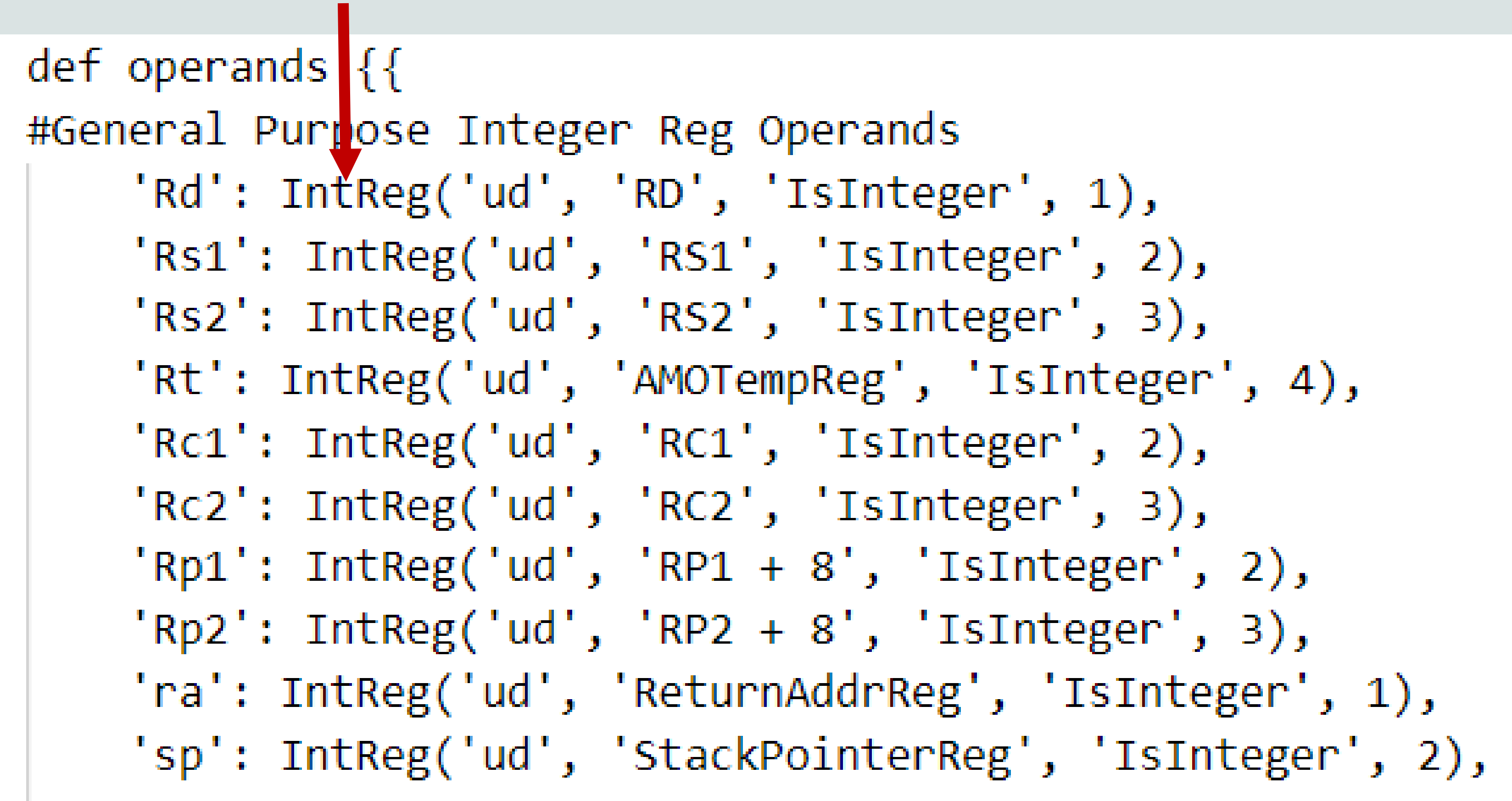


```
def operands {{
#General Purpose Integer Reg Operands
    'Rd': IntReg('ud', 'RD', 'IsInteger', 1),
    'Rs1': IntReg('ud', 'RS1', 'IsInteger', 2),
    'Rs2': IntReg('ud', 'RS2', 'IsInteger', 3),
    'Rt': IntReg('ud', 'AMOTempReg', 'IsInteger', 4),
    'Rc1': IntReg('ud', 'RC1', 'IsInteger', 2),
    'Rc2': IntReg('ud', 'RC2', 'IsInteger', 3),
    'Rp1': IntReg('ud', 'RP1 + 8', 'IsInteger', 2),
    'Rp2': IntReg('ud', 'RP2 + 8', 'IsInteger', 3),
    'ra': IntReg('ud', 'ReturnAddrReg', 'IsInteger', 1),
    'sp': IntReg('ud', 'StackPointerReg', 'IsInteger', 2),
```

src/arch/riscv/isa/operands.isa

Instruction Operands

Operand class

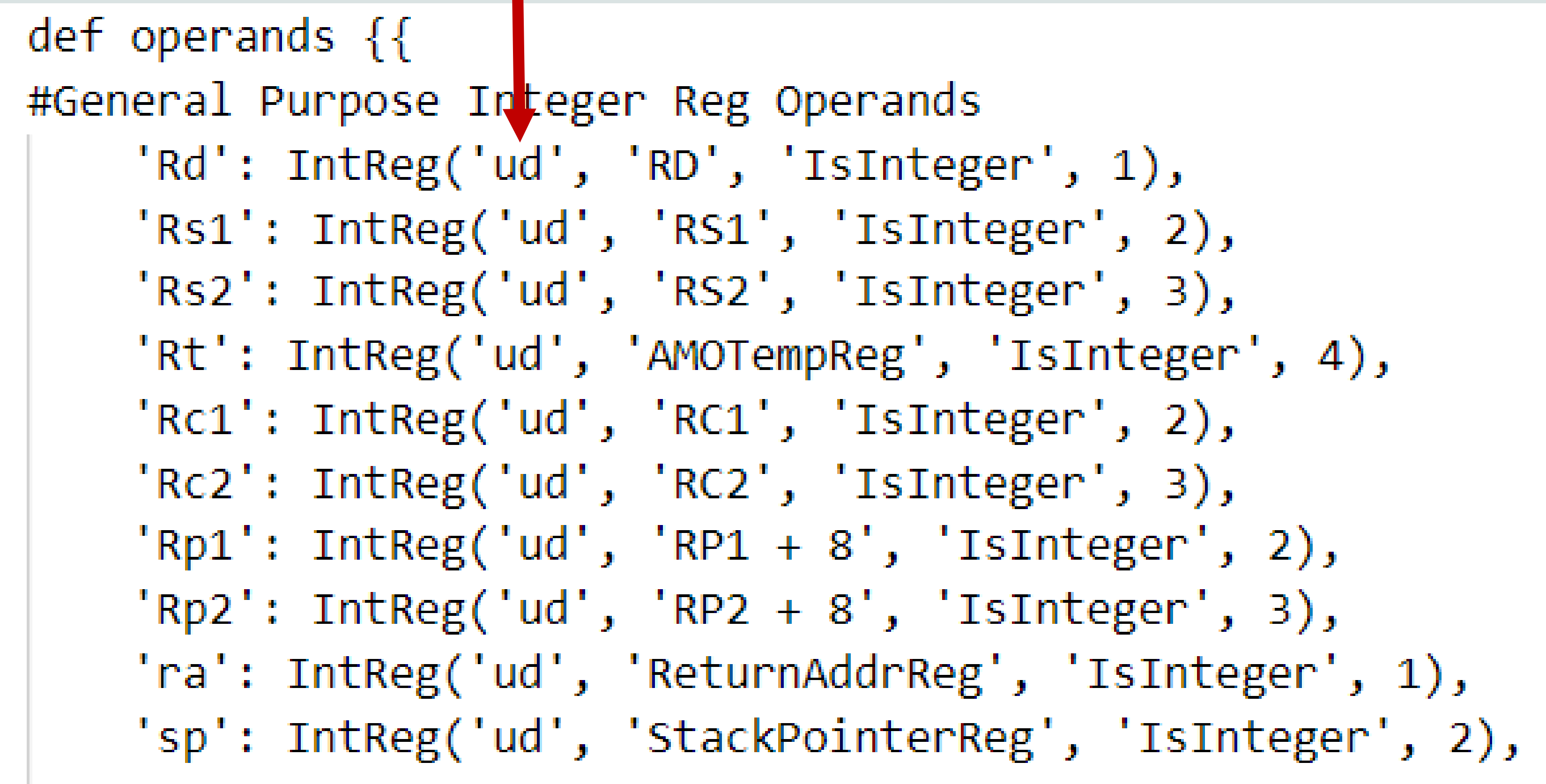


```
def operands {{
#General Purpose Integer Reg Operands
    'Rd': IntReg('ud', 'RD', 'IsInteger', 1),
    'Rs1': IntReg('ud', 'RS1', 'IsInteger', 2),
    'Rs2': IntReg('ud', 'RS2', 'IsInteger', 3),
    'Rt': IntReg('ud', 'AMOTempReg', 'IsInteger', 4),
    'Rc1': IntReg('ud', 'RC1', 'IsInteger', 2),
    'Rc2': IntReg('ud', 'RC2', 'IsInteger', 3),
    'Rp1': IntReg('ud', 'RP1 + 8', 'IsInteger', 2),
    'Rp2': IntReg('ud', 'RP2 + 8', 'IsInteger', 3),
    'ra': IntReg('ud', 'ReturnAddrReg', 'IsInteger', 1),
    'sp': IntReg('ud', 'StackPointerReg', 'IsInteger', 2),
```

src/arch/riscv/isa/operands.isa

Instruction Operands

Default operand type

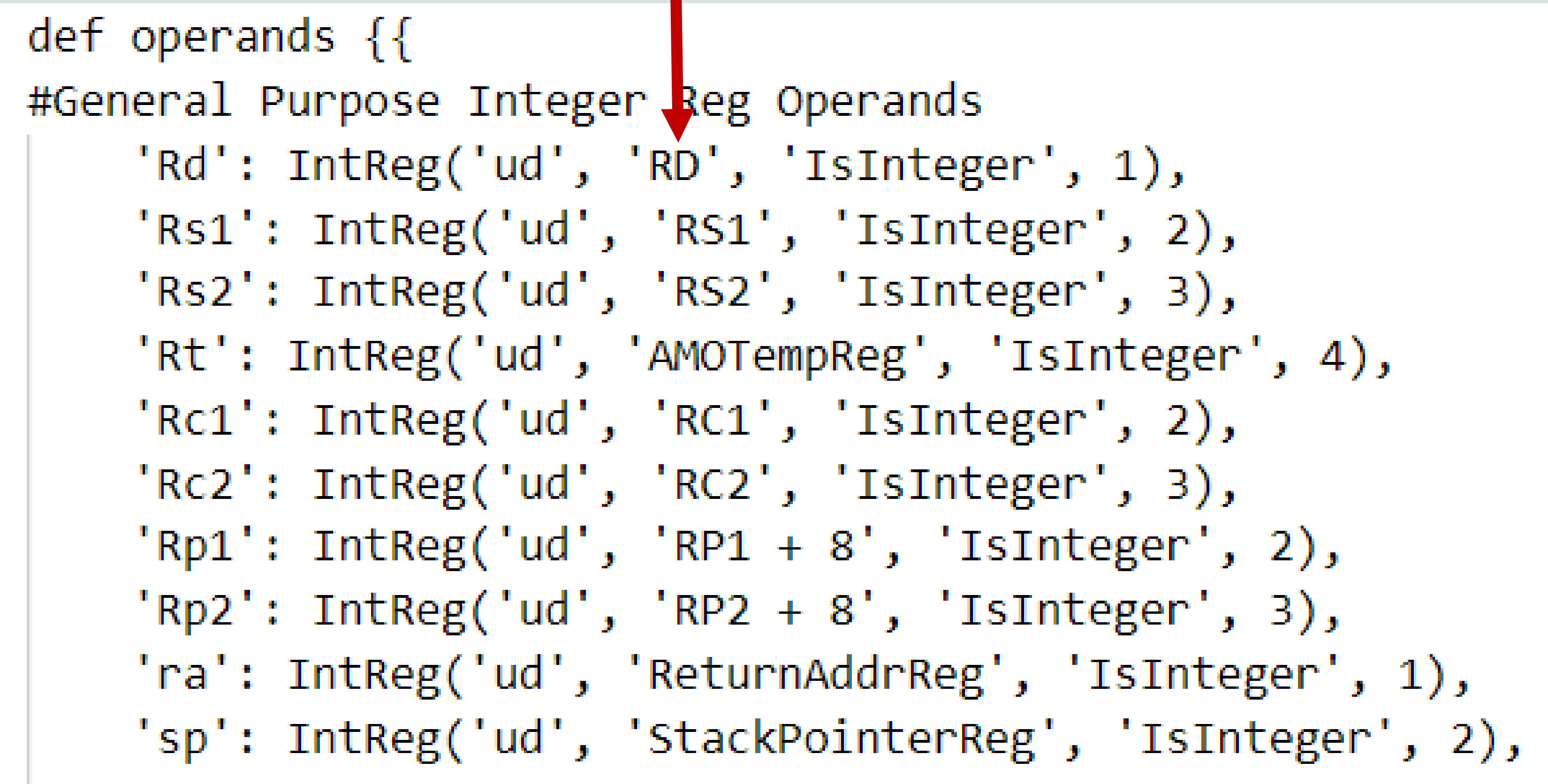


```
def operands {{
#General Purpose Integer Reg Operands
    'Rd': IntReg('ud', 'RD', 'IsInteger', 1),
    'Rs1': IntReg('ud', 'RS1', 'IsInteger', 2),
    'Rs2': IntReg('ud', 'RS2', 'IsInteger', 3),
    'Rt': IntReg('ud', 'AMOTempReg', 'IsInteger', 4),
    'Rc1': IntReg('ud', 'RC1', 'IsInteger', 2),
    'Rc2': IntReg('ud', 'RC2', 'IsInteger', 3),
    'Rp1': IntReg('ud', 'RP1 + 8', 'IsInteger', 2),
    'Rp2': IntReg('ud', 'RP2 + 8', 'IsInteger', 3),
    'ra': IntReg('ud', 'ReturnAddrReg', 'IsInteger', 1),
    'sp': IntReg('ud', 'StackPointerReg', 'IsInteger', 2),
```

src/arch/riscv/isa/operands.isa

Instruction Operands

Bitfield name (how to specify specific instance)

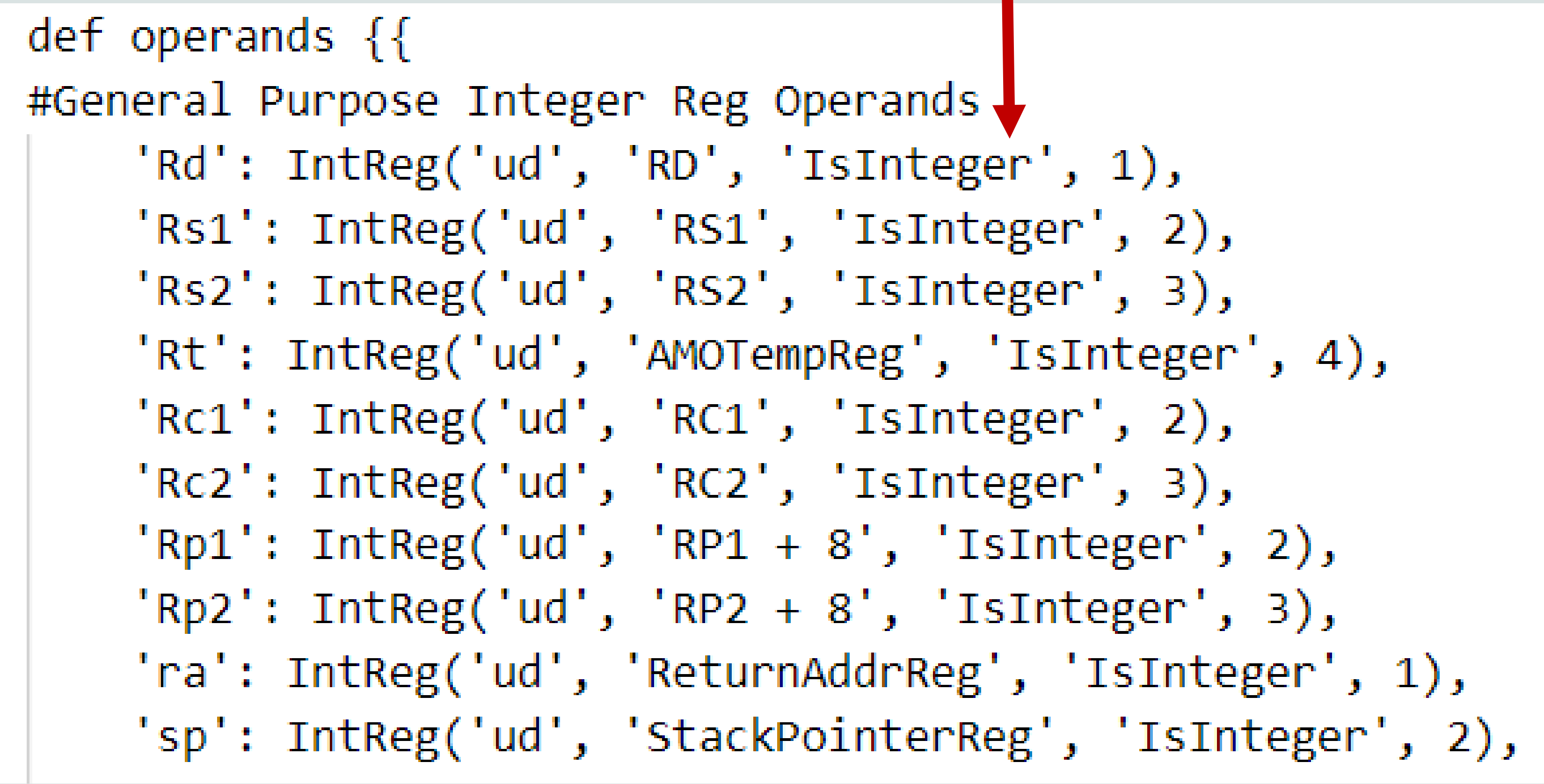


```
def operands {{
#General Purpose Integer Reg Operands
    'Rd': IntReg('ud', 'RD', 'IsInteger', 1),
    'Rs1': IntReg('ud', 'RS1', 'IsInteger', 2),
    'Rs2': IntReg('ud', 'RS2', 'IsInteger', 3),
    'Rt': IntReg('ud', 'AMOTempReg', 'IsInteger', 4),
    'Rc1': IntReg('ud', 'RC1', 'IsInteger', 2),
    'Rc2': IntReg('ud', 'RC2', 'IsInteger', 3),
    'Rp1': IntReg('ud', 'RP1 + 8', 'IsInteger', 2),
    'Rp2': IntReg('ud', 'RP2 + 8', 'IsInteger', 3),
    'ra': IntReg('ud', 'ReturnAddrReg', 'IsInteger', 1),
    'sp': IntReg('ud', 'StackPointerReg', 'IsInteger', 2),
```

src/arch/riscv/isa/operands.isa

Instruction Operands

Instruction Flag (string or a tripe of strings)



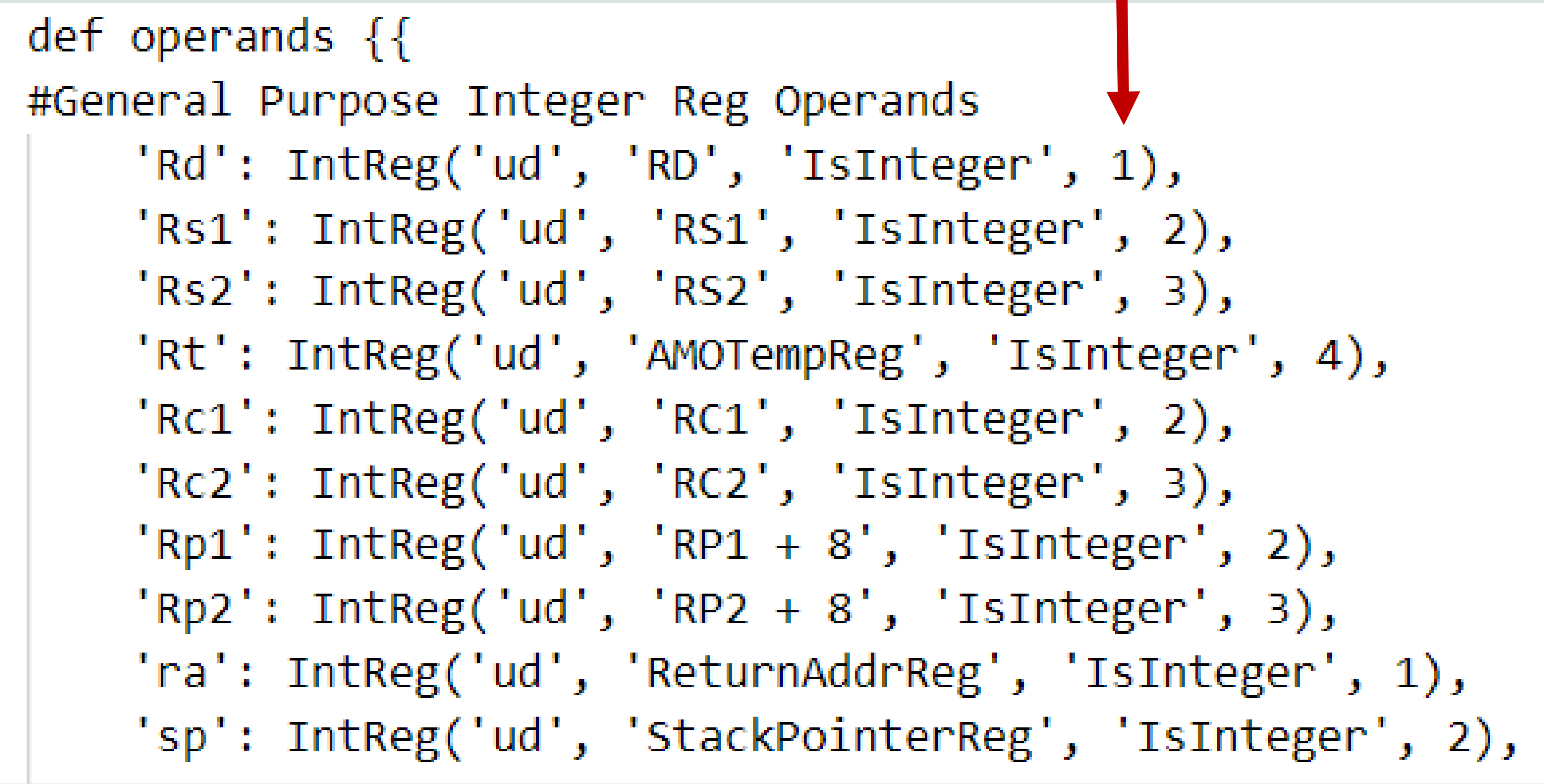
```
def operands {{
#General Purpose Integer Reg Operands
    'Rd': IntReg('ud', 'RD', 'IsInteger', 1),
    'Rs1': IntReg('ud', 'RS1', 'IsInteger', 2),
    'Rs2': IntReg('ud', 'RS2', 'IsInteger', 3),
    'Rt': IntReg('ud', 'AMOTempReg', 'IsInteger', 4),
    'Rc1': IntReg('ud', 'RC1', 'IsInteger', 2),
    'Rc2': IntReg('ud', 'RC2', 'IsInteger', 3),
    'Rp1': IntReg('ud', 'RP1 + 8', 'IsInteger', 2),
    'Rp2': IntReg('ud', 'RP2 + 8', 'IsInteger', 3),
    'ra': IntReg('ud', 'ReturnAddrReg', 'IsInteger', 1),
    'sp': IntReg('ud', 'StackPointerReg', 'IsInteger', 2),
```

src/arch/riscv/isa/operands.isa

gem5

Instruction Operands

Order of operand in disassembly

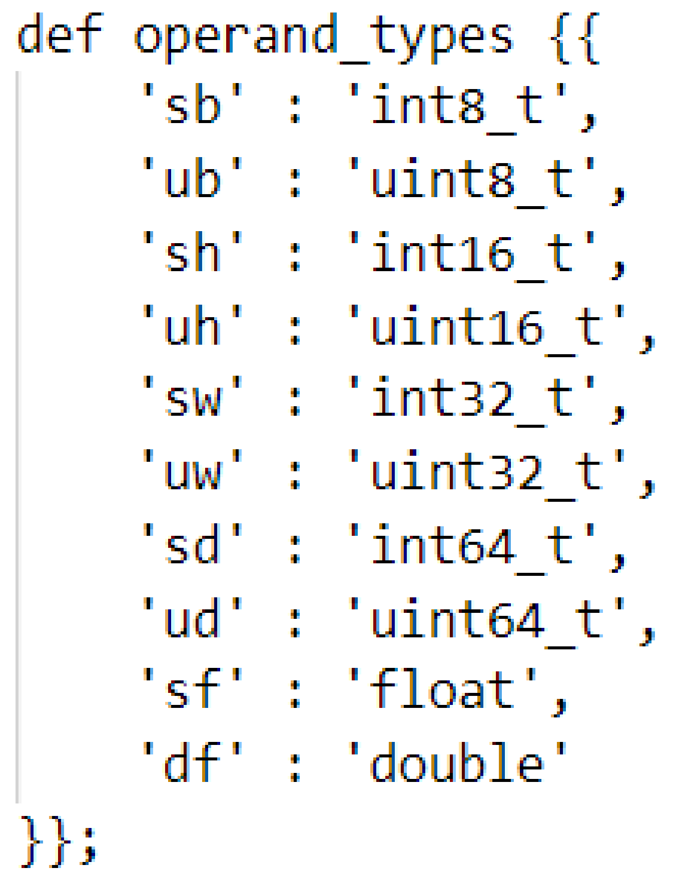


```
def operands {{
#General Purpose Integer Reg Operands
    'Rd': IntReg('ud', 'RD', 'IsInteger', 1),
    'Rs1': IntReg('ud', 'RS1', 'IsInteger', 2),
    'Rs2': IntReg('ud', 'RS2', 'IsInteger', 3),
    'Rt': IntReg('ud', 'AMOTempReg', 'IsInteger', 4),
    'Rc1': IntReg('ud', 'RC1', 'IsInteger', 2),
    'Rc2': IntReg('ud', 'RC2', 'IsInteger', 3),
    'Rp1': IntReg('ud', 'RP1 + 8', 'IsInteger', 2),
    'Rp2': IntReg('ud', 'RP2 + 8', 'IsInteger', 3),
    'ra': IntReg('ud', 'ReturnAddrReg', 'IsInteger', 1),
    'sp': IntReg('ud', 'StackPointerReg', 'IsInteger', 2),
```

src/arch/riscv/isa/operands.isa

Operand type qualifiers

Type qualifier can be appended to the instruction operand



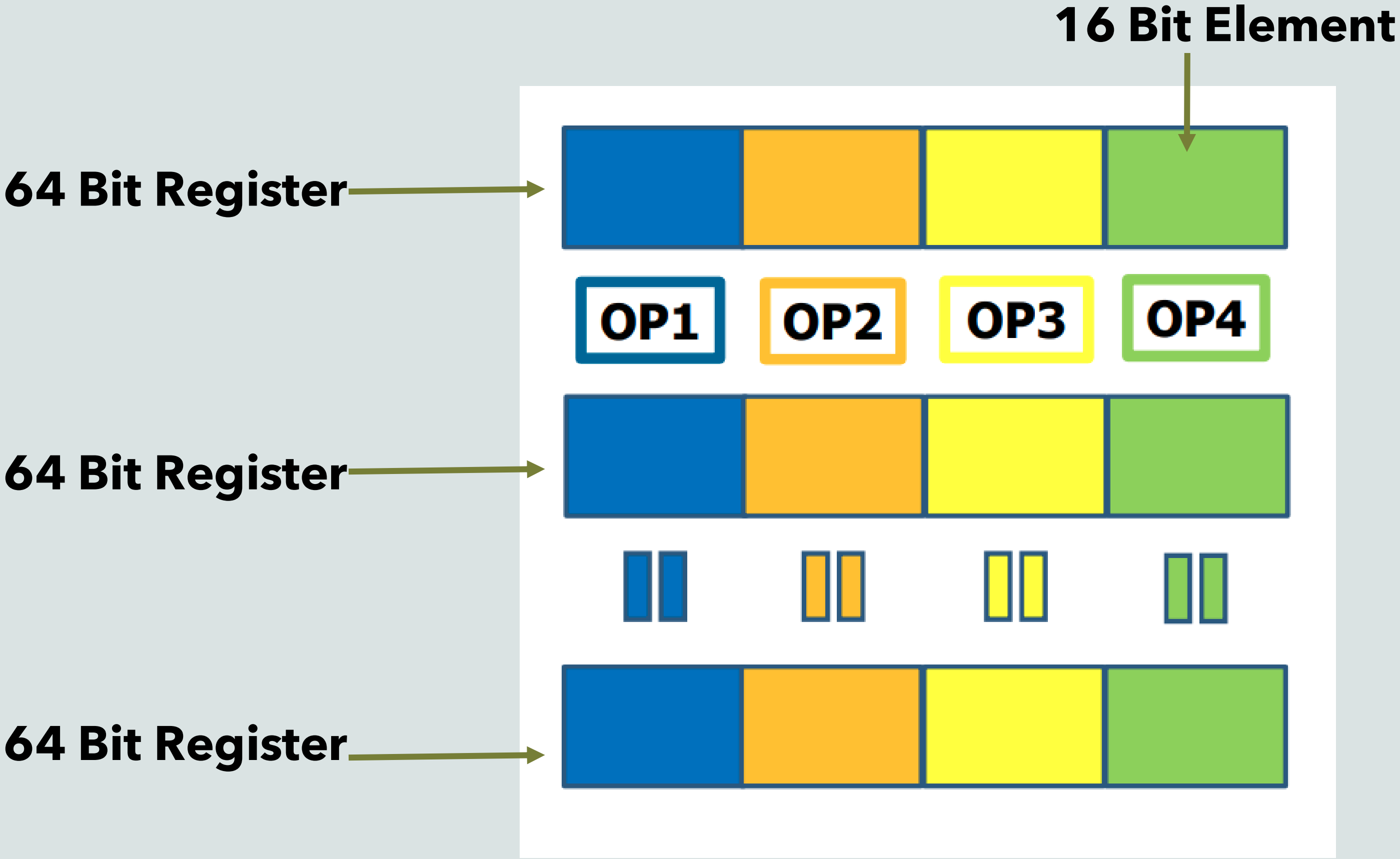
```
def operand_types {{
    'sb' : 'int8_t',
    'ub' : 'uint8_t',
    'sh' : 'int16_t',
    'uh' : 'uint16_t',
    'sw' : 'int32_t',
    'uw' : 'uint32_t',
    'sd' : 'int64_t',
    'ud' : 'uint64_t',
    'sf' : 'float',
    'df' : 'double'
}};
```

src/arch/riscv/isa/operands.isa

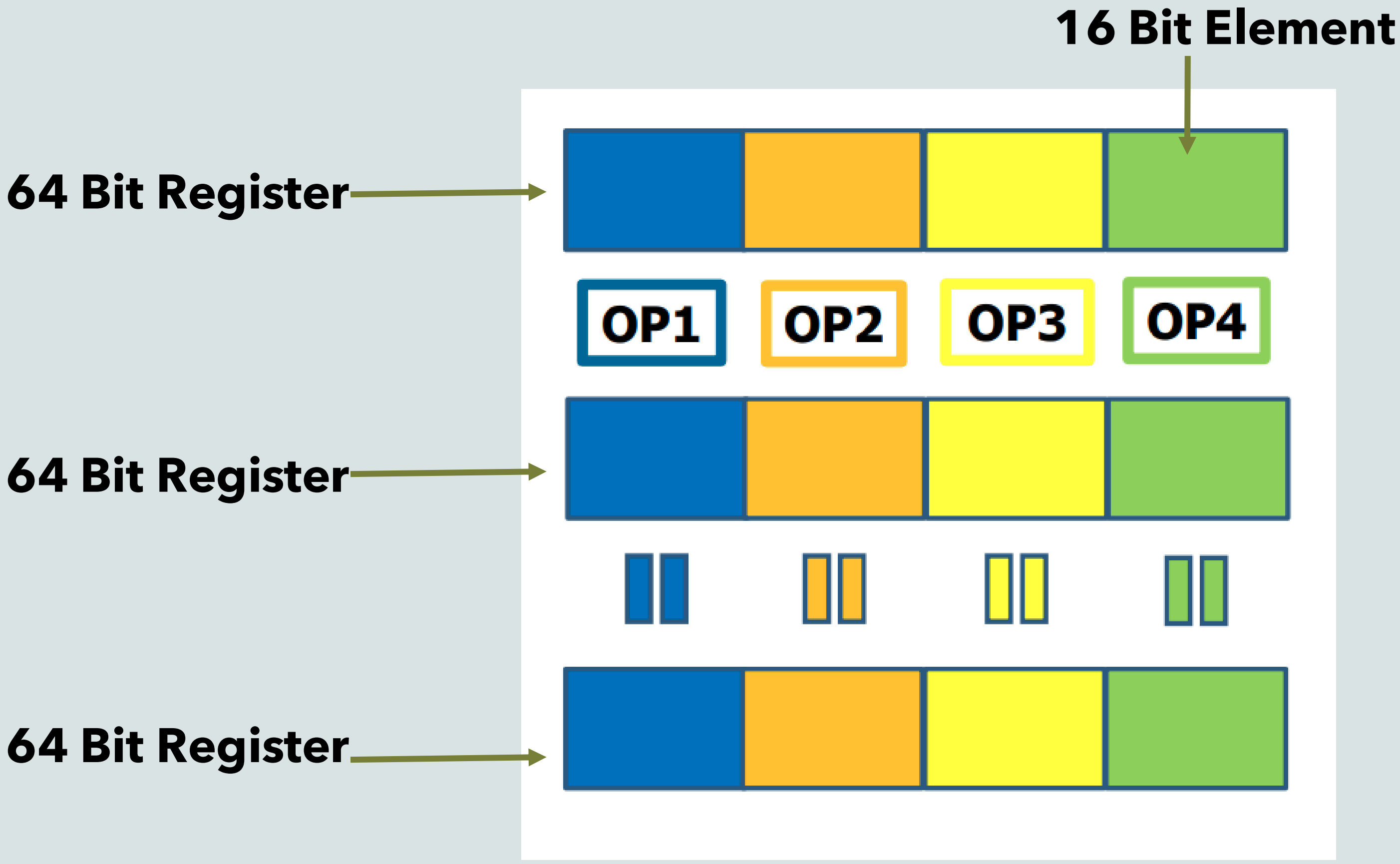
Adding new Instructions

RISC-V Packed SIMD

RISC-V Packed SIMD

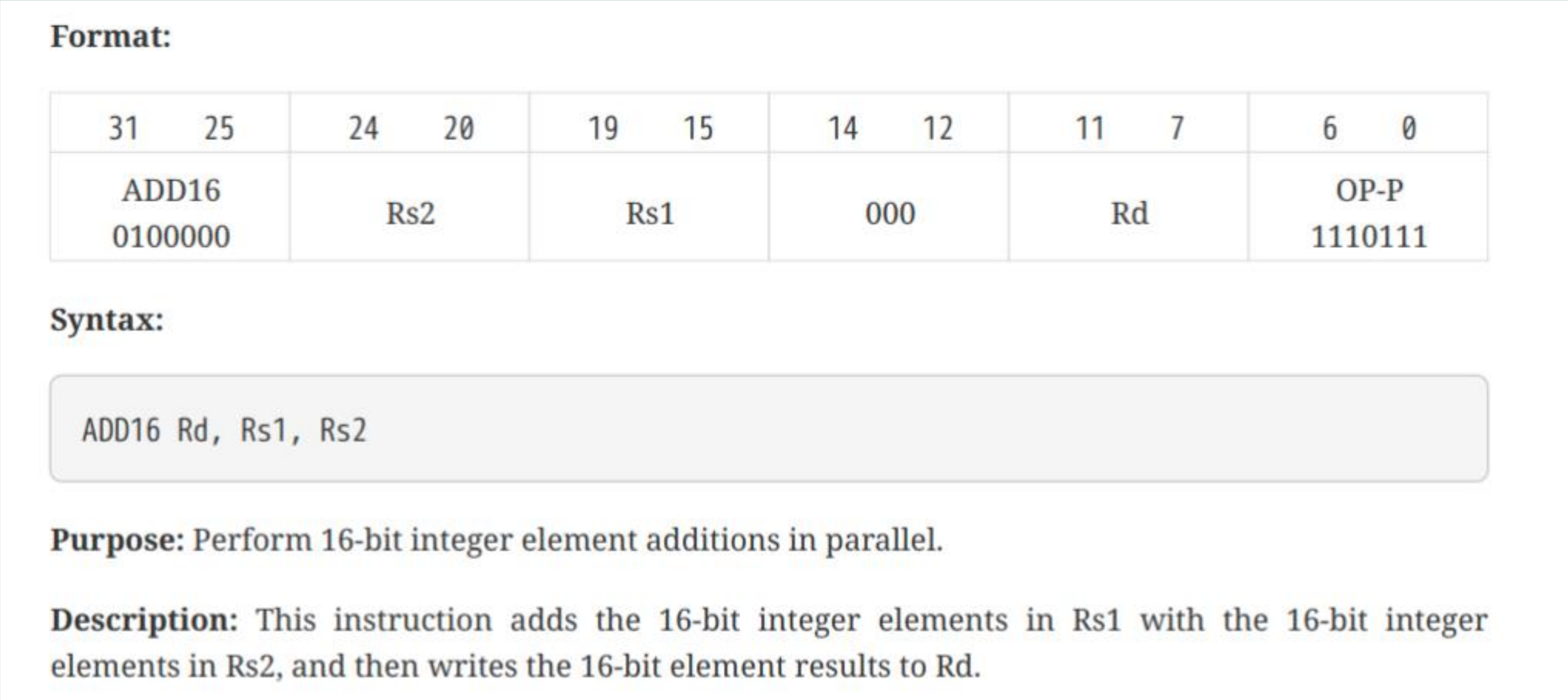


Let's implement an instruction "ADD16"



gem5

ADD16 Encoding from Reference Manual



Format:

31 25	24 20	19 15	14 12	11 7	6 0
ADD16 0100000	Rs2	Rs1	000	Rd	OP-P 1110111

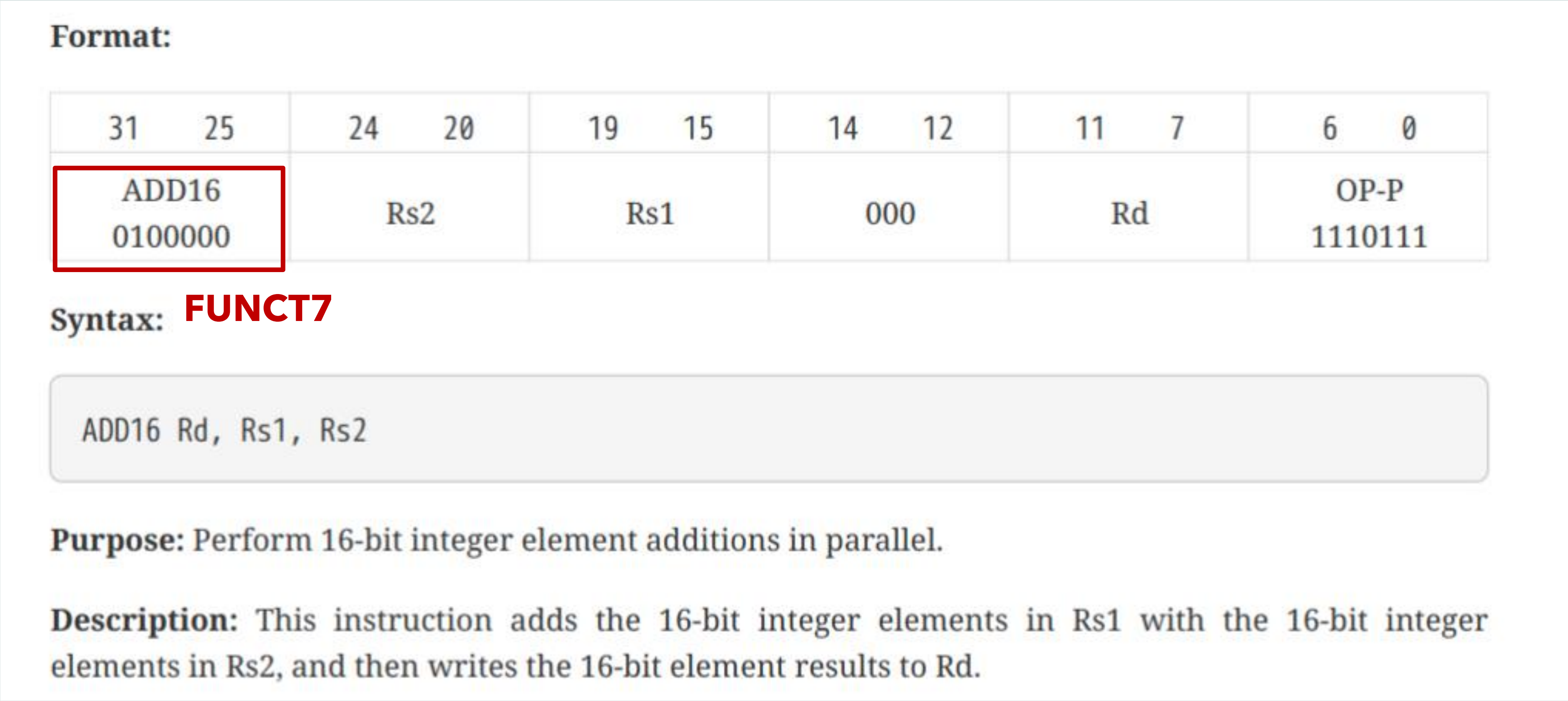
Syntax:

```
ADD16 Rd, Rs1, Rs2
```

Purpose: Perform 16-bit integer element additions in parallel.

Description: This instruction adds the 16-bit integer elements in Rs1 with the 16-bit integer elements in Rs2, and then writes the 16-bit element results to Rd.

ADD16 Encoding from Reference Manual



Format:

31 25	24 20	19 15	14 12	11 7	6 0
ADD16 0100000	Rs2	Rs1	000	Rd	OP-P 1110111

FUNCT7

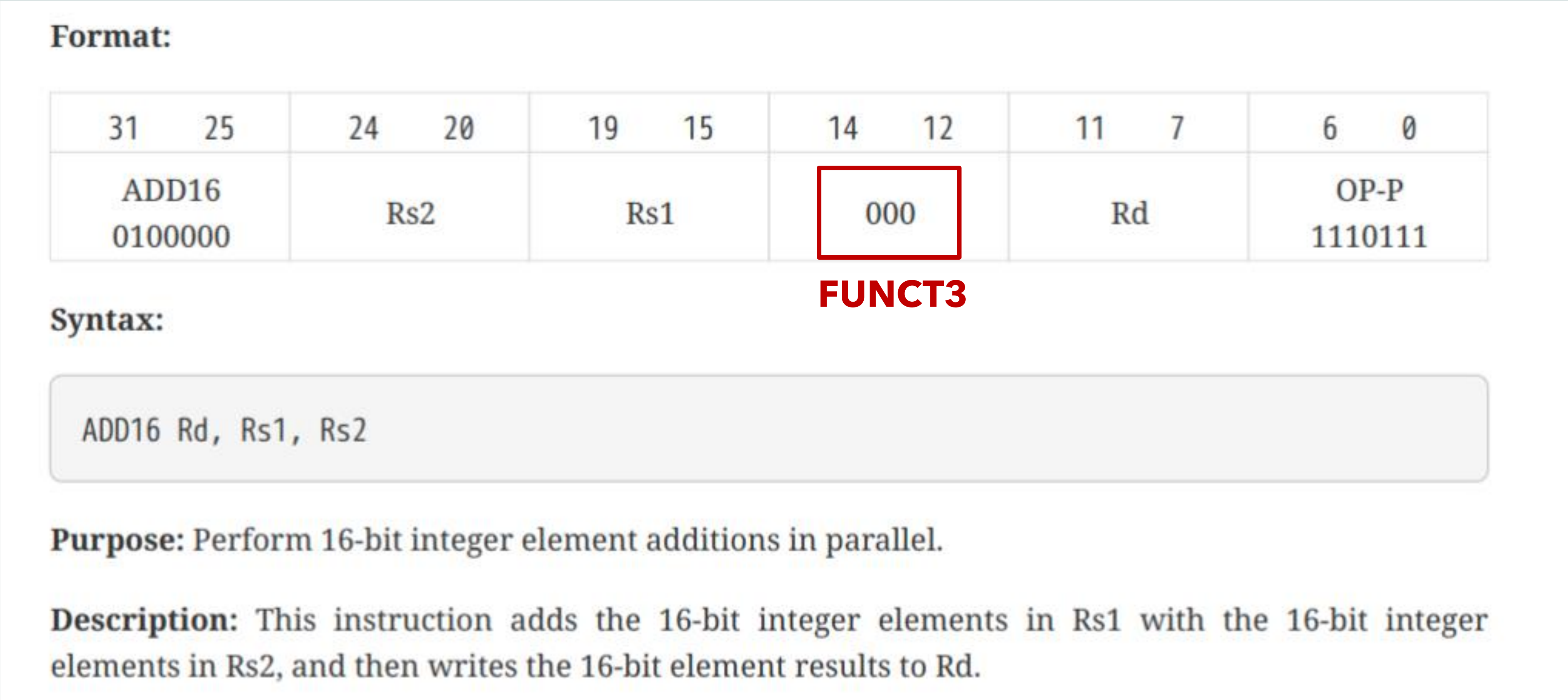
Syntax:

```
ADD16 Rd, Rs1, Rs2
```

Purpose: Perform 16-bit integer element additions in parallel.

Description: This instruction adds the 16-bit integer elements in Rs1 with the 16-bit integer elements in Rs2, and then writes the 16-bit element results to Rd.

ADD16 Encoding from Reference Manual



Format:

31 25	24 20	19 15	14 12	11 7	6 0
ADD16 0100000	Rs2	Rs1	000	Rd	OP-P 1110111

FUNCT3

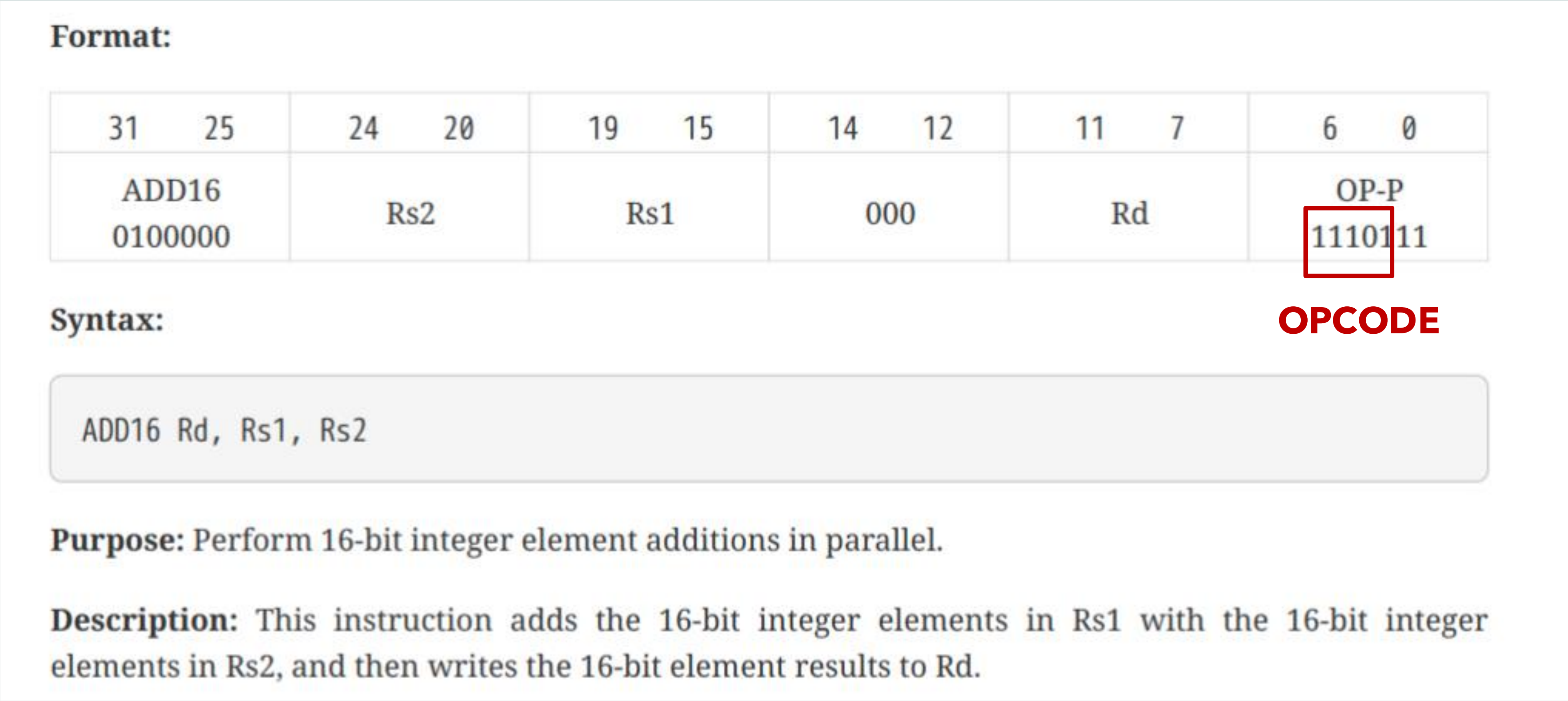
Syntax:

```
ADD16 Rd, Rs1, Rs2
```

Purpose: Perform 16-bit integer element additions in parallel.

Description: This instruction adds the 16-bit integer elements in Rs1 with the 16-bit integer elements in Rs2, and then writes the 16-bit element results to Rd.

ADD16 Encoding from Reference Manual



Format:

31 25	24 20	19 15	14 12	11 7	6 0
ADD16 0100000	Rs2	Rs1	000	Rd	OP-P 1110111

OPCODE

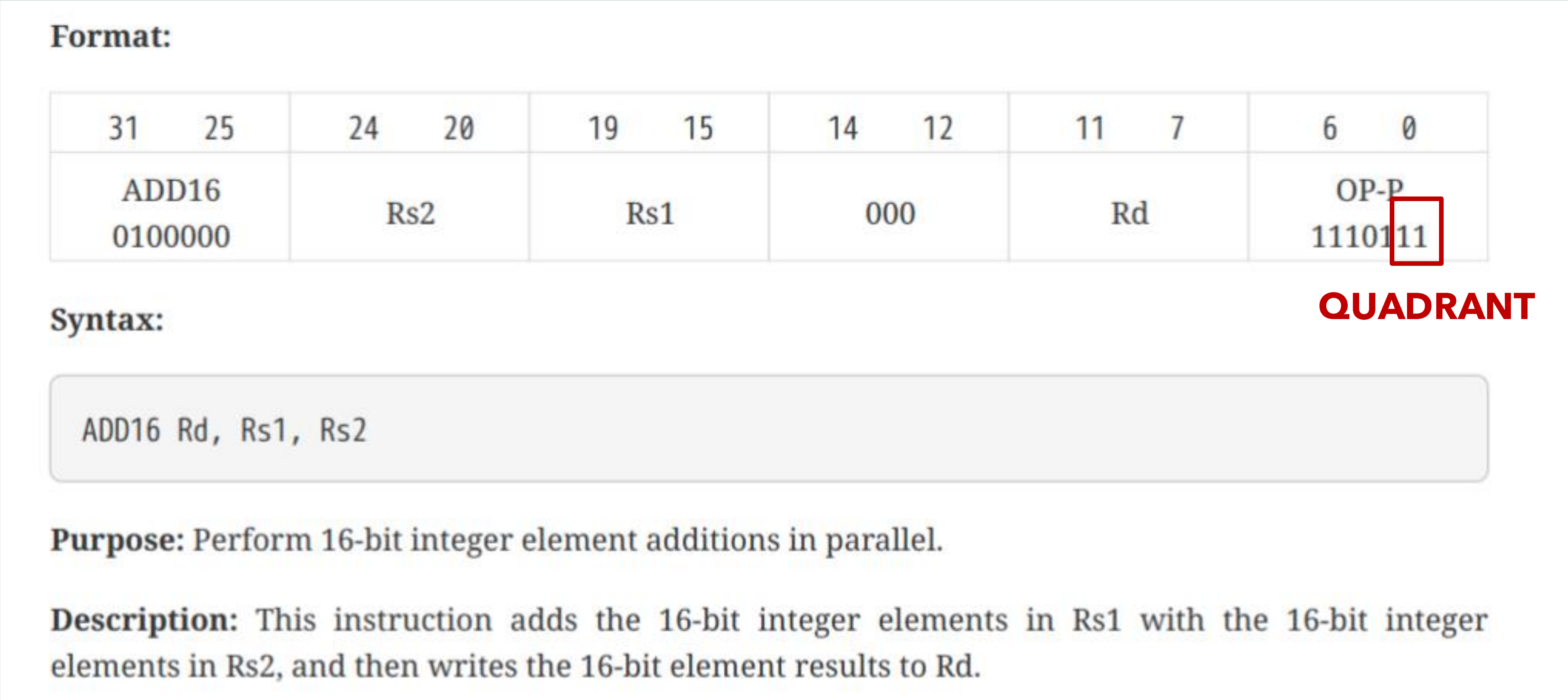
Syntax:

```
ADD16 Rd, Rs1, Rs2
```

Purpose: Perform 16-bit integer element additions in parallel.

Description: This instruction adds the 16-bit integer elements in Rs1 with the 16-bit integer elements in Rs2, and then writes the 16-bit element results to Rd.

ADD16 Encoding from Reference Manual



Format:

31 25	24 20	19 15	14 12	11 7	6 0
ADD16 0100000	Rs2	Rs1	000	Rd	OP-P 1110111

QUADRANT

Syntax:

```
ADD16 Rd, Rs1, Rs2
```

Purpose: Perform 16-bit integer element additions in parallel.

Description: This instruction adds the 16-bit integer elements in Rs1 with the 16-bit integer elements in Rs2, and then writes the 16-bit element results to Rd.

Exercise

- Implement the instruction and compile gem5
- Test binary path : **developing-gem5-models/06-cpu-instructions/tests/add16_test**
- Disassemble the provided test binary
- Run gem5 (using ATOMIC CPU) – test should pass!
- Create execution trace from gem5
- Look at the trace and see the ADD16 instruction.

Add another instruction – do all by yourself!

- We will add `sra16` instruction

- Specs of the instruction can be found in https://github.com/riscv/riscv-p-spec/blob/master/P-ext-proposal.pdf

Exercise

- Implement the new instruction all by yourself
- Disassemble the provided test binary
- Compile and run gem5 – test should pass!
- Create execution trace from gem5
- Look at the trace and see your instruction.
- We can also put some debug points in the code and go through them when the instruction executes

RISC-V Assembler

How to create the test binaries we used?

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/opcodes/riscv-opc.c

Add new instruction definition

```
{"add16",0,INSN_CLASS_I,"d,s,t",MATCH_ADD16,MASK_ADD16,match_opcode,0}
```

name

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/opcodes/riscv-opc.c

Add new instruction definition

```
{"add16",0,INSN_CLASS_I,"d,s,t",MATCH_ADD16,MASK_ADD16,match_opcode,0}
```

Target arch., 0 means both 32 and 64 bit

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/include/opcode/riscv-opc.h

Add new instruction definition

```
{"add16",0,INSN_CLASS_I,"d,s,t",MATCH_ADD16,MASK_ADD16,match_opcode,0}
```

Instruction class (integer)

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/opcodes/riscv-opc.c

Add new instruction definition

```
{"add16",0,INSN_CLASS_I,"d,s,t",MATCH_ADD16,MASK_ADD16,match_opcode,0}
```

Specifies operands

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/opcodes/riscv-opc.c

Add new instruction definition

```
{"add16",0,INSN_CLASS_I,"d,s,t",MATCH_ADD16,MASK_ADD16,match_opcode,0}
```

Instruction opcode

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/opcodes/riscv-opc.c

Add new instruction definition

```
{"add16",0,INSN_CLASS_I,"d,s,t",MATCH_ADD16,MASK_ADD16,match_opcode,0}
```

Specifies position of operands

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/opcodes/riscv-opc.c

Add new instruction definition

```
{"add16",0,INSN_CLASS_I,"d,s,t",MATCH_ADD16,MASK_ADD16,match_opcode,0}
```

Specifies position of operands

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/include/opcode/riscv-opc.h

Add new instruction definition

```
{"add16",0,INSN_CLASS_I,"d,s,t",MATCH_ADD16,MASK_ADD16,match_opcode,0}
```

Specifies position of operands

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/include/opcode/riscv-opc.h

Add the match and mask codes for the instruction

```
#define MATCH_ADD16 0x40000077
#define MASK_ADD16 0xfe00707f

DECLARE_INSN(add16, MATCH_ADD16,MASK_ADD16)
```

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/include/opcode/riscv-opc.h

Add the match and mask codes for the instruction

```
#define MATCH_ADD16 0x40000077
#define MASK_ADD16 0xfe00707f

DECLARE_INSN(add16, MATCH_ADD16,MASK_ADD16)
```

Instruction Opcode – replace operands with 0's

Modifications in RISC-V GNU Toolchain

- Modify the GNU assembler

riscv-binutils/include/opcode/riscv-opc.h

Add the match and mask codes for the instruction

```
#define MATCH_ADD16 0x40000077
#define MASK_ADD16 0xfe00707f

DECLARE_INSN(add16, MATCH_ADD16,MASK_ADD16)
```

Instruction Operand – every bit is 1 except if it is used to specify and operand