SIM2022

Generated by Doxygen 1.9.1

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 sim Namespace Reference

Classes

- struct Instruction
- class RegFile
- struct State
- class Decoder
- class Executor
- class Hart
- class Memory

Typedefs

- using Word = std::uint32_t
- using RegVal = Word
- using Addr = std::uint32_t
- using RegId = std::size_t

Functions

- template<typename T >
  consteval std::size_t sizeofBits ()

  Calculate size of a type in bits function.
- template<std::size_t high, std::size_t low>
  Word getBits (Word word)

  Get bits from number function.
- template<std::size_t pos, bool toSet>
  Word setBit (Word word)

  Set bits in number function.
- template<std::size_t newSize, std::size_t oldSize>
  Word signExtend (Word word)

  Sign extend number from one size to another.
- template<std::size_t oldSize>
  Word signExtend (Word word)

  Sign extend small number to Word.

Variables

- constexpr RegId kRegNum = 32
- constexpr std::uint8_t kBitsInByte = 8

4.1.1 Typedef Documentation

4.1.1.1 Word

```
using sim::Word = typedef std::uint32_t
```

Definition at line 8 of file common.hh.

4.1.1.2 RegVal

```
using sim::RegVal = typedef Word
```

Definition at line 9 of file common.hh.

4.1.1.3 Addr

```
using sim::Addr = typedef std::uint32_t
```

Definition at line 10 of file common.hh.

4.1.1.4 RegId

```
using sim::RegId = typedef std::size_t
```

Definition at line 11 of file common.hh.

4.1.2 Function Documentation

4.1.2.1 sizeofBits()

```
template<typename T >
consteval std::size_t sim::sizeofBits ( )
```

Calculate size of a type in bits function.

Note

All calculations are guaranteed to be compile-time

Template Parameters

T	type to calculate its size

Returns

std::size_t size in bits

Definition at line 22 of file common.hh.

References kBitsInByte.

4.1.2.2 getBits()

```
template<std::size_t high, std::size_t low>
Word sim::getBits (
            Word word )
```

Get bits from number function.

Template Parameters

high	end of bit range (msb)
low	begin of bit range (lsb)

Parameters

in	*word*	number to get bits from

Returns

Word bits from range [high, low] (shifted to the beginning)

Definition at line 34 of file common.hh.

Referenced by signExtend().

4.1.2.3 setBit()

```
template<std::size_t pos, bool toSet>
Word sim::setBit (
            Word word )
```

Set bits in number function.

Template Parameters

pos	Position of bit to set (start from 0)
toSet	value to set

Parameters

in	*word*	number to set bit in

Returns

Word number with bit set

Definition at line 53 of file common.hh.

4.1.2.4 signExtend() [1/2]

```
template<std::size_t newSize, std::size_t oldSize>
Word sim::signExtend (
            Word word )
```

Sign extend number from one size to another.

The idea of sign extension is to get leftmost bit and broadcast it to all new bits. Consider number 110_2 (oldSize = 3). Assume that we want to sign extend it to 7 bits. To simplify all listings, also assume that sizeof(Word) == 1. The implemented algorithm works as follows:

1. All bits left to oldSize - 1 are zeroed:
   - $01110110_2 \rightarrow 00000110_2$
2. Mask for current signbit is generated:
   - mask $\leftarrow 00000100_2$
3. Zeroed value is XORed with mask:
   - $XOR \begin{array}{l} 00000110_2 \\ 00000100_2 \end{array} = 00000010_2$
4. Mask is subtracted from previous result:
   - $00000010_2 - 00000100_2 = 2_{10} - 4_{10} = 00000000_2 - 2_{10} = 11111110_2$
5. All bits left to newSize - 1 are zeroed:
   - $11111110_2 \rightarrow 01111110_2$
6. Result is $01111110_2 = 126_{10}$

If sign bit is zero, then operations with mask do nothing together $\Rightarrow$ only zeroing has effect

Template Parameters

newSize	size to sign extend to
oldSize	initial size

Parameters

in	*word*	number to sign extend

Returns

Word sign extended number

Definition at line 91 of file common.hh.

References getBits().

4.1.2.5 signExtend() [2/2]

```
template<std::size_t oldSize>
Word sim::signExtend (
            Word word )
```

Sign extend small number to Word.

Template Parameters

oldSize	current size

Parameters

in	*word*	number to sign extend

Returns

Word sign extended number

Definition at line 113 of file common.hh.

4.1.3 Variable Documentation

4.1.3.1 kRegNum

```
constexpr RegId sim::kRegNum = 32  [constexpr]
```

Definition at line 13 of file common.hh.

4.1.3.2 kBitsInByte

```
constexpr std::uint8_t sim::kBitsInByte = 8  [constexpr]
```

Definition at line 14 of file common.hh.

Referenced by sizeofBits().

Chapter 5

Class Documentation

5.1 sim::Decoder Class Reference

`#include <decoder.hh>`

Static Public Member Functions

- static Instruction decode (Word binInst)

  Decode an instruction function.

5.1.1 Detailed Description

Definition at line 11 of file decoder.hh.

5.1.2 Member Function Documentation

5.1.2.1 decode()

```
static Instruction sim::Decoder::decode (
            Word binInst )  [static]
```

Decode an instruction function.

Parameters

binInst	instruction bytes to decode

Returns

Instruction decoded instruction

Referenced by sim::Hart::run().

The documentation for this class was generated from the following file:

- include/decoder/decoder.hh

5.2 sim::Executor Class Reference

`#include <executor.hh>`

Public Member Functions

- void execute (const Instruction &inst, State &state) const

5.2.1 Detailed Description

Definition at line 9 of file executor.hh.

5.2.2 Member Function Documentation

5.2.2.1 execute()

```
void sim::Executor::execute (
            const Instruction & inst,
            State & state ) const
```

Referenced by sim::Hart::run().

The documentation for this class was generated from the following file:

- include/executor/executor.hh

5.3 sim::Hart Class Reference

`#include <hart.hh>`

Public Member Functions

- void run ()

5.3.1 Detailed Description

Definition at line 14 of file hart.hh.

5.3.2 Member Function Documentation

5.3.2.1 run()

```
void sim::Hart::run ( )
```

Definition at line 5 of file hart.cc.

References sim::Decoder::decode(), sim::Executor::execute(), and sim::Memory::load().

The documentation for this class was generated from the following files:

- include/hart/hart.hh
- src/hart/hart.cc

5.4 sim::Instruction Struct Reference

```
#include <inst.hh>
```

Public Attributes

- RegId rs1 {}
- RegId rs2 {}
- RegId rs3 {}
- RegId rd {}
- RegId rm {}
- RegId csr {}
- OpType type {OpType::UNKNOWN}
- RegVal imm {}

5.4.1 Detailed Description

Definition at line 8 of file inst.hh.

5.4.2 Member Data Documentation

5.4.2.1 rs1

```
RegId sim::Instruction::rs1 {}
```

Definition at line 9 of file inst.hh.

5.4.2.2 rs2

```
RegId sim::Instruction::rs2 {}
```

Definition at line 10 of file inst.hh.

5.4.2.3 rs3

```
RegId sim::Instruction::rs3 {}
```

Definition at line 11 of file inst.hh.

5.4.2.4 rd

```
RegId sim::Instruction::rd {}
```

Definition at line 13 of file inst.hh.

5.4.2.5 rm

```
RegId sim::Instruction::rm {}
```

Definition at line 14 of file inst.hh.

5.4.2.6 csr

```
RegId sim::Instruction::csr {}
```

Definition at line 15 of file inst.hh.

5.4.2.7 type

```
OpType sim::Instruction::type {OpType::UNKNOWN}
```

Definition at line 17 of file inst.hh.

5.4.2.8 imm

```
RegVal sim::Instruction::imm {}
```

Definition at line 18 of file inst.hh.

The documentation for this struct was generated from the following file:

- include/common/inst.hh

5.5 sim::Memory Class Reference

```
#include <memory.hh>
```

Public Member Functions

- Word load (std::size_t)

5.5.1 Detailed Description

Definition at line 8 of file memory.hh.

5.5.2 Member Function Documentation

5.5.2.1 load()

```
Word sim::Memory::load (
            std::size_t  )
```

Referenced by sim::Hart::run().

The documentation for this class was generated from the following file:

- include/memory/memory.hh

5.6 sim::RegFile Class Reference

```
#include <state.hh>
```

Public Member Functions

- Word get (RegId regnum) const
- void set (RegId regnum, Word val)

5.6.1 Detailed Description

Definition at line 12 of file state.hh.

5.6.2 Member Function Documentation

5.6.2.1 get()

```
Word sim::RegFile::get (
            RegId regnum ) const  [inline]
```

Definition at line 17 of file state.hh.

5.6.2.2 set()

```
void sim::RegFile::set (
            RegId regnum,
            Word val )  [inline]
```

Definition at line 19 of file state.hh.

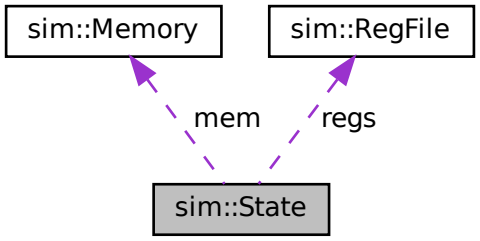
The documentation for this class was generated from the following file:

- include/common/state.hh

5.7 sim::State Struct Reference

`#include <state.hh>`

Collaboration diagram for sim::State:



Public Attributes

- Addr pc {}
- Addr npc {}
- RegFile regs {}
- Memory mem {}

5.7.1 Detailed Description

Definition at line 26 of file state.hh.

5.7.2 Member Data Documentation

5.7.2.1 pc

`Addr sim::State::pc {}`

Definition at line 27 of file state.hh.

5.7.2.2 npc

`Addr sim::State::npc {}`

Definition at line 28 of file state.hh.

5.7.2.3 regs

```
RegFile sim::State::regs {}
```

Definition at line 29 of file state.hh.

5.7.2.4 mem

```
Memory sim::State::mem {}
```

Definition at line 30 of file state.hh.

The documentation for this struct was generated from the following file:

- include/common/state.hh

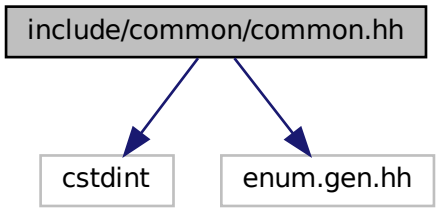
Chapter 6

File Documentation

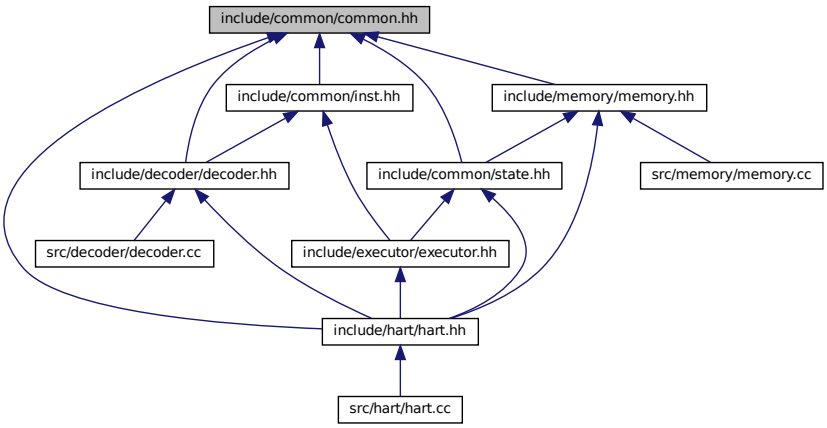
6.1 include/common/common.hh File Reference

```
#include <cstdint>
#include "enum.gen.hh"
```

Include dependency graph for common.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- sim

Typedefs

- using sim::Word = std::uint32_t
- using sim::RegVal = Word
- using sim::Addr = std::uint32_t
- using sim::RegId = std::size_t

Functions

- template<typename T >
  consteval std::size_t sim::sizeofBits ()

  Calculate size of a type in bits function.
- template<std::size_t high, std::size_t low>
  Word sim::getBits (Word word)

  Get bits from number function.
- template<std::size_t pos, bool toSet>
  Word sim::setBit (Word word)

  Set bits in number function.
- template<std::size_t newSize, std::size_t oldSize>
  Word sim::signExtend (Word word)

  Sign extend number from one size to another.
- template<std::size_t oldSize>
  Word sim::signExtend (Word word)

  Sign extend small number to Word.

Variables

- constexpr RegId sim::kRegNum = 32
- constexpr std::uint8_t sim::kBitsInByte = 8

6.2 common.hh

```
00001 #ifndef __INCLUDE_COMMON_COMMON_HH__
00002 #define __INCLUDE_COMMON_COMMON_HH__
00003
00004 #include <cstdint>
00005
00006 namespace sim {
00007
00008 using Word = std::uint32_t;
00009 using RegVal = Word;
00010 using Addr = std::uint32_t;
00011 using RegId = std::size_t;
00012
00013 constexpr RegId kRegNum = 32;
00014 constexpr std::uint8_t kBitsInByte = 8;
00015
00016 /**
00017  * @brief Calculate size of a type in bits function
00018  * @note All calculations are guaranteed to be compile-time
00019  * @tparam T type to calculate its size
00020  * @return std::size_t size in bits
00021  */
00022 template <typename T> consteval std::size_t sizeofBits() {
```

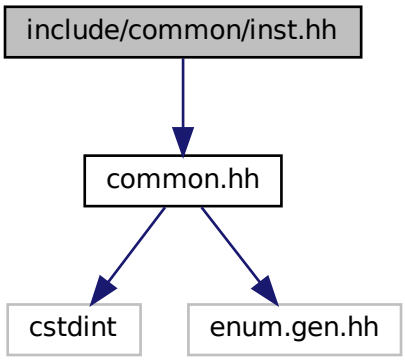
```
00023   return sizeof(T) * kBitsInByte;
00024 }
00025
00026 /**
00027  * @brief Get bits from number function
00028  *
00029  * @tparam high end of bit range (msb)
00030  * @tparam low begin of bit range (lsb)
00031  * @param[in] word number to get bits from
00032  * @return Word bits from range [high, low] (shifted to the beginning)
00033  */
00034 template <std::size_t high, std::size_t low> Word getBits(Word word) {
00035   static_assert(high >= low, "Incorrect bits range");
00036   static_assert(high < sizeofBits<Word>(), "Bit index out of range");
00037
00038   auto mask = ~Word(0);
00039   if constexpr (high != sizeofBits<Word>() - 1)
00040     mask = ~(mask « (high + 1));
00041
00042   return (word & mask) » low;
00043 }
00044
00045 /**
00046  * @brief Set bits in number function
00047  *
00048  * @tparam pos Position of bit to set (start from 0)
00049  * @tparam toSet value to set
00050  * @param[in] word number to set bit in
00051  * @return Word number with bit set
00052  */
00053 template <std::size_t pos, bool toSet> Word setBit(Word word) {
00054   static_assert(pos < sizeofBits<Word>(), "Bit index out of range");
00055
00056   constexpr auto mask = Word(1) « pos;
00057   if constexpr (toSet)
00058     return word | mask;
00059
00060   return word & ~mask;
00061 }
00062
00063 /**
00064  * @brief Sign extend number from one size to another
00065  * @details
00066  * The idea of sign extension is to get leftmost bit and broadcast it to
00067  * all new bits.
00068  * Consider number \f$ 110_2 \f$ (oldSize = \f$ 3 \f$).
00069  * Assume that we want to sign extend it to \f$ 7 \f$ bits. To simplify all
00070  * listings, also assume that sizeof(Word) == \f$ 1 \f$. The implemented algorithm works
00071  * as follows:
00072  *  -# All bits left to oldSize - 1 are zeroed:
00073  *    - \f$ 01110110_2 \f$ \f$ \rightarrow \f$  \f$ 00000110_2 \f$
00074  *  -# Mask for current signbit is generated:
00075  *    - mask \f$ \leftarrow \f$ \f$ 00000100_2 \f$
00076  *  -# Zeroed value is XORed with mask:
00077  *    - \f$ XOR \:\:\: \frac{00000110_2}{00000100_2} = 00000010_2 \f$
00078  *  -# Mask is subtracted from previous result:
00079  *    - \f$ 00000010_2 - 00000100_2 = 2_{10} - 4_{10} = 00000000_2 - 2_{10} = 11111110_2 \f$
00080  *  -# All bits left to newSize - 1 are zeroed:
00081  *    - \f$ 11111110_2 \f$ \f$ \rightarrow \f$  \f$ 01111110_2 \f$
00082  *  -# Result is \f$ 01111110_2 = 126_{10}\f$
00083  *
00084  * If sign bit is zero, then operations with mask do nothing together \f$ \Rightarrow \f$
00085  * only zeroing has effect
00086  * @tparam newSize size to sign extend to
00087  * @tparam oldSize initial size
00088  * @param[in] word number to sign extend
00089  * @return Word sign extended number
00090  */
00091 template <std::size_t newSize, std::size_t oldSize> Word signExtend(Word word) {
00092   static_assert(newSize >= oldSize, "Trying to sign extend to smaller size");
00093   static_assert(oldSize > 0, "Initial size must be non-zero");
00094   static_assert(newSize <= sizeofBits<Word>(), "newSize is out of bits range");
00095
00096   if constexpr (newSize == oldSize)
00097     return word;
00098
00099   Word zeroed = getBits<oldSize - 1, 0>(word);
00100   constexpr Word mask = Word(1) « (oldSize - 1);
00101   Word res = (zeroed ^ mask) - mask;
00102
00103   return getBits<newSize - 1, 0>(res);
00104 }
00105
00106 /**
00107  * @brief Sign extend small number to Word
00108  *
00109  * @tparam oldSize current size
```

```
00110  * @param[in] word number to sign extend
00111  * @return Word sign extended number
00112  */
00113 template <std::size_t oldSize> Word signExtend(Word word) {
00114   return signExtend<sizeofBits<Word>(), oldSize>(word);
00115 }
00116
00117 #include "enum.gen.hh"
00118
00119 } // namespace sim
00120
00121 #endif // __INCLUDE_COMMON_COMMON_HH__
```

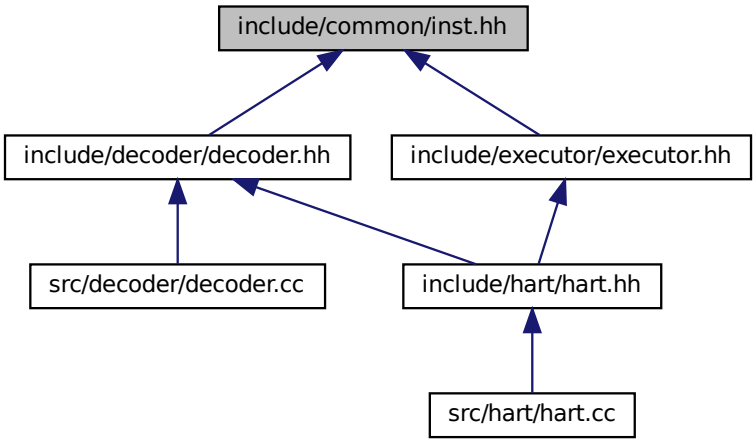
6.3 include/common/inst.hh File Reference

`#include "common.hh"`

Include dependency graph for inst.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct sim::Instruction

Namespaces

- sim

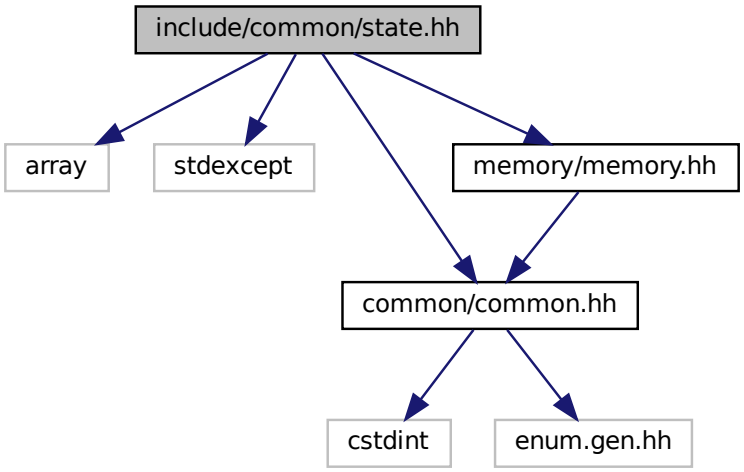
6.4 inst.hh

```
00001 #ifndef __INCLUDE_COMMON_INST_HH__
00002 #define __INCLUDE_COMMON_INST_HH__
00003
00004 #include "common.hh"
00005
00006 namespace sim {
00007
00008 struct Instruction final {
00009   RegId rs1{};
00010   RegId rs2{};
00011   RegId rs3{};
00012
00013   RegId rd{};
00014   RegId rm{};  // rounding mode (for future use w/ floating-point operations)
00015   RegId csr{}; // a placeholder
00016
00017   OpType type{OpType::UNKNOWN};
00018   RegVal imm{};
00019 };
00020
00021 } // namespace sim
00022
00023 #endif // __INCLUDE_COMMON_INST_HH__
```

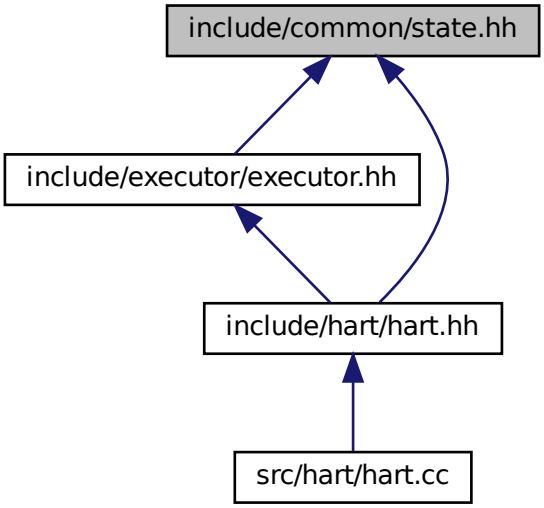
6.5 include/common/state.hh File Reference

```
#include <array>
#include <stdexcept>
#include "common/common.hh"
#include "memory/memory.hh"
```

Include dependency graph for state.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class sim::RegFile
- struct sim::State

Namespaces

- sim

6.6 state.hh

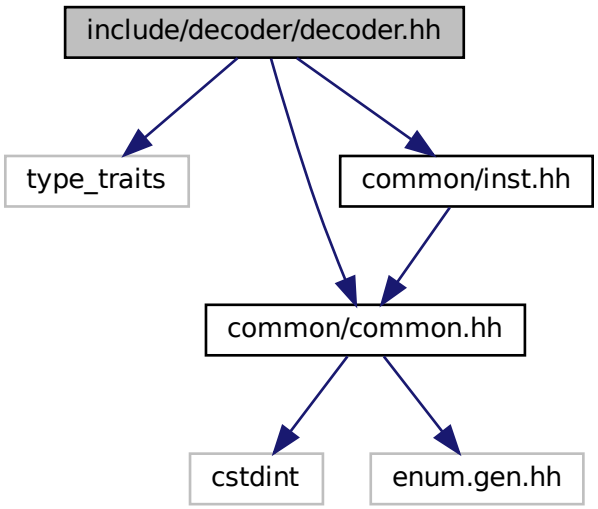
```
00001 #ifndef __INCLUDE_STATE_STATE_HH__
00002 #define __INCLUDE_STATE_STATE_HH__
00003
00004 #include <array>
00005 #include <stdexcept>
00006
00007 #include "common/common.hh"
00008 #include "memory/memory.hh"
00009
00010 namespace sim {
00011
00012 class RegFile final {
00013 private:
00014   std::array<RegVal, kRegNum> regs{};
00015
00016 public:
00017   Word get(RegId regnum) const { return regs.at(regnum); }
00018
00019   void set(RegId regnum, Word val) {
00020     if (regnum == 0)
00021       throw std::runtime_error{"Trying to set value to register x0"};
00022     regs.at(regnum) = val;
00023   }
00024 };
00025
```

```
00026 struct State final {
00027   Addr pc{};
00028   Addr npc{};
00029   RegFile regs{};
00030   Memory mem{};
00031 };
00032
00033 } // namespace sim
00034
00035 #endif // __INCLUDE_STATE_STATE_HH__
```

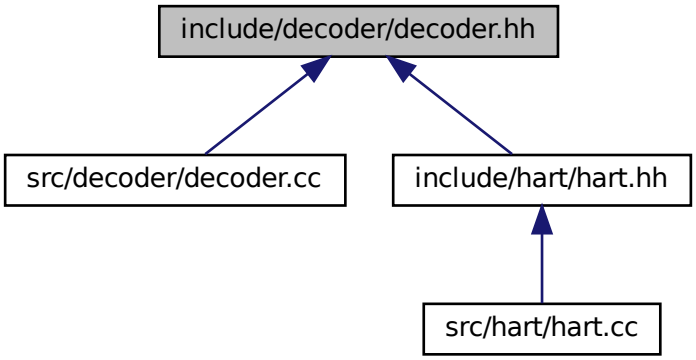
6.7 include/decoder/decoder.hh File Reference

```
#include <type_traits>
#include "common/common.hh"
#include "common/inst.hh"
```

Include dependency graph for decoder.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class sim::Decoder

Namespaces

- sim

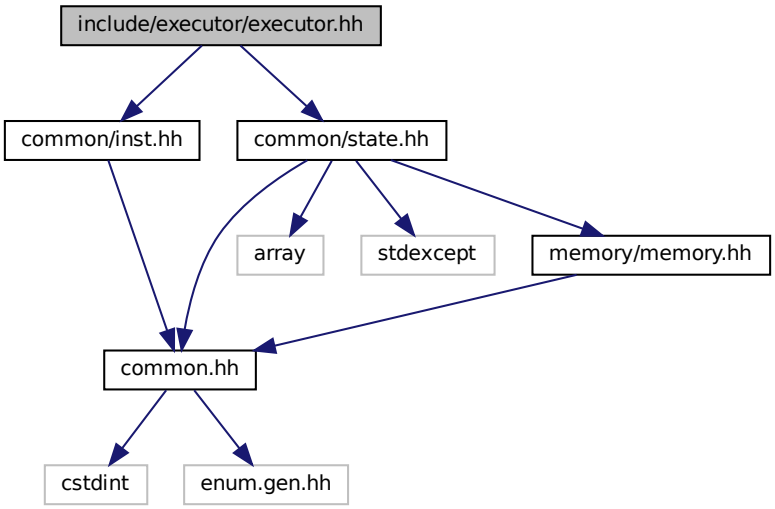
6.8 decoder.hh

```
00001 #ifndef __INCLUDE_DECODER_DECODER_HH__
00002 #define __INCLUDE_DECODER_DECODER_HH__
00003
00004 #include <type_traits>
00005
00006 #include "common/common.hh"
00007 #include "common/inst.hh"
00008
00009 namespace sim {
00010
00011 class Decoder final {
00012 public:
00013   /**
00014    * @brief Decode an instruction function
00015    *
00016    * @param binInst instruction bytes to decode
00017    * @return Instruction decoded instruction
00018    */
00019   static Instruction decode(Word binInst);
00020 };
00021
00022 } // namespace sim
00023
00024 #endif // __INCLUDE_DECODER_DECODER_HH__
```

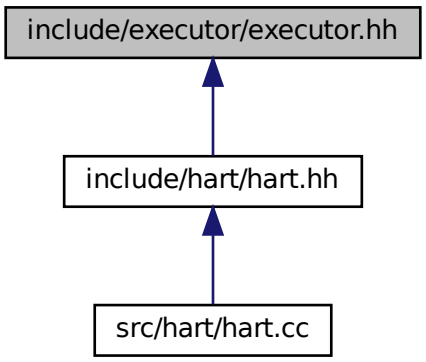
6.9 include/executor/executor.hh File Reference

```
#include "common/inst.hh"
#include "common/state.hh"
```

Include dependency graph for executor.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class sim::Executor

Namespaces

- sim

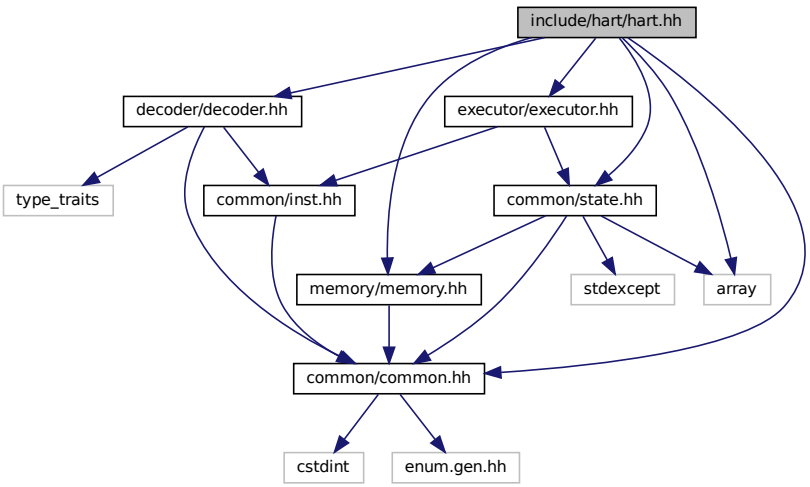
6.10 executor.hh

```
00001 #ifndef __INCLUDE_EXEC_EXEC_HH__
00002 #define __INCLUDE_EXEC_EXEC_HH__
00003
00004 #include "common/inst.hh"
00005 #include "common/state.hh"
00006
00007 namespace sim {
00008
00009 class Executor final {
00010 public:
00011   void execute(const Instruction &inst, State &state) const;
00012 };
00013
00014 } // namespace sim
00015
00016 #endif // __INCLUDE_EXEC_EXEC_HH__
```

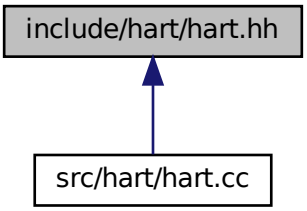
6.11 include/hart/hart.hh File Reference

```
#include <array>
#include "common/common.hh"
#include "common/state.hh"
#include "decoder/decoder.hh"
#include "executor/executor.hh"
#include "memory/memory.hh"
```

Include dependency graph for hart.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class sim::Hart

Namespaces

- sim

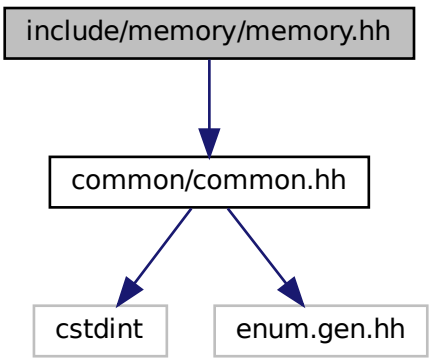
6.12 hart.hh

```
00001 #ifndef __INCLUDE_HART_HART_HH__
00002 #define __INCLUDE_HART_HART_HH__
00003
00004 #include <array>
00005
00006 #include "common/common.hh"
00007 #include "common/state.hh"
00008 #include "decoder/decoder.hh"
00009 #include "executor/executor.hh"
00010 #include "memory/memory.hh"
00011
00012 namespace sim {
00013
00014 class Hart final {
00015 private:
00016   State state_;
00017   Executor exec_;
00018   Decoder decoder_;
00019
00020   Memory &mem() { return state_.mem; };
00021   Addr &pc() { return state_.pc; };
00022
00023 public:
00024   [[noreturn]] void run();
00025 };
00026
00027 } // namespace sim
00028
00029 #endif // __INCLUDE_HART_HART_HH__
```

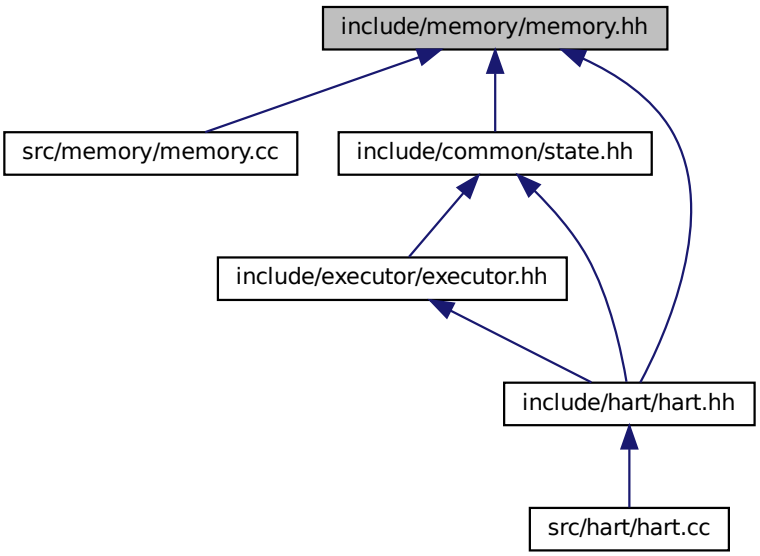
6.13 include/memory/memory.hh File Reference

```
#include "common/common.hh"
```

Include dependency graph for memory.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class sim::Memory

Namespaces

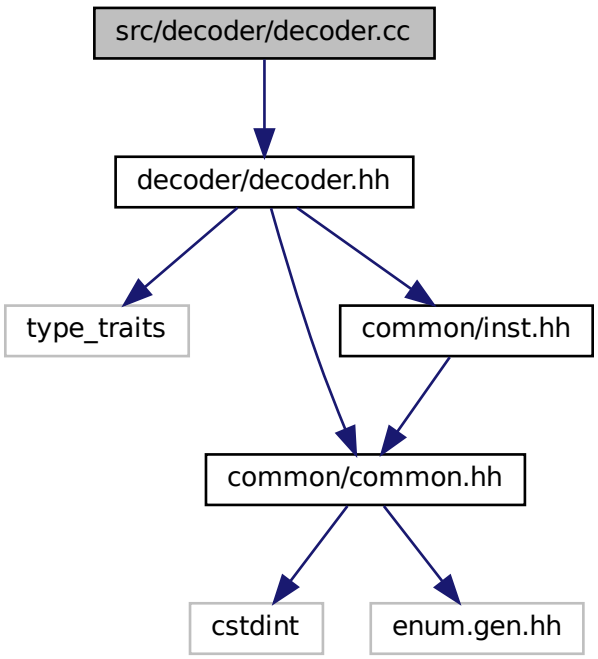
- sim

6.14 memory.hh

```
00001 #ifndef __INCLUDE_MEMORY_MEMORY_HH__
00002 #define __INCLUDE_MEMORY_MEMORY_HH__
00003
00004 #include "common/common.hh"
00005
00006 namespace sim {
00007
00008 class Memory final {
00009 public:
00010   Word load(std::size_t);
00011 };
00012
00013 } // namespace sim
00014
00015 #endif // __INCLUDE_MEMORY_MEMORY_HH__
```

6.15 src/decoder/decoder.cc File Reference

`#include "decoder/decoder.hh"`
Include dependency graph for decoder.cc:



Namespaces

- sim

6.16 decoder.cc

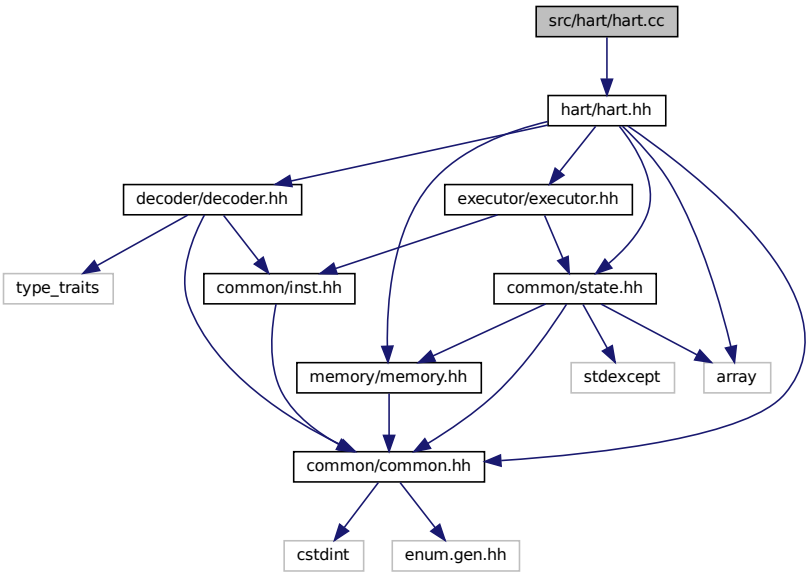
```
00001 #include "decoder/decoder.hh"
00002
00003 namespace sim {}
```

6.17 src/executor/executor.cc File Reference

6.18 executor.cc

6.19 src/hart/hart.cc File Reference

`#include "hart/hart.hh"`
Include dependency graph for hart.cc:



Namespaces

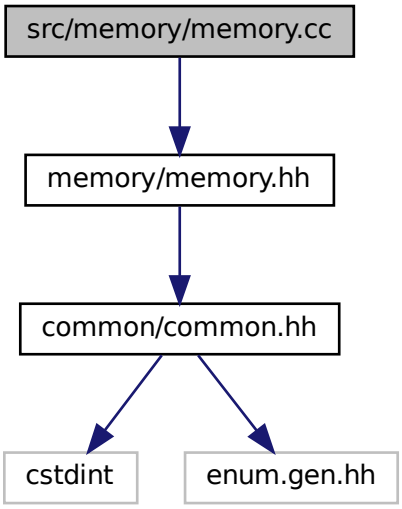
- sim

6.20 hart.cc

```
00001 #include "hart/hart.hh"
00002
00003 namespace sim {
00004
00005 void Hart::run() {
00006   for (;;) {
00007     auto binInst = mem().load(pc());
00008     auto inst = decoder_.decode(binInst);
00009     exec_.execute(inst, state_);
00010   }
00011 }
00012
00013 } // namespace sim
```

6.21 src/memory/memory.cc File Reference

`#include "memory/memory.hh"`
Include dependency graph for memory.cc:



Namespaces

- sim

6.22 memory.cc

```
00001 #include "memory/memory.hh"
00002
00003 namespace sim {}
```