ARM11 Emulator

Generated by Doxygen 1.8.11

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 instruction_t Struct Reference

A struct that holds information about a decoded instruction.

`#include <instruction.h>`

Data Fields

- instruction_type_t type
  The type of instruction (None, Zero, DPI, MUL, SDT, or BRA).
- byte_t cond
  The condition code.
- opcode_t operation
  The opcode, for data processing instructions.
- uint32_t immediate_value
  An immediate offset or operand.
- reg_address_t rn
  Register Rn.
- reg_address_t rd
  Register Rd.
- reg_address_t rs
  Register Rs.
- reg_address_t rm
  Register Rm.
- bool flag_0
  Holds the I or A bit (depending on instruction type).
- bool flag_1
  Holds the S or P bit (depending on instruction type).
- bool flag_2
  Holds the U bit (SDT instructions only).
- bool flag_3
  Holds the L bit (SDT instructions only).
- shift_t shift_type
  The type of shift to be used.
- byte_t shift_amount
  The number of shifts to be applied.

3.1.1 Detailed Description

A struct that holds information about a decoded instruction.

3.1.2 Field Documentation

3.1.2.1 byte_t instruction_t::cond

The condition code.

3.1.2.2 bool instruction_t::flag_0

Holds the I or A bit (depending on instruction type).

3.1.2.3 bool instruction_t::flag_1

Holds the S or P bit (depending on instruction type).

3.1.2.4 bool instruction_t::flag_2

Holds the U bit (SDT instructions only).

3.1.2.5 bool instruction_t::flag_3

Holds the L bit (SDT instructions only).

3.1.2.6 uint32_t instruction_t::immediate_value

An immediate offset or operand.

3.1.2.7 opcode_t instruction_t::operation

The opcode, for data processing instructions.

3.1.2.8 reg_address_t instruction_t::rd

Register Rd.

3.1.2.9 reg_address_t instruction_t::rm

Register Rm.

3.1.2.10 reg_address_t instruction_t::rn

Register Rn.

3.1.2.11 reg_address_t instruction_t::rs

Register Rs.

3.1.2.12 byte_t instruction_t::shift_amount

The number of shifts to be applied.

3.1.2.13 shift_t instruction_t::shift_type

The type of shift to be used.

3.1.2.14 instruction_type_t instruction_t::type

The type of instruction (None, Zero, DPI, MUL, SDT, or BRA).

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

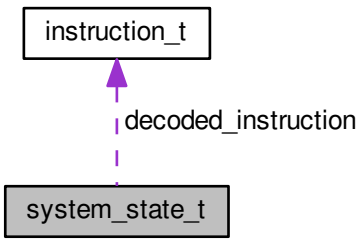
- instruction.h

3.2 system_state_t Struct Reference

A struct that holds information about the current system state.

`#include <system_state.h>`

Collaboration diagram for system_state_t:



Data Fields

- word_t registers [NUM_REGISTERS]

  Holds the values currently held in registers.
- byte_t memory [NUM_ADDRESSES]

  Holds the values currently held in memory.
- word_t fetched_instruction

  Holds the last fetched instruction, as a word.
- instruction_t * decoded_instruction

  Holds the last decoded instruction, as an instruction_t type.
- bool has_fetched_instruction

  Whether or not the system currently has a fetched instruction.

3.2.1 Detailed Description

A struct that holds information about the current system state.

3.2.2 Field Documentation

3.2.2.1 instruction_t* system_state_t::decoded_instruction

Holds the last decoded instruction, as an instruction_t type.

3.2.2.2 word_t system_state_t::fetched_instruction

Holds the last fetched instruction, as a word.

3.2.2.3 bool system_state_t::has_fetched_instruction

Whether or not the system currently has a fetched instruction.

3.2.2.4 byte_t system_state_t::memory[NUM_ADDRESSES]

Holds the values currently held in memory.

3.2.2.5 word_t system_state_t::registers[NUM_REGISTERS]

Holds the values currently held in registers.

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

- system_state.h

3.3 value_carry_t Struct Reference

A struct that has a value and a carry.

`#include <value_carry.h>`

Data Fields

- word_t value
  The value.
- bool carry
  Whether or not there is a carry bit present.

3.3.1 Detailed Description

A struct that has a value and a carry.

3.3.2 Field Documentation

3.3.2.1 bool value_carry_t::carry

Whether or not there is a carry bit present.

3.3.2.2 word_t value_carry_t::value

The value.

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

- value_carry.h

Chapter 4

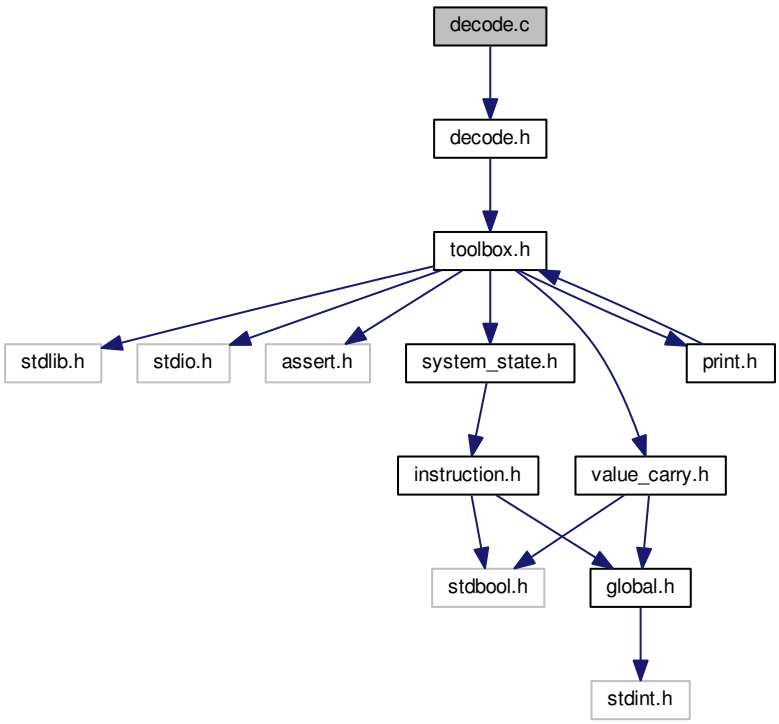
File Documentation

4.1 decode.c File Reference

Functions for the decode cycle.

```
#include "decode.h"
```

Include dependency graph for decode.c:



Functions

- void decode_instruction (system_state_t *machine)

  Decodes the fetched instruction in current system state.
- void halt (system_state_t *machine)

  Sets decoded_instruction type to a stop (ZER) instruction.
- void branch (system_state_t *machine)

  Set branch instruction data in decoded_instruction.
- void multiply (system_state_t *machine)

  Set multiply instruction data in decoded_instruction.
- void single_data_transfer (system_state_t *machine)

  Set single_data_transfer instruction data in decoded_instruction.
- void data_processing (system_state_t *machine)

  Set data_processing instruction data in decoded_instruction.

4.1.1 Detailed Description

Functions for the decode cycle.

4.1.2 Function Documentation

4.1.2.1 void branch (system_state_t * *machine*)

Set branch instruction data in decoded_instruction.

The offset (24 bits) is bit 0 to 23 of the branch instruction. It is then bit shifted to the left by 2 and then sign extended to 32 bits. The offset is stored in the immediate_value of the decoded_instruction.

Parameters

machine	The current system state.

4.1.2.2 void data_processing (system_state_t * *machine*)

Set data_processing instruction data in decoded_instruction.

The fields in decoded_instruction are used as follows:

- flag_0 stores the I bit:
  - If set, the Operand2 is used as an immediate constant.
  - Otherwise, Operand2 is used as a shifted register.
- flag_1 stores the S bit (if set, CPSR flags are set when executed).
- operation is used to store the corresponding opcode_t enum to the opcode. provided in the fetched_↩ instruction.
- rd is the source/destination register address.
- rn is the first operand register.

Parameters

machine	The current system state.

4.1.2.3 void decode_instruction (system_state_t * *machine*)

Decodes the fetched instruction in current system state.

Given the pointer to the current system state, it moves the fetched instruction information into the decoded_↩ instruction struct (for use when executing the decoded instruction). A pre-condition is that the instruction must not be all zero (type ZER).

Parameters

machine	The current system state.

4.1.2.4 void halt (system_state_t * *machine*)

Sets decoded_instruction type to a stop (ZER) instruction.

Parameters

machine	The current system state.

4.1.2.5 void multiply (system_state_t * *machine*)

Set multiply instruction data in decoded_instruction.

The fields in decoded_instruction are used as follows:

- flag_0 stores the A bit (if set, perform multiply and accumulate).
- flag_1 stores the S bit (if set, CPSR flags are set when executed).
- rd is the destination register address.
- rn, rs and rm are the addresses of the operand registers.

Parameters

machine	The current system state.

4.1.2.6 void single_data_transfer (system_state_t * *machine*)

Set single_data_transfer instruction data in decoded_instruction.

The fields in decoded_instruction are used as follows:

- flag_0 stores the I bit:
  - If set, Offset is used as a shifted register.
  - Otherwise, Offset is used as an unsigned 12 bit immediate offset).
- flag_1 stores the P bit:
  - If set, pre-indexing is used.
  - Otherwise, post-indexing is used.
- flag_2 stores the U bit:
  - If set, Offset is added to the base register.
  - Otherwise, Offset is subtracted from the base register.
- flag_3 stores the L bit:
  - If set, the word is loaded from memory.
  - Otherwise, the word is stored into memory.
- rd is the source/destination register address.
- rn is the base register.

Parameters

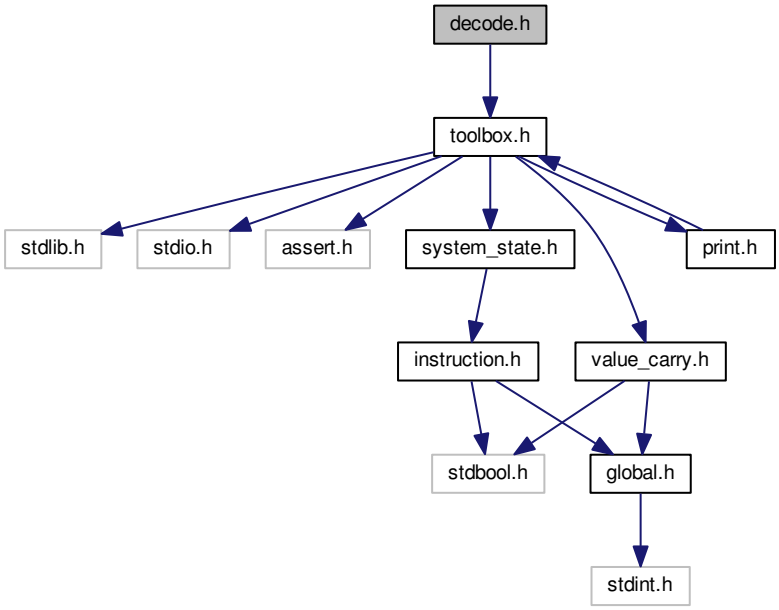
machine	The current system state.

4.2 decode.h File Reference

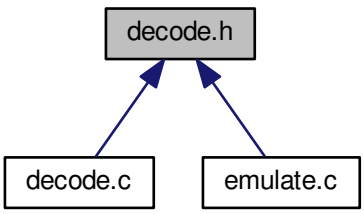
Header file for decode.c.

```
#include "toolbox.h"
```

Include dependency graph for decode.h:



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



Functions

- void decode_instruction (system_state_t *machine)

  Decodes the fetched instruction in current system state.
- void halt (system_state_t *machine)

  Sets decoded_instruction type to a stop (ZER) instruction.
- void branch (system_state_t *machine)

  Set branch instruction data in decoded_instruction.
- void single_data_transfer (system_state_t *machine)

Set single_data_transfer instruction data in decoded_instruction.

- void multiply (system_state_t *machine)

  Set multiply instruction data in decoded_instruction.

- void data_processing (system_state_t *machine)

  Set data_processing instruction data in decoded_instruction.

4.2.1 Detailed Description

Header file for decode.c.

4.2.2 Function Documentation

4.2.2.1 void branch (system_state_t * *machine*)

Set branch instruction data in decoded_instruction.

The offset (24 bits) is bit 0 to 23 of the branch instruction. It is then bit shifted to the left by 2 and then sign extended to 32 bits. The offset is stored in the immediate_value of the decoded_instruction.

Parameters

machine	The current system state.

4.2.2.2 void data_processing (system_state_t * *machine*)

Set data_processing instruction data in decoded_instruction.

The fields in decoded_instruction are used as follows:

- flag_0 stores the I bit:
  - If set, the Operand2 is used as an immediate constant.
  - Otherwise, Operand2 is used as a shifted register.
- flag_1 stores the S bit (if set, CPSR flags are set when executed).
- operation is used to store the corresponding opcode_t enum to the opcode. provided in the fetched_↩ instruction.
- rd is the source/destination register address.
- rn is the first operand register.

Parameters

machine	The current system state.

4.2.2.3 void decode_instruction (system_state_t * machine)

Decodes the fetched instruction in current system state.

Given the pointer to the current system state, it moves the fetched instruction information into the decoded_↩ instruction struct (for use when executing the decoded instruction). A pre-condition is that the instruction must not be all zero (type ZER).

Parameters

machine	The current system state.

4.2.2.4 void halt (system_state_t * machine)

Sets decoded_instruction type to a stop (ZER) instruction.

Parameters

machine	The current system state.

4.2.2.5 void multiply (system_state_t * machine)

Set multiply instruction data in decoded_instruction.

The fields in decoded_instruction are used as follows:

- flag_0 stores the A bit (if set, perform multiply and accumulate).
- flag_1 stores the S bit (if set, CPSR flags are set when executed).
- rd is the destination register address.
- rn, rs and rm are the addresses of the operand registers.

Parameters

machine	The current system state.

4.2.2.6 void single_data_transfer (system_state_t * machine)

Set single_data_transfer instruction data in decoded_instruction.

The fields in decoded_instruction are used as follows:

- flag_0 stores the I bit:
  - If set, Offset is used as a shifted register.

  - Otherwise, Offset is used as an unsigned 12 bit immediate offset).
- flag_1 stores the P bit:
  - If set, pre-indexing is used.
  - Otherwise, post-indexing is used.
- flag_2 stores the U bit:
  - If set, Offset is added to the base register.
  - Otherwise, Offset is subtracted from the base register.
- flag_3 stores the L bit:
  - If set, the word is loaded from memory.
  - Otherwise, the word is stored into memory.
- rd is the source/destination register address.
- rn is the base register.

Parameters

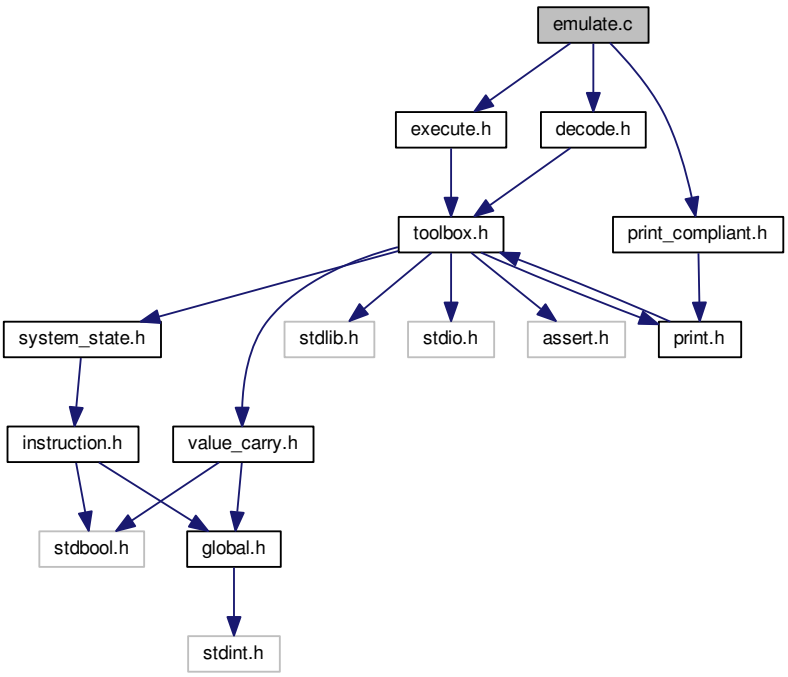
machine	The current system state.

4.3 emulate.c File Reference

The main functionality for the ARM11 emulator.

```
#include "decode.h"
#include "execute.h"
#include "print_compliant.h"
```

Include dependency graph for emulate.c:



Functions

- int main (int argc, char **argv)

  Emulates an ARM11 machine operating on a given binary file.

4.3.1 Detailed Description

The main functionality for the ARM11 emulator.

4.3.2 Function Documentation

4.3.2.1 int main (int *argc,* char ** *argv*)

Emulates an ARM11 machine operating on a given binary file.

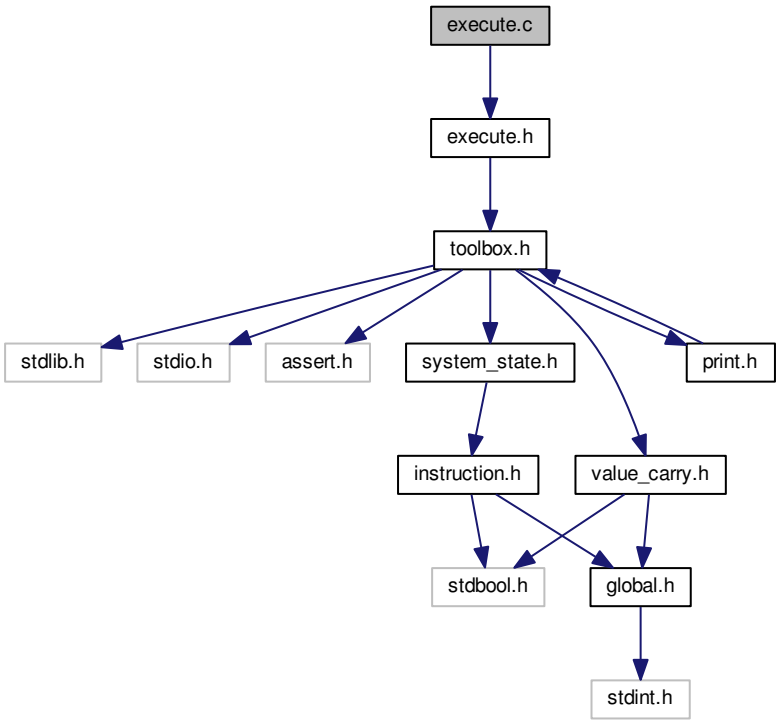
The user must provide a single argument, which is a valid file name for an ARM11 binary object code file. This function emulates the ARM architecture, returning details of the registers and non-zero memory at the end of execution.

4.4 execute.c File Reference

Functions for the execute cycle.

```
#include "execute.h"
```

Include dependency graph for execute.c:



Functions

- int condition (system_state_t *machine)

  Returns whether the condition is met.
- void execute (system_state_t *machine)

  Runs one execute cycle.
- void execute_dpi (system_state_t *machine)

  Executes a data processing instruction.
- void execute_mul (system_state_t *machine)

  Executes a multiply instruction.
- void execute_sdt (system_state_t *machine)

  Executes a single data transfer instruction.
- void execute_branch (system_state_t *machine)

  Executes a branch instruction.

4.4.1 Detailed Description

Functions for the execute cycle.

4.4.2 Function Documentation

4.4.2.1 int condition (system_state_t * machine)

Returns whether the condition is met.

Returns true if and only if the condition required by the current decoded instruction is met by the current state of the flags register (CPSR).

Parameters

machine	The current system state.

Returns

Whether condition is met.

4.4.2.2 void execute (system_state_t * machine)

Runs one execute cycle.

Executes the current decoded instruction if the condition is met, and updates the system state accordingly. A pre-condition is that the instruction must not be type NUL or ZER.

Parameters

machine	The current system state.

4.4.2.3 void execute_branch (system_state_t * machine)

Executes a branch instruction.

Parameters

machine	The current system state.

4.4.2.4 void execute_dpi (system_state_t * machine)

Executes a data processing instruction.

Parameters

machine	The current system state.

4.4.2.5 void execute_mul (system_state_t * machine)

Executes a multiply instruction.

Parameters

machine	The current system state.

4.4.2.6 void execute_sdt (system_state_t * machine)

Executes a single data transfer instruction.

Parameters

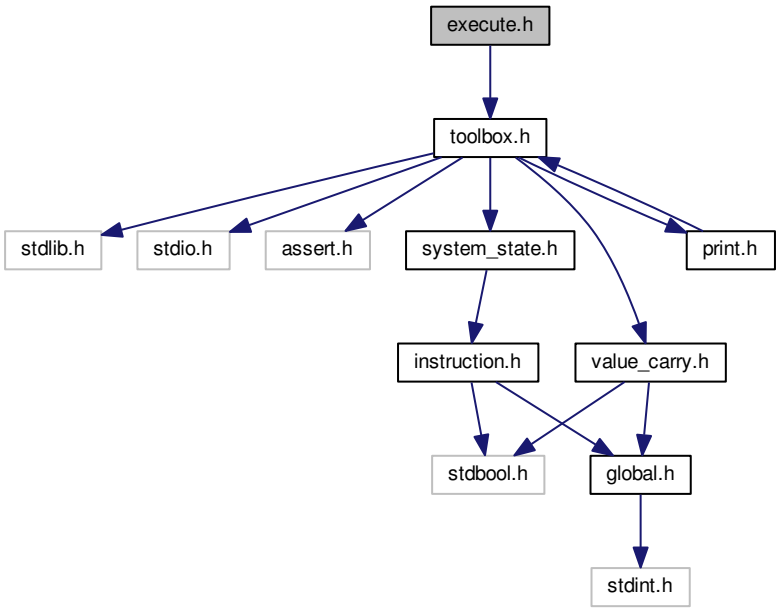
machine	The current system state.

4.5 execute.h File Reference

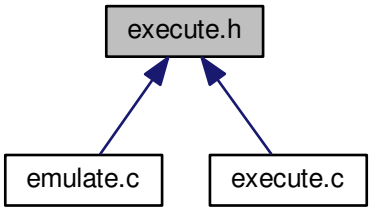
Header file for execute.c.

```
#include "toolbox.h"
```

Include dependency graph for execute.h:



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



Functions

- void execute (system_state_t *machine)
  Runs one execute cycle.
- void execute_dpi (system_state_t *machine)
  Executes a data processing instruction.
- void execute_mul (system_state_t *machine)
  Executes a multiply instruction.
- void execute_branch (system_state_t *machine)
  Executes a branch instruction.
- void execute_sdt (system_state_t *machine)
  Executes a single data transfer instruction.

4.5.1 Detailed Description

Header file for execute.c.

4.5.2 Function Documentation

4.5.2.1 void execute (system_state_t * *machine*)

Runs one execute cycle.

Executes the current decoded instruction if the condition is met, and updates the system state accordingly. A pre-condition is that the instruction must not be type NUL or ZER.

Parameters

machine	The current system state.

4.5.2.2 void execute_branch (system_state_t * machine)

Executes a branch instruction.

Parameters

machine	The current system state.

4.5.2.3 void execute_dpi (system_state_t * machine)

Executes a data processing instruction.

Parameters

machine	The current system state.

4.5.2.4 void execute_mul (system_state_t * machine)

Executes a multiply instruction.

Parameters

machine	The current system state.

4.5.2.5 void execute_sdt (system_state_t * machine)

Executes a single data transfer instruction.

Parameters

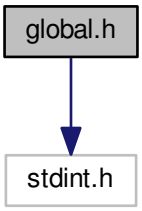
machine	The current system state.

4.6 global.h File Reference

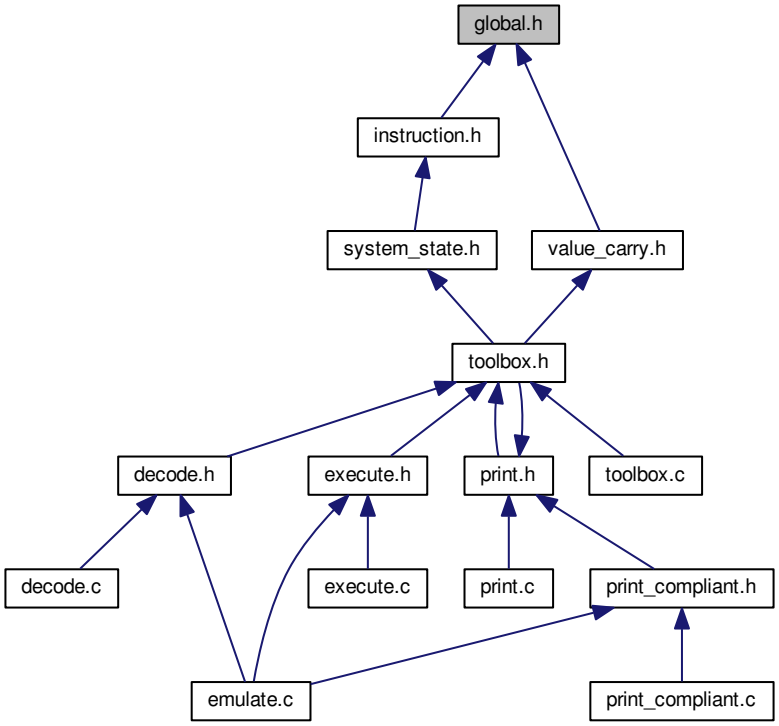
Definition of useful constants and type aliases.

```
#include <stdint.h>
```

Include dependency graph for global.h:



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



Macros

- #define NUM_REGISTERS 17

  The total number of registers.
- #define NUM_ADDRESSES 65536

  The total number of memory addresses.

- #define WORD_SIZE 32

  The architecture word size.
- #define PC 15

  The register number of the program counter.
- #define CPSR 16

  The register number of the current program status register.
- #define MASK_FIRST_4 0xFFFFFFF

  A mask which removes the first 4 bits when used with bitwise and.
- #define MASK_FIRST_6 0x3FFFFFF

  A mask which removes the first 6 bits when used with bitwise and.
- #define MASK_FIRST_8 0xFFFFFF

  A mask which removes the first 8 bits when used with bitwise and.
- #define GPIO_ACCESS_START 0x20200000

  The first memory address for accessing GPIO pins.
- #define GPIO_ACCESS_SIZE 12

  The number of bytes allocated for accessing GPIO pins.
- #define GPIO_CLEAR_START 0x20200028

  The first memory address for clearing GPIO pins.
- #define GPIO_CLEAR_SIZE 4

  The number of bytes allocated for clearing GPIO pins.
- #define GPIO_SET_START 0x2020001C

  The first memory address for setting GPIO pins.
- #define GPIO_SET_SIZE 4

  The number of bytes allocated for setting GPIO pins.
- #define COMPLIANT_MODE true

  A setting which determines the format of output.

Typedefs

- typedef uint8_t byte_t

  A type alias for a byte (8 bits).
- typedef int8_t reg_address_t

  A type alias for a register number (supports up to 2^ 8 registers).
- typedef uint16_t address_t

  A type alias for a memory address (supports up to 2^ 16 addresses).
- typedef uint32_t word_t

  A type alias for a word (32 bits).

Enumerations

- enum condition_t {
  EQ = 0, NE = 1, GE = 0xA, LT = 0xB,
  GT = 0xC, LE = 0xD, AL = 0xE }

  An enum that identifies the type of condition.
- enum instruction_type_t {
  DPI, MUL, SDT, BRA,
  ZER, NUL }

  An enum that identifies the format of the instruction.
- enum shift_t { LSL = 0, LSR = 1, ASR = 2, ROR = 3 }

  An enum used for defining the type of shift for the shifter to use.

- enum opcode_t {
  AND = 0x0, EOR = 0x1, SUB = 0x2, RSB = 0x3,
  ADD = 0x4, TST = 0x8, TEQ = 0x9, CMP = 0xA,
  ORR = 0xC, MOV = 0xD }

  An enum used for defining the opcode.
- enum cpsr_flags_t { N = 0x8, Z = 0x4, C = 0x2, V = 0x1 }

  An enum used for retrieving individual flag bits from CPSR register.

4.6.1 Detailed Description

Definition of useful constants and type aliases.

4.6.2 Macro Definition Documentation

4.6.2.1 #define COMPLIANT_MODE true

A setting which determines the format of output.

- Using COMPLIANT_MODE will print to stdout in the exact format required by test cases. Only registers and memory are printed. Errors are printed to stdout.
- Otherwise, a much more detailed ouput will be printed, including details on instructions. Formatting is improved. Errors are printed to stderr. The recommended setting is false.

4.6.2.2 #define CPSR 16

The register number of the current program status register.

4.6.2.3 #define GPIO_ACCESS_SIZE 12

The number of bytes allocated for accessing GPIO pins.

4.6.2.4 #define GPIO_ACCESS_START 0x20200000

The first memory address for accessing GPIO pins.

4.6.2.5 #define GPIO_CLEAR_SIZE 4

The number of bytes allocated for clearing GPIO pins.

4.6.2.6 #define GPIO_CLEAR_START 0x20200028

The first memory address for clearing GPIO pins.

4.6.2.7 #define GPIO_SET_SIZE 4

The number of bytes allocated for setting GPIO pins.

4.6.2.8 #define GPIO_SET_START 0x2020001C

The first memory address for setting GPIO pins.

4.6.2.9 #define MASK_FIRST_4 0xFFFFFFF

A mask which removes the first 4 bits when used with bitwise and.

4.6.2.10 #define MASK_FIRST_6 0x3FFFFFF

A mask which removes the first 6 bits when used with bitwise and.

4.6.2.11 #define MASK_FIRST_8 0xFFFFFF

A mask which removes the first 8 bits when used with bitwise and.

4.6.2.12 #define NUM_ADDRESSES 65536

The total number of memory addresses.

4.6.2.13 #define NUM_REGISTERS 17

The total number of registers.

4.6.2.14 #define PC 15

The register number of the program counter.

4.6.2.15 #define WORD_SIZE 32

The architecture word size.

4.6.3 Typedef Documentation

4.6.3.1 typedef uint16_t address_t

A type alias for a memory address (supports up to 2^16 addresses).

4.6.3.2 typedef uint8_t byte_t

A type alias for a byte (8 bits).

4.6.3.3 typedef int8_t reg_address_t

A type alias for a register number (supports up to 2^8 registers).

4.6.3.4 typedef uint32_t word_t

A type alias for a word (32 bits).

4.6.4 Enumeration Type Documentation

4.6.4.1 enum condition_t

An enum that identifies the type of condition.

Enumerator

EQ Equal.

NE Not equal.

GE Greater or equal.

LT Less than.

GT Greater than.

LE Less than or equal to.

AL No condition (always).

4.6.4.2 enum cpsr_flags_t

An enum used for retrieving individual flag bits from CPSR register.

Enumerator

N N (bit 4): the last result was negative.

Z Z (bit 3): the last result was zero.

C C (bit 2): the last result caused a bit to be carried out.

V V (bit 1): the last result overflowed.

4.6.4.3 enum instruction_type_t

An enum that identifies the format of the instruction.

Enumerator

DPI Data processing instruction.

MUL Multiply instruction.

SDT Single data transfer instruction.

BRA Branch instruction.

ZER All zero (STOP) instruction.

NUL NULL (not present) instruction.

4.6.4.4 enum opcode_t

An enum used for defining the opcode.

Enumerator

AND And.

EOR Exclusive or.

SUB Subtract.

RSB Reverse subtract.

ADD Add.

TST And, set flags only.

TEQ Exclusive or, set flags only.

CMP Subtract, set flags only.

ORR Or.

MOV Move.

4.6.4.5 enum shift_t

An enum used for defining the type of shift for the shifter to use.

Enumerator

LSL Logical shift left.

LSR Logical shift right.

ASR Arithmetic shift right.

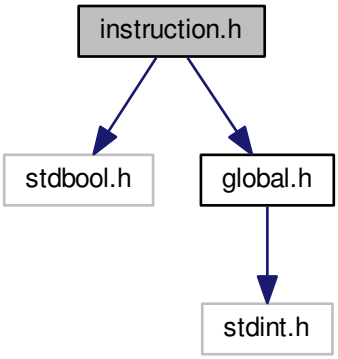
ROR Rotate right.

4.7 instruction.h File Reference

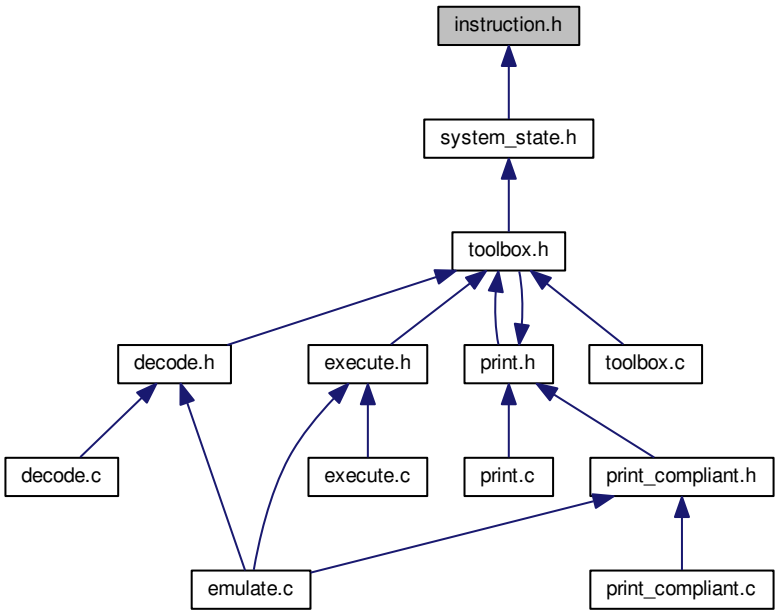
A header to define the instruction_t type.

```
#include <stdbool.h>
#include "global.h"
```

Include dependency graph for instruction.h:



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



Data Structures

- struct instruction_t

  A struct that holds information about a decoded instruction.

4.7.1 Detailed Description

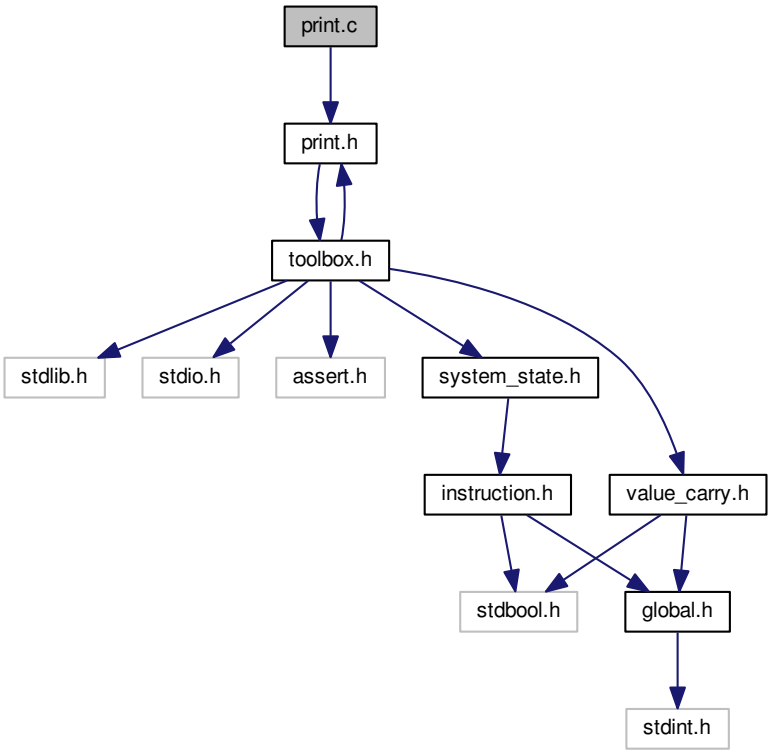
A header to define the instruction_t type.

4.8 print.c File Reference

Functions for printing system details to standard output.

```
#include "print.h"
```

Include dependency graph for print.c:



Functions

- void print_array (byte_t *memory, size_t bytes_to_print)
  Prints a given number of bytes, from an array of bytes.
- void print_system_state (system_state_t *machine)
  Prints system state details.
- void print_registers (system_state_t *machine)
  Prints the values of registers.
- void print_memory (system_state_t *machine)
  Prints any non-zero words from memory.
- void print_decoded_instruction (system_state_t *machine)
  Prints details for the decoded instruction.
- void print_instruction (instruction_t *instruction)
  Prints details for the instruction.
- void print_fetched_instruction (system_state_t *machine)
  Prints the fetched instruction, if present.
- void print_value (word_t value)
  Prints a value for debugging, in binary, hex and 2's complement.
- long twos_complement_to_long (word_t value)
  Converts a signed 2's complement word to a sign long.
- void print_binary_value (word_t value)
  Prints the padded binary representation of value.
- char * get_cond (condition_t cond)
  Returns the string representing the condition type.
- char * get_opcode (opcode_t operation)
  Returns the string representing the opcode.
- char * get_shift (shift_t shift)
  Returns the string representing the shift type.

4.8.1 Detailed Description

Functions for printing system details to standard output.

4.8.2 Function Documentation

4.8.2.1 char* get_cond (condition_t *cond*)

Returns the string representing the condition type.

Parameters

cond	The condition type.

Returns

The string of the condition type for printing.

4.8.2.2 char* get_opcode (opcode_t *operation*)

Returns the string representing the opcode.

Parameters

operation	The opcode.

Returns

The string of the opcode for printing.

4.8.2.3 char* get_shift (shift_t *shift*)

Returns the string representing the shift type.

Parameters

operation	The type of shift.

Returns

The string of the type of shift for printing.

4.8.2.4 void print_array (byte_t * *memory,* size_t *bytes_to_print*)

Prints a given number of bytes, from an array of bytes.

Prints a given number bytes from memory, starting from address 0. Lines are broken every word (4 bytes). Useful for debugging.

Parameters

memory	An array of bytes to print.
bytes_to_print	The number of bytes to print (from 0).

4.8.2.5 void print_binary_value (word_t *value*)

Prints the padded binary representation of value.

Prints WORD_SIZE bits.

Parameters

value	The word for printing.

4.8.2.6 void print_decoded_instruction (system_state_t * machine)

Prints details for the decoded instruction.

Prints the type of the instruction, and any details required:

- For branch instructions, prints the condition and the offset.
- For multiply instructions, prints the condition, flags and registers.
- For data processing instructions, prints the condition, flags, opcodes, operands, and shift information.
- For single data transfer instructions, prints flags, registers and offset.

Parameters

machine	The current system state.

4.8.2.7 void print_fetched_instruction (system_state_t * machine)

Prints the fetched instruction, if present.

Parameters

machine	The current system state.

4.8.2.8 void print_instruction (instruction_t * instruction)

Prints details for the instruction.

Prints the type of the instruction, and any details required:

- For branch instructions, prints the condition and the offset.
- For multiply instructions, prints the condition, flags and registers.
- For data processing instructions, prints the condition, flags, opcodes, operands, and shift information.
- For single data transfer instructions, prints flags, registers and offset.

Parameters

instruction	The instruction.

4.8.2.9 void print_memory (system_state_t * machine)

Prints any non-zero words from memory.

Prints any non-zero words from memory and their addresses.

Parameters

machine	The current system state.

4.8.2.10 void print_registers (system_state_t * *machine*)

Prints the values of registers.

Prints the values held in each of the NUM_REGISTERS registers.

Parameters

machine	The current system state.

4.8.2.11 void print_system_state (system_state_t * *machine*)

Prints system state details.

Prints the current system state. Prints all register values, any memory values which are not 0, the decoded instruction and the fetched instruction.

Parameters

machine	The current system state.

4.8.2.12 void print_value (word_t *value*)

Prints a value for debugging, in binary, hex and 2's complement.

Parameters

value	The word to print.

4.8.2.13 long twos_complement_to_long (word_t *value*)

Converts a signed 2's complement word to a sign long.

Parameters

value	The signed 2's complement word to convert.

Returns

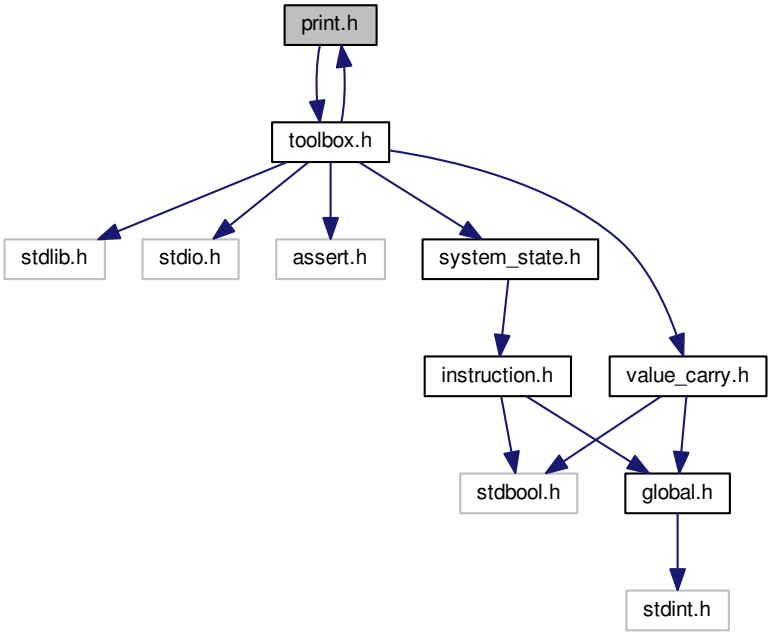
The signed long representation of the word.

4.9 print.h File Reference

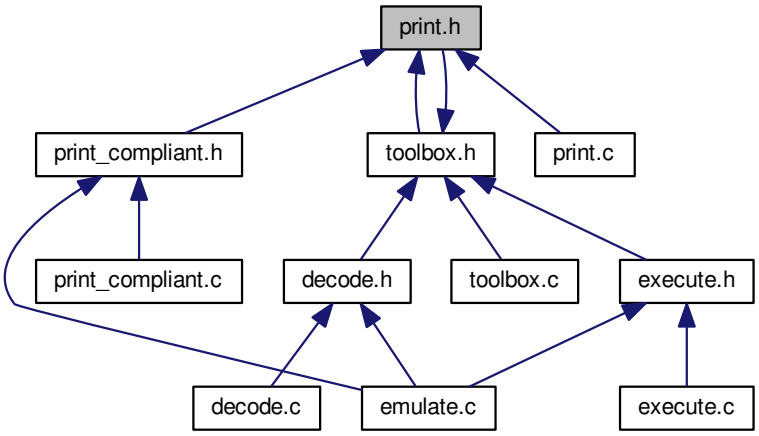
Header file for print.c.

```
#include "toolbox.h"
```

Include dependency graph for print.h:



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



Functions

- void print_array (byte_t *memory, size_t bytes_to_print)

  Prints a given number of bytes, from an array of bytes.
- void print_system_state (system_state_t *machine)

  Prints system state details.
- void print_registers (system_state_t *machine)

  Prints the values of registers.
- void print_memory (system_state_t *machine)

  Prints any non-zero words from memory.
- void print_decoded_instruction (system_state_t *machine)

  Prints details for the decoded instruction.
- void print_fetched_instruction (system_state_t *machine)

  Prints the fetched instruction, if present.
- void print_instruction (instruction_t *instruction)

  Prints details for the instruction.
- void print_value (word_t value)

  Prints a value for debugging, in binary, hex and 2's complement.
- char * get_cond (condition_t cond)

  Returns the string representing the condition type.
- char * get_opcode (opcode_t operation)

  Returns the string representing the opcode.
- char * get_shift (shift_t shift)

  Returns the string representing the shift type.
- long twos_complement_to_long (word_t value)

  Converts a signed 2's complement word to a sign long.
- void print_binary_value (word_t value)

  Prints the padded binary representation of value.

4.9.1 Detailed Description

Header file for print.c.

4.9.2 Function Documentation

4.9.2.1 char* get_cond (condition_t *cond*)

Returns the string representing the condition type.

Parameters

cond	The condition type.

Returns

The string of the condition type for printing.

4.9.2.2 char* get_opcode (opcode_t *operation*)

Returns the string representing the opcode.

Parameters

operation	The opcode.

Returns

The string of the opcode for printing.

4.9.2.3 char* get_shift (shift_t *shift*)

Returns the string representing the shift type.

Parameters

operation	The type of shift.

Returns

The string of the type of shift for printing.

4.9.2.4 void print_array (byte_t * *memory*, size_t *bytes_to_print*)

Prints a given number of bytes, from an array of bytes.

Prints a given number bytes from memory, starting from address 0. Lines are broken every word (4 bytes). Useful for debugging.

Parameters

memory	An array of bytes to print.
bytes_to_print	The number of bytes to print (from 0).

4.9.2.5 void print_binary_value (word_t *value*)

Prints the padded binary representation of value.

Prints WORD_SIZE bits.

Parameters

value	The word for printing.

4.9.2.6 void print_decoded_instruction (system_state_t * machine)

Prints details for the decoded instruction.

Prints the type of the instruction, and any details required:

- For branch instructions, prints the condition and the offset.
- For multiply instructions, prints the condition, flags and registers.
- For data processing instructions, prints the condition, flags, opcodes, operands, and shift information.
- For single data transfer instructions, prints flags, registers and offset.

Parameters

machine	The current system state.

4.9.2.7 void print_fetched_instruction (system_state_t * machine)

Prints the fetched instruction, if present.

Parameters

machine	The current system state.

4.9.2.8 void print_instruction (instruction_t * instruction)

Prints details for the instruction.

Prints the type of the instruction, and any details required:

- For branch instructions, prints the condition and the offset.
- For multiply instructions, prints the condition, flags and registers.
- For data processing instructions, prints the condition, flags, opcodes, operands, and shift information.
- For single data transfer instructions, prints flags, registers and offset.

Parameters

instruction	The instruction.

4.9.2.9 void print_memory (system_state_t * machine)

Prints any non-zero words from memory.
Prints any non-zero words from memory and their addresses.

Parameters

machine	The current system state.

4.9.2.10 void print_registers (system_state_t * machine)

Prints the values of registers.

Prints the values held in each of the NUM_REGISTERS registers.

Parameters

machine	The current system state.

4.9.2.11 void print_system_state (system_state_t * machine)

Prints system state details.

Prints the current system state. Prints all register values, any memory values which are not 0, the decoded instruction and the fetched instruction.

Parameters

machine	The current system state.

4.9.2.12 void print_value (word_t value)

Prints a value for debugging, in binary, hex and 2's complement.

Parameters

value	The word to print.

4.9.2.13 long twos_complement_to_long (word_t value)

Converts a signed 2's complement word to a sign long.

Parameters

value	The signed 2's complement word to convert.

Returns

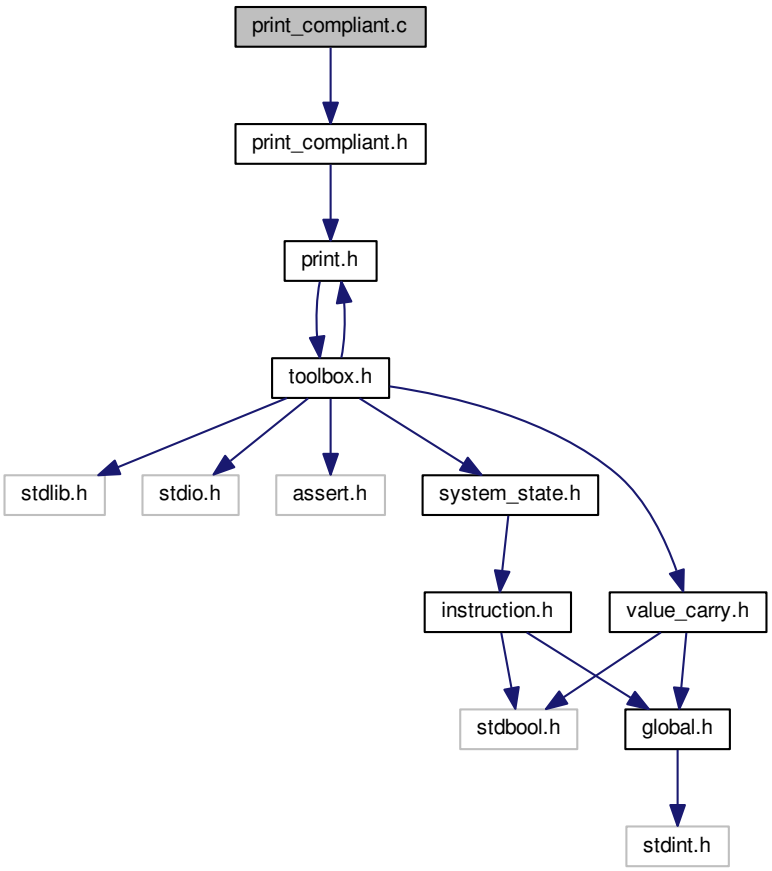
The signed long representation of the word.

4.10 print_compliant.c File Reference

Functions for printing system details to match test cases.

```
#include "print_compliant.h"
```

Include dependency graph for print_compliant.c:



Functions

- void print_system_state_compliant (system_state_t *machine)

  Prints system state details for test cases.
- void print_registers_compliant (system_state_t *machine)

  Prints the values of the registers of the machine for test cases.
- void print_memory_compliant (system_state_t *machine)

  Prints non-zero memory entries for test cases.
- void print_value_compliant (word_t value)

  Prints a value for test cases, in hex and two's complement.

4.10.1 Detailed Description

Functions for printing system details to match test cases.

4.10.2 Function Documentation

4.10.2.1 void print_memory_compliant (system_state_t * machine)

Prints non-zero memory entries for test cases.

Parameters

machine	The current system state.

4.10.2.2 void print_registers_compliant (system_state_t * machine)

Prints the values of the registers of the machine for test cases.

Parameters

machine	The current system state.

4.10.2.3 void print_system_state_compliant (system_state_t * machine)

Prints system state details for test cases.

Prints the current system state. Prints all register values, any memory values which are not 0. Prints in test case format.

Parameters

machine	The current system state.

4.10.2.4 void print_value_compliant (word_t value)

Prints a value for test cases, in hex and two's complement.

Parameters

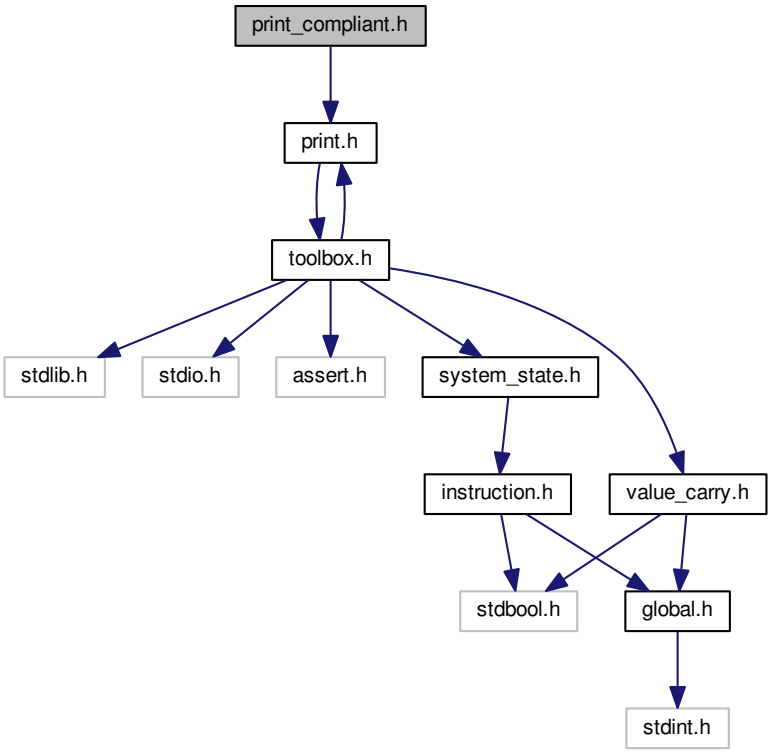
value	The word to print.

4.11 print_compliant.h File Reference

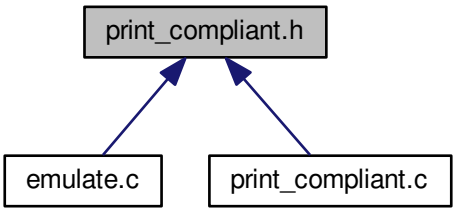
Header file for print_compliant.c.

```
#include "print.h"
```

Include dependency graph for print_compliant.h:



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



Functions

- void print_system_state_compliant (system_state_t *machine)

  Prints system state details for test cases.
- void print_registers_compliant (system_state_t *machine)

Prints the values of the registers of the machine for test cases.

- void print_memory_compliant (system_state_t *machine)

  Prints non-zero memory entries for test cases.

- void print_value_compliant (word_t value)

  Prints a value for test cases, in hex and two's complement.

4.11.1 Detailed Description

Header file for print_compliant.c.

4.11.2 Function Documentation

4.11.2.1 void print_memory_compliant (system_state_t * *machine*)

Prints non-zero memory entries for test cases.

Parameters

machine	The current system state.

4.11.2.2 void print_registers_compliant (system_state_t * *machine*)

Prints the values of the registers of the machine for test cases.

Parameters

machine	The current system state.

4.11.2.3 void print_system_state_compliant (system_state_t * *machine*)

Prints system state details for test cases.

Prints the current system state. Prints all register values, any memory values which are not 0. Prints in test case format.

Parameters

machine	The current system state.

4.11.2.4 void print_value_compliant (word_t *value*)

Prints a value for test cases, in hex and two's complement.

Parameters

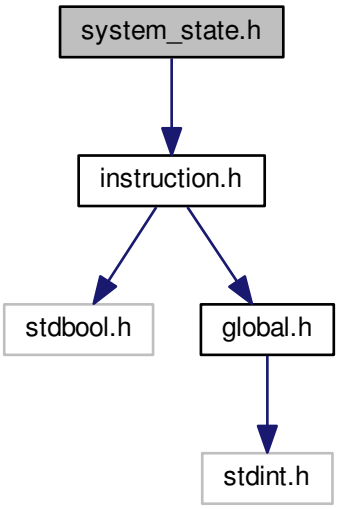
value	The word to print.

4.12 system_state.h File Reference

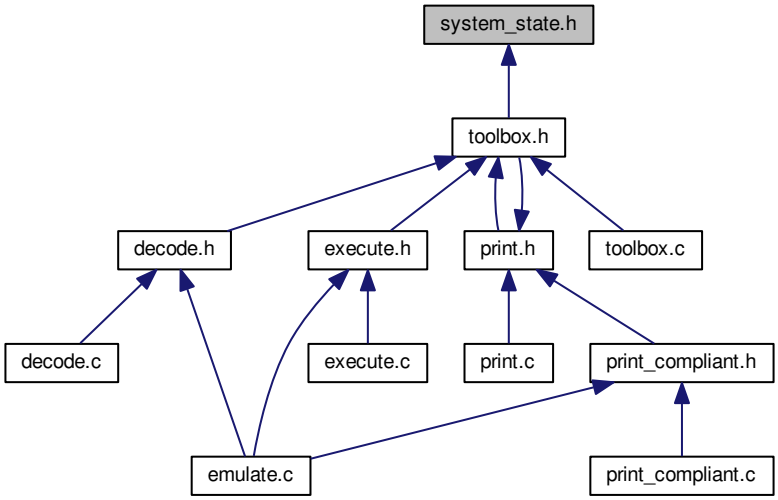
A header to define the system_state_t type.

```
#include "instruction.h"
```

Include dependency graph for system_state.h:



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



Data Structures

- struct system_state_t

  A struct that holds information about the current system state.

4.12.1 Detailed Description

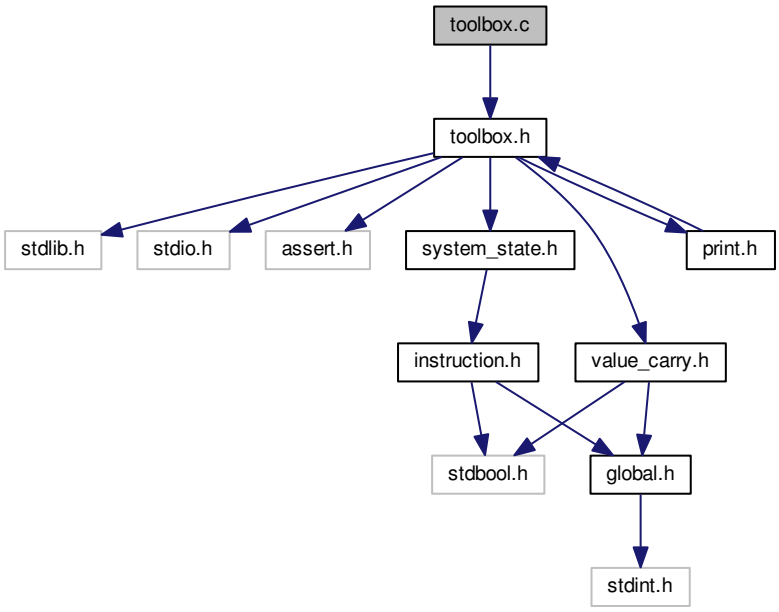
A header to define the system_state_t type.

4.13 toolbox.c File Reference

Miscellaneous functions that are widely used throughout the code.

```
#include "toolbox.h"
```

Include dependency graph for toolbox.c:



Functions

- void load_file (char *fname, byte_t *memory)

  Loads a binary file into the memory.
- void exit_program (system_state_t *machine)

  Exits gracefully.
- word_t get_word (system_state_t *machine, uint32_t mem_address)

  Gets a memory word from a given address.
- word_t get_word_compliant (system_state_t *machine, address_t mem_address)

  Gets a memory word from a given address (for printing only).
- void set_word (system_state_t *machine, uint32_t mem_address, word_t word)

  Writes a word to memory at a given address.
- word_t negate (word_t value)

  Negates a two's complement value.
- bool is_negative (word_t value)

  Returns true iff two's complement value is negative.
- word_t absolute (word_t value)

  Returns absolute two's complement value.
- value_carry_t * shifter (shift_t type, word_t shift_amount, word_t value)

  Shifts a value and returns a pointer.

4.13.1 Detailed Description

Miscellaneous functions that are widely used throughout the code.

4.13.2 Function Documentation

4.13.2.1 word_t absolute (word_t *value*)

Returns absolute two's complement value.

Parameters

value	A two's complement word.

Returns

The absolute value of the provided word in two's complement.

4.13.2.2 void exit_program (system_state_t * *machine*)

Exits gracefully.

Prints the current system state, frees allocated memory and exits with a failure. To be used in the case of an error which cannot be recovered from.

Parameters

machine	The current system state.

4.13.2.3 word_t get_word (system_state_t * *machine,* uint32_t *mem_address*)

Gets a memory word from a given address.

- If GPIO access adddress is read, prints a message to stdout.
- If another out of bounds address is read, prints an error.

Parameters

machine	The current system state.
mem_address	The memory address to be read from.

Returns

The word at the given memory address in the current system state.

4.13.2.4 word_t get_word_compliant (system_state_t * *machine,* address_t *mem_address*)

Gets a memory word from a given address (for printing only).

For use in compliant printing. Gets the word in little endian order.

Parameters

machine	The current system state.
mem_address	The memory address to be read from.

Returns

The word at the given memory address in the current system state.

4.13.2.5 bool is_negative (word_t *value*)

Returns true iff two's complement value is negative.

Parameters

value	A two's complement word to check sign of.

Returns

True iff provided value is negative in two's complement.

4.13.2.6 void load_file (char * *fname,* byte_t * *memory*)

Loads a binary file into the memory.

Writes the contents of the provided binary object code file to the memory, starting at the provided location. Returns an error message and exits if the file cannot be opened or cannot be read.

Parameters

fname	The filename containing object code to be loaded.
memory	A pointer to the first byte of memory to be written to.

4.13.2.7 word_t negate (word_t *value*)

Negates a two's complement value.

Parameters

value	The word to be negated.

Returns

The negated word.

4.13.2.8 void set_word (system_state_t * *machine*, uint32_t *mem_address*, word_t *word*)

Writes a word to memory at a given address.

- If GPIO access adddress is written to, prints a message to stdout.
- If GPIO clear or set adddress is written to, prints a message to stdout.
- If another out of bounds address is read, prints an error.

Parameters

machine	The current system state.
mem_address	The memory address to write to.
word	The word to write to memory.

4.13.2.9 value_carry_t* shifter (shift_t *type*, word_t *shift_amount*, word_t *value*)

Shifts a value and returns a pointer.

Parameters

type	The type of shift to use.
shift_amount	The amount to shift by.
value	The value to shift.

Returns

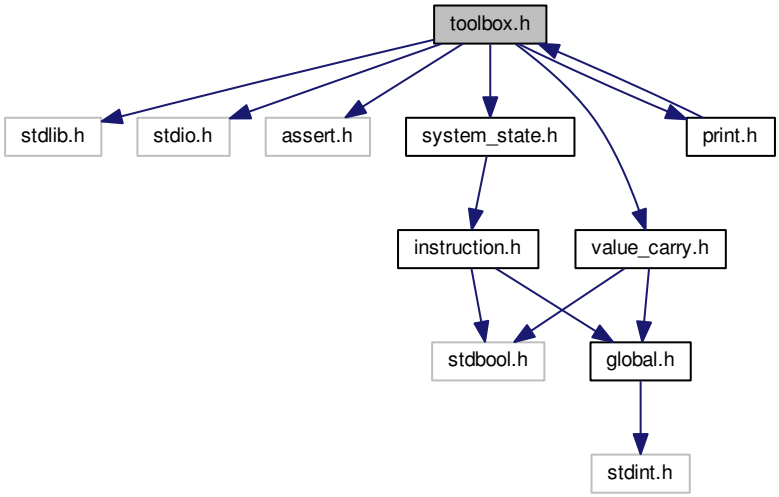
The pointer to the shifted value.

4.14 toolbox.h File Reference

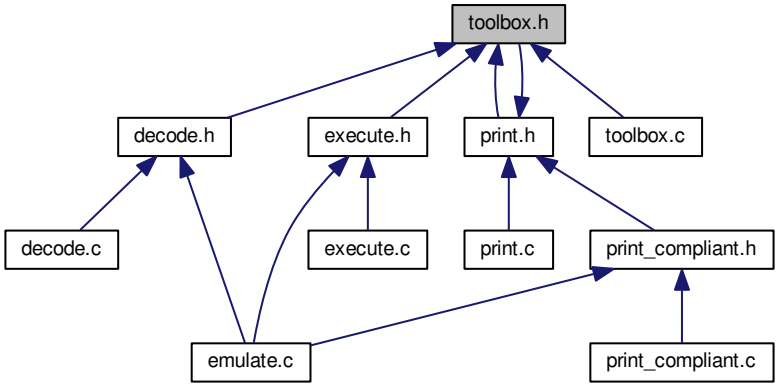
Header file for toolbox.c.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "system_state.h"
#include "value_carry.h"
#include "print.h"
```

Include dependency graph for toolbox.h:



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



Functions

- void load_file (char *fname, byte_t *memory)

  Loads a binary file into the memory.
- void exit_program (system_state_t *machine)

  Exits gracefully.
- word_t get_word (system_state_t *machine, uint32_t mem_address)

  Gets a memory word from a given address.
- word_t get_word_compliant (system_state_t *machine, address_t mem_address)

Gets a memory word from a given address (for printing only).

- void set_word (system_state_t *machine, uint32_t mem_address, word_t word)

  Writes a word to memory at a given address.
- word_t negate (word_t value)

  Negates a two's complement value.
- bool is_negative (word_t value)

  Returns true iff two's complement value is negative.
- word_t absolute (word_t value)

  Returns absolute two's complement value.
- value_carry_t * shifter (shift_t type, word_t shift_amount, word_t value)

  Shifts a value and returns a pointer.

4.14.1 Detailed Description

Header file for toolbox.c.

4.14.2 Function Documentation

4.14.2.1 word_t absolute (word_t *value*)

Returns absolute two's complement value.

Parameters

value	A two's complement word.

Returns

The absolute value of the provided word in two's complement.

4.14.2.2 void exit_program (system_state_t * *machine*)

Exits gracefully.

Prints the current system state, frees allocated memory and exits with a failure. To be used in the case of an error which cannot be recovered from.

Parameters

machine	The current system state.

4.14.2.3 word_t get_word (system_state_t * *machine*, uint32_t *mem_address*)

Gets a memory word from a given address.

- If GPIO access adddress is read, prints a message to stdout.
- If another out of bounds address is read, prints an error.

Parameters

machine	The current system state.
mem_address	The memory address to be read from.

Returns

The word at the given memory address in the current system state.

4.14.2.4 word_t get_word_compliant (system_state_t * *machine,* address_t *mem_address*)

Gets a memory word from a given address (for printing only).

For use in compliant printing. Gets the word in little endian order.

Parameters

machine	The current system state.
mem_address	The memory address to be read from.

Returns

The word at the given memory address in the current system state.

4.14.2.5 bool is_negative (word_t *value*)

Returns true iff two's complement value is negative.

Parameters

value	A two's complement word to check sign of.

Returns

True iff provided value is negative in two's complement.

4.14.2.6 void load_file (char * *fname,* byte_t * *memory*)

Loads a binary file into the memory.

Writes the contents of the provided binary object code file to the memory, starting at the provided location. Returns an error message and exits if the file cannot be opened or cannot be read.

Parameters

fname	The filename containing object code to be loaded.
memory	A pointer to the first byte of memory to be written to.

4.14.2.7 word_t negate (word_t *value*)

Negates a two's complement value.

Parameters

value	The word to be negated.

Returns

The negated word.

4.14.2.8 void set_word (system_state_t * *machine,* uint32_t *mem_address,* word_t *word*)

Writes a word to memory at a given address.

- If GPIO access adddress is written to, prints a message to stdout.
- If GPIO clear or set adddress is written to, prints a message to stdout.
- If another out of bounds address is read, prints an error.

Parameters

machine	The current system state.
mem_address	The memory address to write to.
word	The word to write to memory.

4.14.2.9 value_carry_t* shifter (shift_t *type,* word_t *shift_amount,* word_t *value*)

Shifts a value and returns a pointer.

Parameters

type	The type of shift to use.
shift_amount	The amount to shift by.
value	The value to shift.

Returns

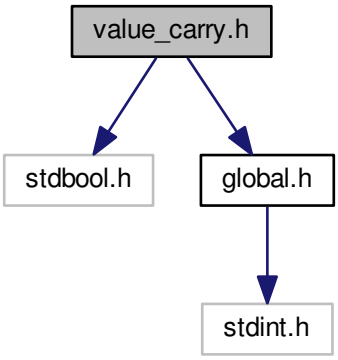
The pointer to the shifted value.

4.15 value_carry.h File Reference

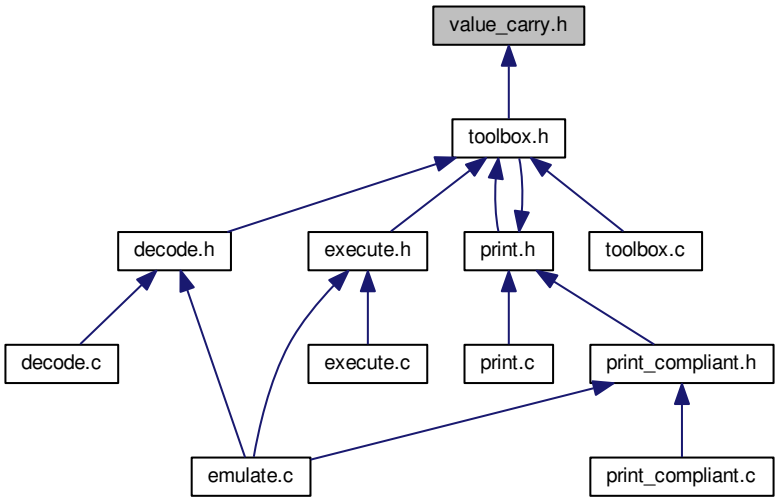
A header to define the value_carry_t type.

```
#include <stdbool.h>
#include "global.h"
```

Include dependency graph for value_carry.h:



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



Data Structures

- struct value_carry_t

  A struct that has a value and a carry.

4.15.1 Detailed Description

A header to define the value_carry_t type.

Index