

## ARM11 Emulator

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	instruction_t Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Field Documentation . . . . .	6
3.1.2.1	cond . . . . .	6
3.1.2.2	flag_0 . . . . .	6
3.1.2.3	flag_1 . . . . .	6
3.1.2.4	flag_2 . . . . .	6
3.1.2.5	flag_3 . . . . .	6
3.1.2.6	immediate_value . . . . .	6
3.1.2.7	operation . . . . .	6
3.1.2.8	rd . . . . .	6
3.1.2.9	rm . . . . .	6
3.1.2.10	rn . . . . .	7
3.1.2.11	rs . . . . .	7
3.1.2.12	shift_amount . . . . .	7
3.1.2.13	shift_type . . . . .	7
3.1.2.14	type . . . . .	7

3.2	system_state_t Struct Reference	7
3.2.1	Detailed Description	8
3.2.2	Field Documentation	8
3.2.2.1	decoded_instruction	8
3.2.2.2	fetches_instruction	8
3.2.2.3	has_fetched_instruction	8
3.2.2.4	memory	8
3.2.2.5	registers	8
3.3	value_carry_t Struct Reference	9
3.3.1	Detailed Description	9
3.3.2	Field Documentation	9
3.3.2.1	carry	9
3.3.2.2	value	9
<b>4</b>	<b>File Documentation</b>	<b>11</b>
4.1	decode.c File Reference	11
4.1.1	Detailed Description	12
4.1.2	Function Documentation	12
4.1.2.1	branch(system_state_t *machine)	12
4.1.2.2	data_processing(system_state_t *machine)	12
4.1.2.3	decode_instruction(system_state_t *machine)	13
4.1.2.4	halt(system_state_t *machine)	13
4.1.2.5	multiply(system_state_t *machine)	13
4.1.2.6	single_data_transfer(system_state_t *machine)	13
4.2	decode.h File Reference	14
4.2.1	Detailed Description	16
4.2.2	Function Documentation	16
4.2.2.1	branch(system_state_t *machine)	16
4.2.2.2	data_processing(system_state_t *machine)	16
4.2.2.3	decode_instruction(system_state_t *machine)	17
4.2.2.4	halt(system_state_t *machine)	17

4.2.2.5	<code>multiply(system_state_t *machine)</code>	17
4.2.2.6	<code>single_data_transfer(system_state_t *machine)</code>	17
4.3	<code>emulate.c</code> File Reference	18
4.3.1	Detailed Description	19
4.3.2	Function Documentation	19
4.3.2.1	<code>main(int argc, char **argv)</code>	19
4.4	<code>execute.c</code> File Reference	20
4.4.1	Detailed Description	20
4.4.2	Function Documentation	21
4.4.2.1	<code>condition(system_state_t *machine)</code>	21
4.4.2.2	<code>execute(system_state_t *machine)</code>	21
4.4.2.3	<code>execute_branch(system_state_t *machine)</code>	21
4.4.2.4	<code>execute_dpi(system_state_t *machine)</code>	21
4.4.2.5	<code>execute_mul(system_state_t *machine)</code>	22
4.4.2.6	<code>execute_sdt(system_state_t *machine)</code>	22
4.5	<code>execute.h</code> File Reference	22
4.5.1	Detailed Description	23
4.5.2	Function Documentation	23
4.5.2.1	<code>execute(system_state_t *machine)</code>	23
4.5.2.2	<code>execute_branch(system_state_t *machine)</code>	24
4.5.2.3	<code>execute_dpi(system_state_t *machine)</code>	24
4.5.2.4	<code>execute_mul(system_state_t *machine)</code>	24
4.5.2.5	<code>execute_sdt(system_state_t *machine)</code>	24
4.6	<code>global.h</code> File Reference	24
4.6.1	Detailed Description	27
4.6.2	Macro Definition Documentation	27
4.6.2.1	<code>COMPLIANT_MODE</code>	27
4.6.2.2	<code>CPSR</code>	27
4.6.2.3	<code>GPIO_ACCESS_SIZE</code>	27
4.6.2.4	<code>GPIO_ACCESS_START</code>	27

4.6.2.5	<a href="#">GPIO_CLEAR_SIZE</a>	27
4.6.2.6	<a href="#">GPIO_CLEAR_START</a>	27
4.6.2.7	<a href="#">GPIO_SET_SIZE</a>	28
4.6.2.8	<a href="#">GPIO_SET_START</a>	28
4.6.2.9	<a href="#">MASK_FIRST_4</a>	28
4.6.2.10	<a href="#">MASK_FIRST_6</a>	28
4.6.2.11	<a href="#">MASK_FIRST_8</a>	28
4.6.2.12	<a href="#">NUM_ADDRESSES</a>	28
4.6.2.13	<a href="#">NUM_REGISTERS</a>	28
4.6.2.14	<a href="#">PC</a>	28
4.6.2.15	<a href="#">WORD_SIZE</a>	28
4.6.3	<a href="#">Typedef Documentation</a>	28
4.6.3.1	<a href="#">address_t</a>	28
4.6.3.2	<a href="#">byte_t</a>	29
4.6.3.3	<a href="#">reg_address_t</a>	29
4.6.3.4	<a href="#">word_t</a>	29
4.6.4	<a href="#">Enumeration Type Documentation</a>	29
4.6.4.1	<a href="#">condition_t</a>	29
4.6.4.2	<a href="#">cpsr_flags_t</a>	29
4.6.4.3	<a href="#">instruction_type_t</a>	30
4.6.4.4	<a href="#">opcode_t</a>	30
4.6.4.5	<a href="#">shift_t</a>	30
4.7	<a href="#">instruction.h File Reference</a>	31
4.7.1	<a href="#">Detailed Description</a>	32
4.8	<a href="#">print.c File Reference</a>	32
4.8.1	<a href="#">Detailed Description</a>	33
4.8.2	<a href="#">Function Documentation</a>	33
4.8.2.1	<a href="#">get_cond(condition_t cond)</a>	33
4.8.2.2	<a href="#">get_opcode(opcode_t operation)</a>	34
4.8.2.3	<a href="#">get_shift(shift_t shift)</a>	34

4.8.2.4	<a href="#">print_array(byte_t *memory, size_t bytes_to_print)</a>	34
4.8.2.5	<a href="#">print_binary_value(word_t value)</a>	34
4.8.2.6	<a href="#">print_decoded_instruction(system_state_t *machine)</a>	35
4.8.2.7	<a href="#">print_fetched_instruction(system_state_t *machine)</a>	35
4.8.2.8	<a href="#">print_instruction(instruction_t *instruction)</a>	35
4.8.2.9	<a href="#">print_memory(system_state_t *machine)</a>	35
4.8.2.10	<a href="#">print_registers(system_state_t *machine)</a>	36
4.8.2.11	<a href="#">print_system_state(system_state_t *machine)</a>	36
4.8.2.12	<a href="#">print_value(word_t value)</a>	36
4.8.2.13	<a href="#">twos_complement_to_long(word_t value)</a>	36
4.9	<a href="#">print.h File Reference</a>	37
4.9.1	<a href="#">Detailed Description</a>	38
4.9.2	<a href="#">Function Documentation</a>	38
4.9.2.1	<a href="#">get_cond(condition_t cond)</a>	38
4.9.2.2	<a href="#">get_opcode(opcode_t operation)</a>	39
4.9.2.3	<a href="#">get_shift(shift_t shift)</a>	39
4.9.2.4	<a href="#">print_array(byte_t *memory, size_t bytes_to_print)</a>	39
4.9.2.5	<a href="#">print_binary_value(word_t value)</a>	39
4.9.2.6	<a href="#">print_decoded_instruction(system_state_t *machine)</a>	40
4.9.2.7	<a href="#">print_fetched_instruction(system_state_t *machine)</a>	40
4.9.2.8	<a href="#">print_instruction(instruction_t *instruction)</a>	40
4.9.2.9	<a href="#">print_memory(system_state_t *machine)</a>	40
4.9.2.10	<a href="#">print_registers(system_state_t *machine)</a>	41
4.9.2.11	<a href="#">print_system_state(system_state_t *machine)</a>	41
4.9.2.12	<a href="#">print_value(word_t value)</a>	41
4.9.2.13	<a href="#">twos_complement_to_long(word_t value)</a>	41
4.10	<a href="#">print_compliant.c File Reference</a>	42
4.10.1	<a href="#">Detailed Description</a>	42
4.10.2	<a href="#">Function Documentation</a>	43
4.10.2.1	<a href="#">print_memory_compliant(system_state_t *machine)</a>	43

4.10.2.2	<code>print_registers_compliant(system_state_t *machine)</code>	43
4.10.2.3	<code>print_system_state_compliant(system_state_t *machine)</code>	43
4.10.2.4	<code>print_value_compliant(word_t value)</code>	43
4.11	<code>print_compliant.h</code> File Reference	43
4.11.1	Detailed Description	45
4.11.2	Function Documentation	45
4.11.2.1	<code>print_memory_compliant(system_state_t *machine)</code>	45
4.11.2.2	<code>print_registers_compliant(system_state_t *machine)</code>	45
4.11.2.3	<code>print_system_state_compliant(system_state_t *machine)</code>	45
4.11.2.4	<code>print_value_compliant(word_t value)</code>	45
4.12	<code>system_state.h</code> File Reference	46
4.12.1	Detailed Description	47
4.13	<code>toolbox.c</code> File Reference	47
4.13.1	Detailed Description	48
4.13.2	Function Documentation	49
4.13.2.1	<code>absolute(word_t value)</code>	49
4.13.2.2	<code>exit_program(system_state_t *machine)</code>	49
4.13.2.3	<code>get_word(system_state_t *machine, uint32_t mem_address)</code>	49
4.13.2.4	<code>get_word_compliant(system_state_t *machine, address_t mem_address)</code>	49
4.13.2.5	<code>is_negative(word_t value)</code>	50
4.13.2.6	<code>load_file(char *fname, byte_t *memory)</code>	50
4.13.2.7	<code>negate(word_t value)</code>	50
4.13.2.8	<code>set_word(system_state_t *machine, uint32_t mem_address, word_t word)</code>	51
4.13.2.9	<code>shifter(shift_t type, word_t shift_amount, word_t value)</code>	51
4.14	<code>toolbox.h</code> File Reference	51
4.14.1	Detailed Description	53
4.14.2	Function Documentation	53
4.14.2.1	<code>absolute(word_t value)</code>	53
4.14.2.2	<code>exit_program(system_state_t *machine)</code>	53
4.14.2.3	<code>get_word(system_state_t *machine, uint32_t mem_address)</code>	53
4.14.2.4	<code>get_word_compliant(system_state_t *machine, address_t mem_address)</code>	54
4.14.2.5	<code>is_negative(word_t value)</code>	54
4.14.2.6	<code>load_file(char *fname, byte_t *memory)</code>	54
4.14.2.7	<code>negate(word_t value)</code>	55
4.14.2.8	<code>set_word(system_state_t *machine, uint32_t mem_address, word_t word)</code>	55
4.14.2.9	<code>shifter(shift_t type, word_t shift_amount, word_t value)</code>	55
4.15	<code>value_carry.h</code> File Reference	56
4.15.1	Detailed Description	57



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">instruction_t</a>	A struct that holds information about a decoded instruction . . . . .	5
<a href="#">system_state_t</a>	A struct that holds information about the current system state . . . . .	7
<a href="#">value_carry_t</a>	A struct that has a value and a carry . . . . .	9



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">decode.c</a>	Functions for the decode cycle . . . . .	11
<a href="#">decode.h</a>	Header file for <a href="#">decode.c</a> . . . . .	14
<a href="#">emulate.c</a>	The main functionality for the ARM11 emulator . . . . .	18
<a href="#">execute.c</a>	Functions for the execute cycle . . . . .	20
<a href="#">execute.h</a>	Header file for <a href="#">execute.c</a> . . . . .	22
<a href="#">global.h</a>	Definition of useful constants and type aliases . . . . .	24
<a href="#">instruction.h</a>	A header to define the <a href="#">instruction_t</a> type . . . . .	31
<a href="#">print.c</a>	Functions for printing system details to standard output . . . . .	32
<a href="#">print.h</a>	Header file for <a href="#">print.c</a> . . . . .	37
<a href="#">print_compliant.c</a>	Functions for printing system details to match test cases . . . . .	42
<a href="#">print_compliant.h</a>	Header file for <a href="#">print_compliant.c</a> . . . . .	43
<a href="#">system_state.h</a>	A header to define the <a href="#">system_state_t</a> type . . . . .	46
<a href="#">toolbox.c</a>	Miscellaneous functions that are widely used throughout the code . . . . .	47
<a href="#">toolbox.h</a>	Header file for <a href="#">toolbox.c</a> . . . . .	51
<a href="#">value_carry.h</a>	A header to define the <a href="#">value_carry_t</a> type . . . . .	56



## 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:

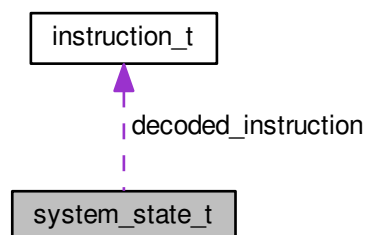
- [instruction.h](#)

## 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

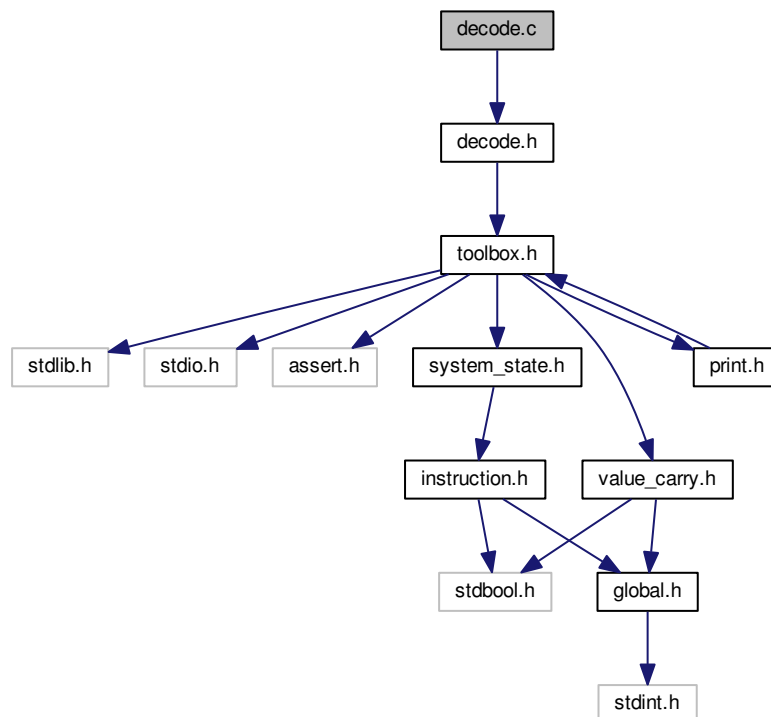
# File Documentation

### 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

<i>machine</i>	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

<i>machine</i>	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

<i>machine</i>	The current system state.
----------------	---------------------------

**4.1.2.4 void halt ( system\_state\_t \* machine )**

Sets decoded\_instruction type to a stop (ZER) instruction.

## Parameters

<i>machine</i>	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

<i>machine</i>	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

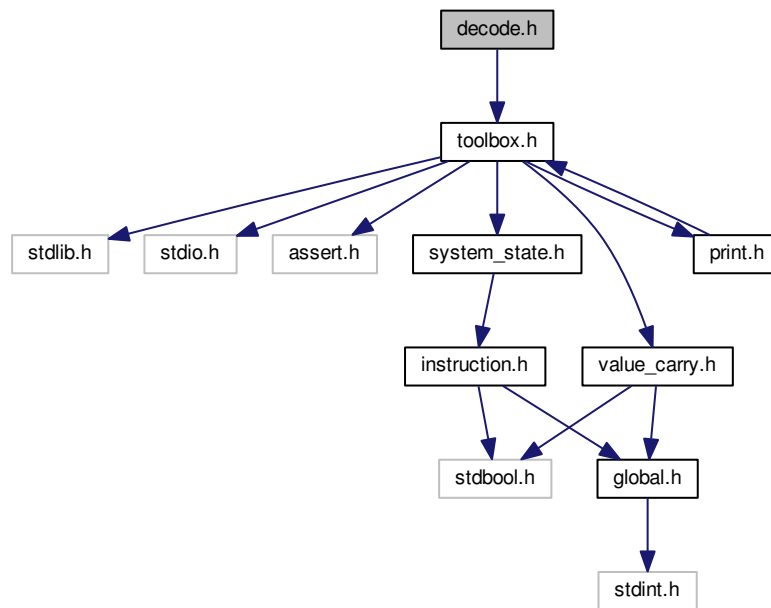
<i>machine</i>	The current system state.
----------------	---------------------------

## 4.2 `decode.h` File Reference

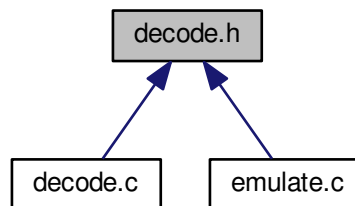
Header file for [decode.c](#).

```
#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

<i>machine</i>	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

<i>machine</i>	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

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.2.2.4 void halt ( system\_state\_t \* machine )

Sets decoded\_instruction type to a stop (ZER) instruction.

##### Parameters

<i>machine</i>	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

<i>machine</i>	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

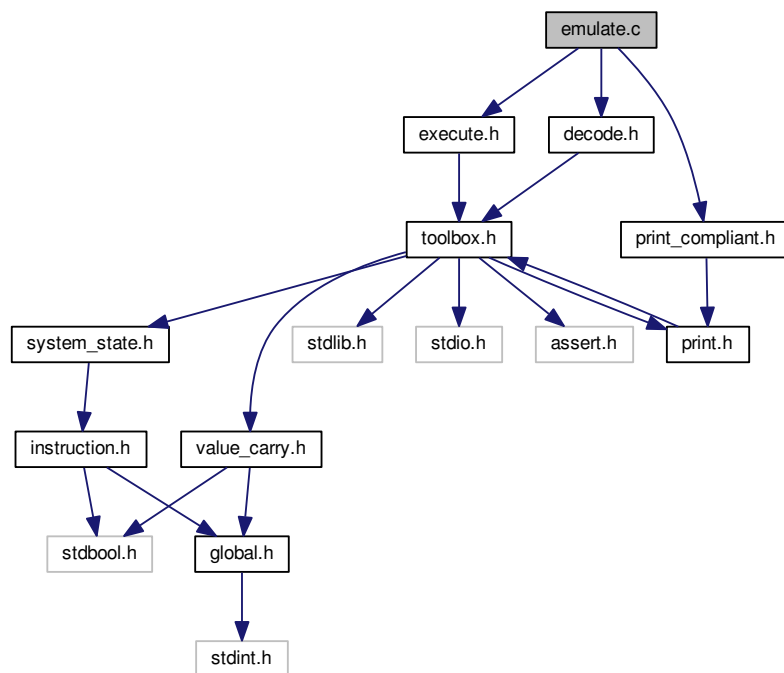
<i>machine</i>	The current system state.
----------------	---------------------------

## 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.

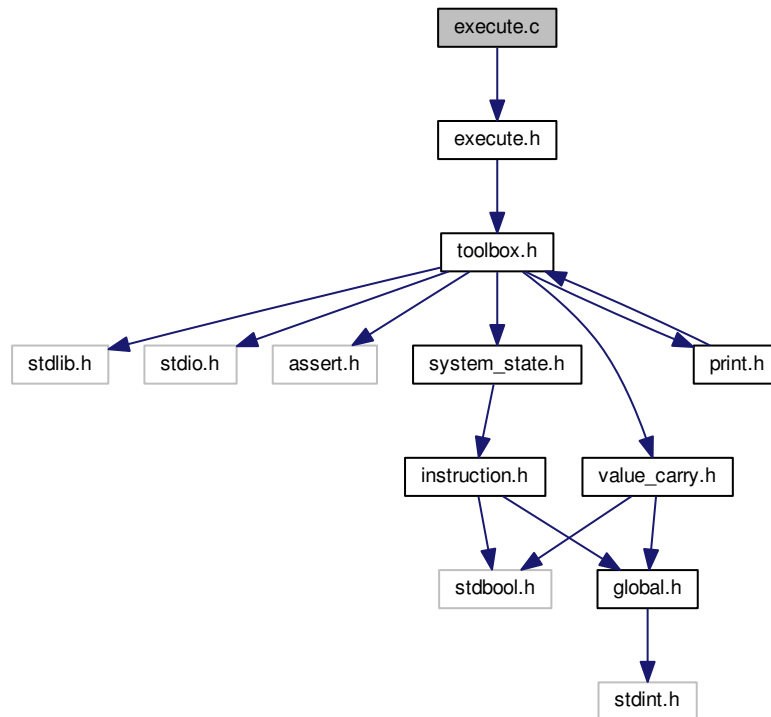
The user must provide a single argument, which is a valid file name for an ARM11 binary object code file. This function emulates the ARM architecture, returning details of the registers and non-zero memory at the end of execution.

## 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

<i>machine</i>	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

<i>machine</i>	The current system state.
----------------	---------------------------

### 4.4.2.3 void execute\_branch ( system\_state\_t \* machine )

Executes a branch instruction.

#### Parameters

<i>machine</i>	The current system state.
----------------	---------------------------

### 4.4.2.4 void execute\_dpi ( system\_state\_t \* machine )

Executes a data processing instruction.

#### Parameters

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.4.2.5 void execute\_mul ( system\_state\_t \* machine )

Executes a multiply instruction.

##### Parameters

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.4.2.6 void execute\_sdt ( system\_state\_t \* machine )

Executes a single data transfer instruction.

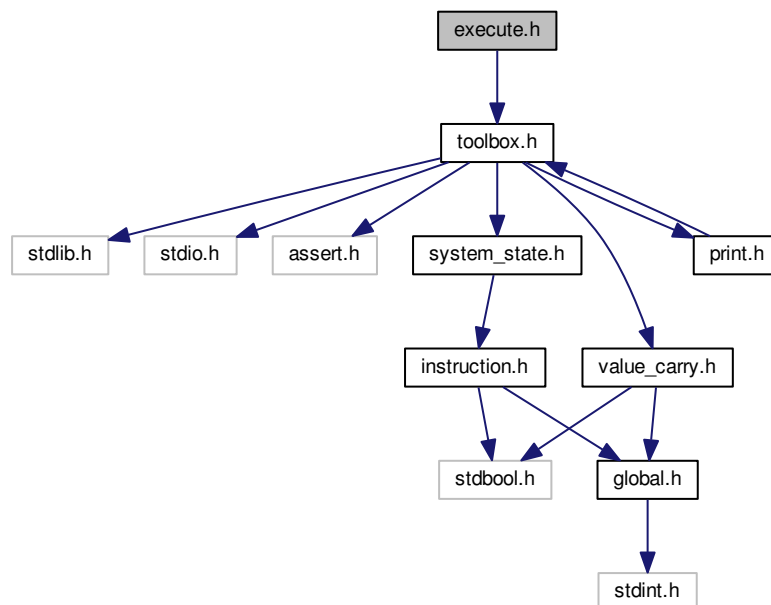
##### Parameters

<i>machine</i>	The current system state.
----------------	---------------------------

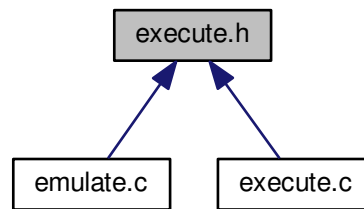
## 4.5 execute.h File Reference

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

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.5.2.2 void execute\_branch ( system\_state\_t \* machine )

Executes a branch instruction.

##### Parameters

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.5.2.3 void execute\_dpi ( system\_state\_t \* machine )

Executes a data processing instruction.

##### Parameters

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.5.2.4 void execute\_mul ( system\_state\_t \* machine )

Executes a multiply instruction.

##### Parameters

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.5.2.5 void execute\_sdt ( system\_state\_t \* machine )

Executes a single data transfer instruction.

##### Parameters

<i>machine</i>	The current system state.
----------------	---------------------------

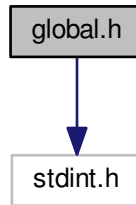
## 4.6 global.h File Reference

Definition of useful constants and type aliases.

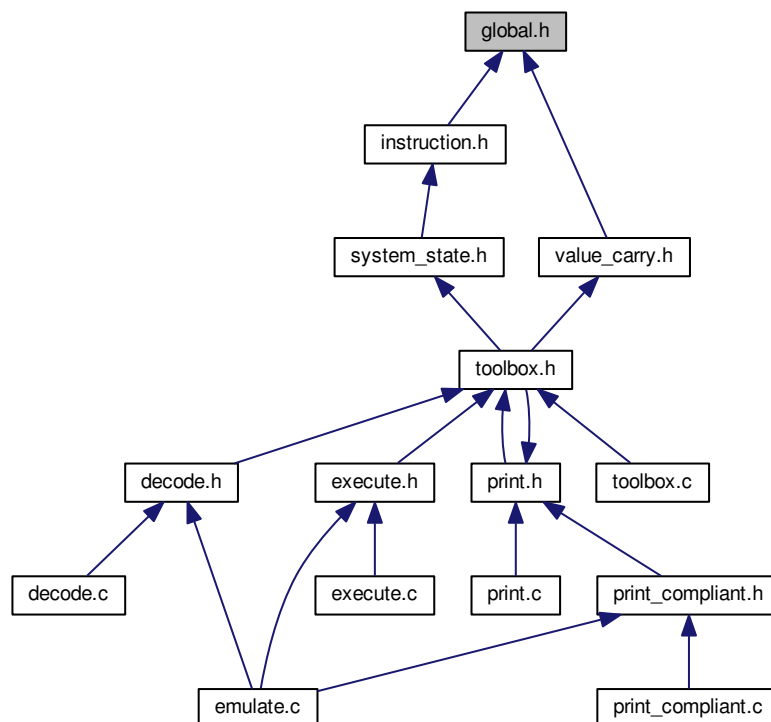


```
#include <stdint.h>
```

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 0xFFFFFFFF`  
*A mask which removes the first 4 bits when used with bitwise and.*
- `#define MASK_FIRST_6 0x3FFFFFFF`  
*A mask which removes the first 6 bits when used with bitwise and.*
- `#define MASK_FIRST_8 0xFFFFFFF`  
*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 output 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 0xFFFFFFF`

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.

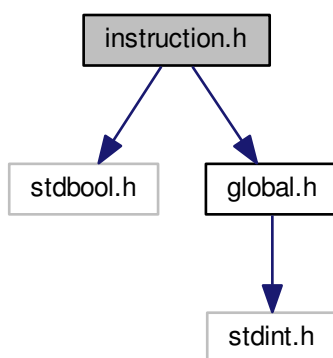
## 4.7 instruction.h File Reference

A header to define the `instruction_t` type.

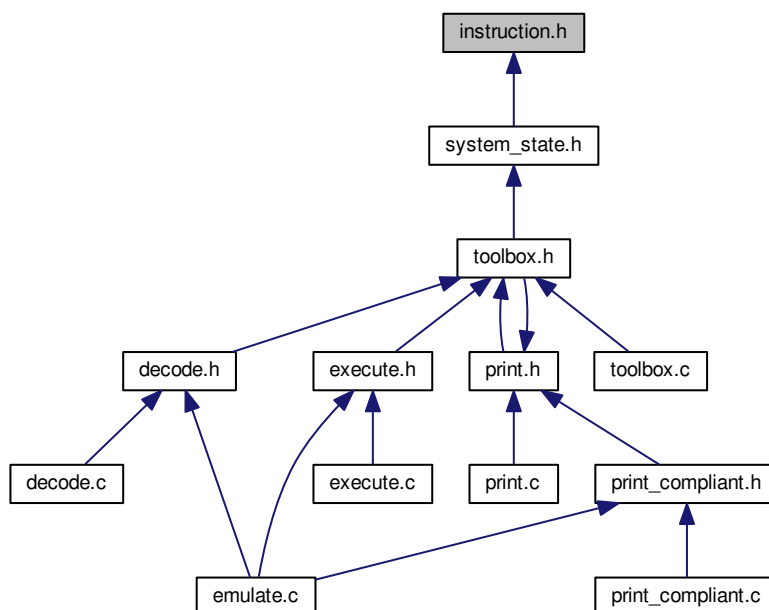
```
#include <stdbool.h>
```

```
#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.*

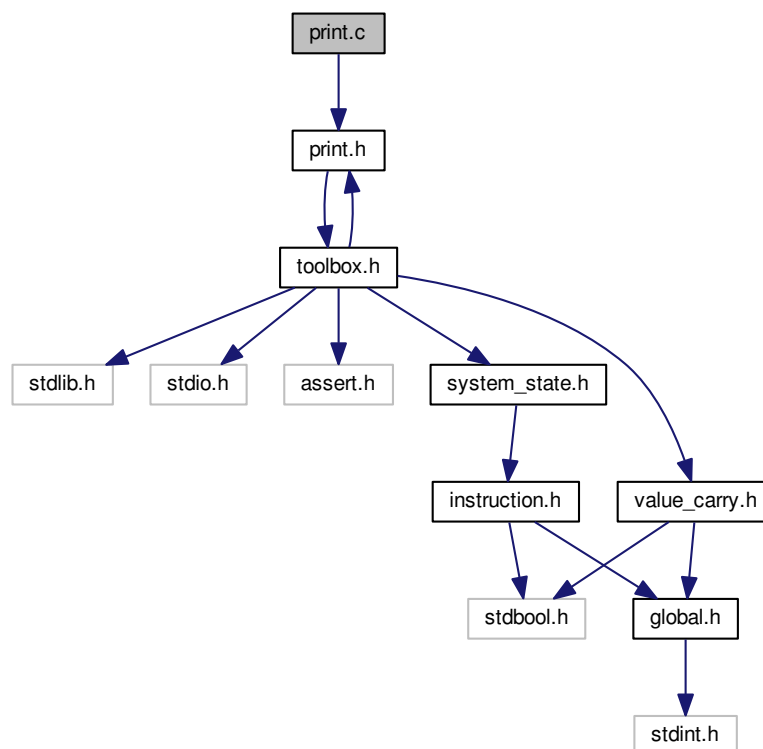
### 4.7.1 Detailed Description

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

<i>cond</i>	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

<i>operation</i>	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

<i>operation</i>	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

<i>memory</i>	An array of bytes to print.
<i>bytes_to_print</i>	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

<i>value</i>	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

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.8.2.7 void print\_fetched\_instruction ( system\_state\_t \* machine )

Prints the fetched instruction, if present.

##### Parameters

<i>machine</i>	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

<i>instruction</i>	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**

<i>machine</i>	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**

<i>machine</i>	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**

<i>machine</i>	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**

<i>value</i>	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**

<i>value</i>	The signed 2's complement word to convert.
--------------	--

**Returns**

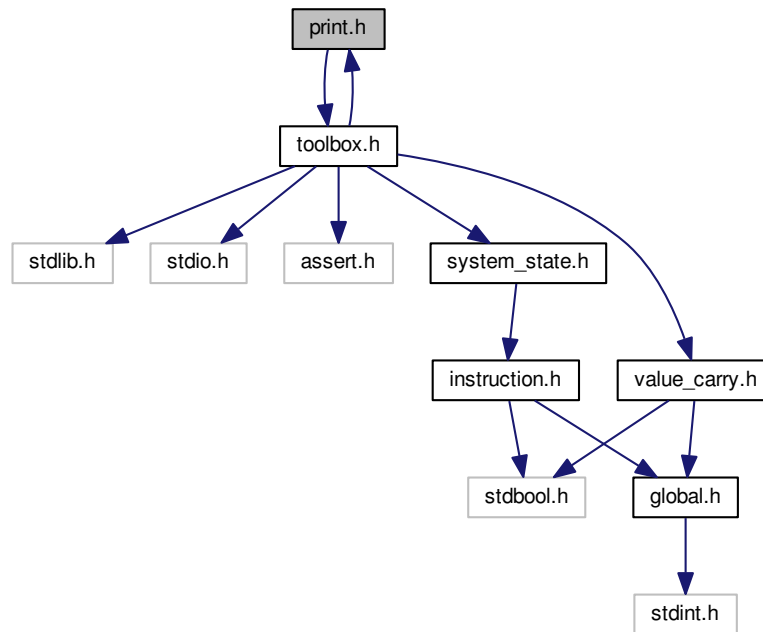
The signed long representation of the word.

## 4.9 print.h File Reference

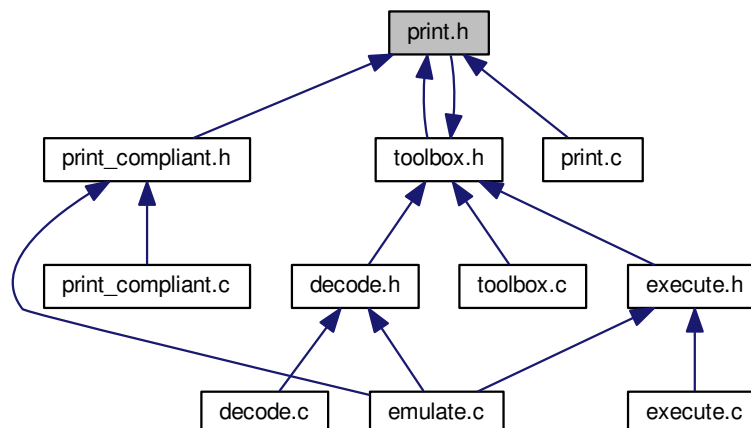
Header file for [print.c](#).

```
#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

<code>cond</code>	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

<i>operation</i>	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

<i>operation</i>	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

<i>memory</i>	An array of bytes to print.
<i>bytes_to_print</i>	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

<i>value</i>	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

<i>machine</i>	The current system state.
----------------	---------------------------

#### 4.9.2.7 void print\_fetched\_instruction ( system\_state\_t \* machine )

Prints the fetched instruction, if present.

##### Parameters

<i>machine</i>	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

<i>instruction</i>	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

<i>machine</i>	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

<i>machine</i>	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

<i>machine</i>	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

<i>value</i>	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

<i>value</i>	The signed 2's complement word to convert.
--------------	--

## Returns

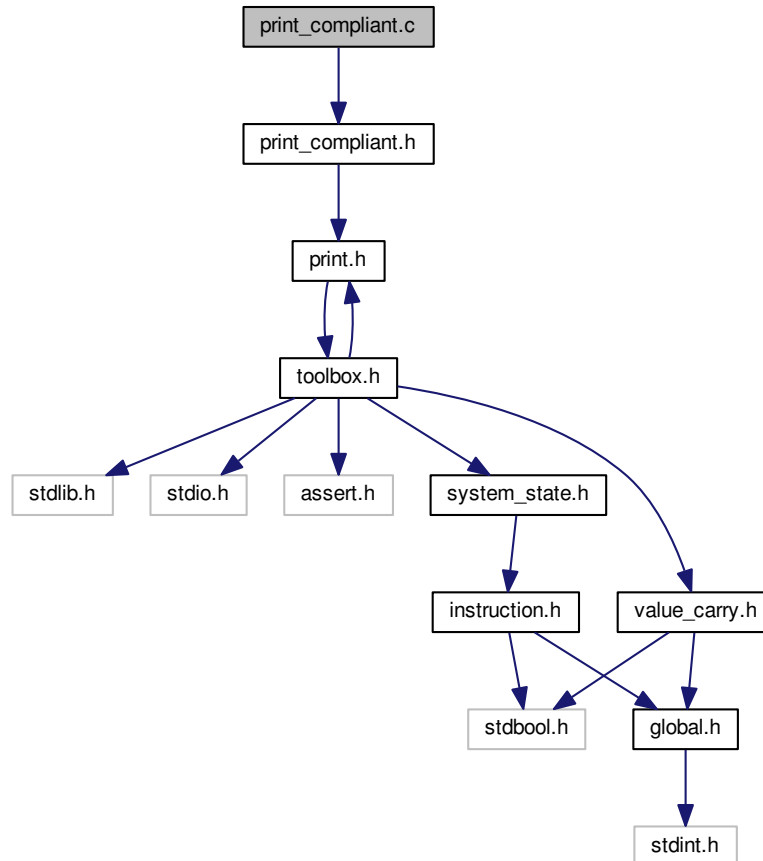
The signed long representation of the word.

## 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

<i>machine</i>	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

<i>machine</i>	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

<i>machine</i>	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

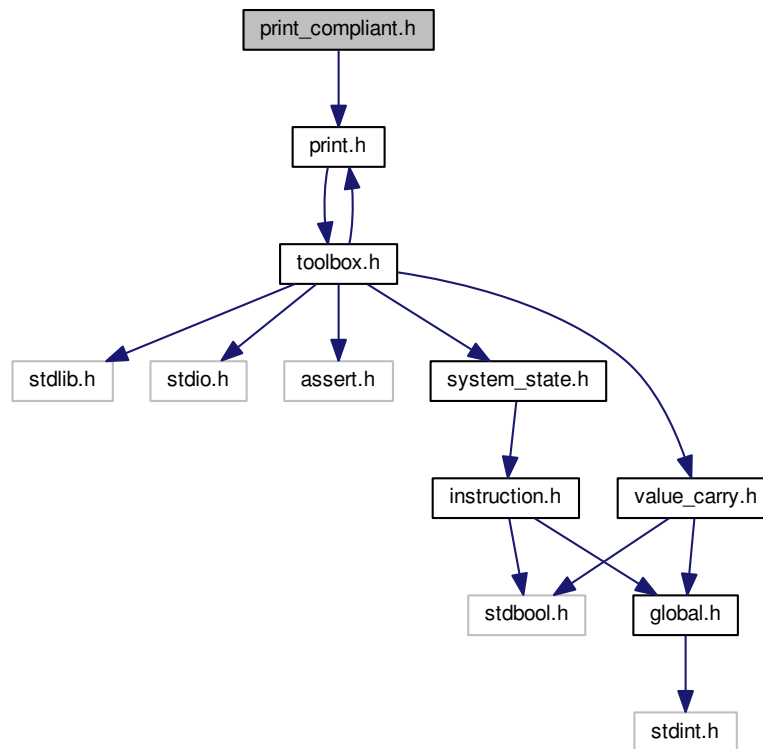
<i>value</i>	The word to print.
--------------	--------------------

## 4.11 print\_compliant.h File Reference

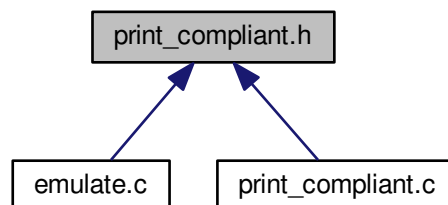
Header file for [print\\_compliant.c](#).

```
#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

<code>machine</code>	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

<code>machine</code>	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

<code>machine</code>	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

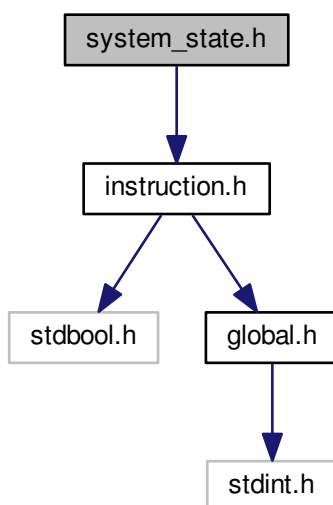
<i>value</i>	The word to print.
--------------	--------------------

## 4.12 system\_state.h File Reference

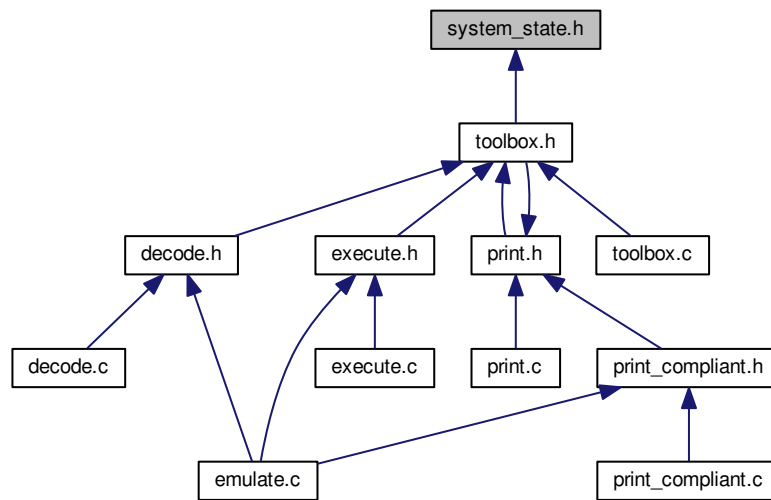
A header to define the [system\\_state\\_t](#) type.

```
#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

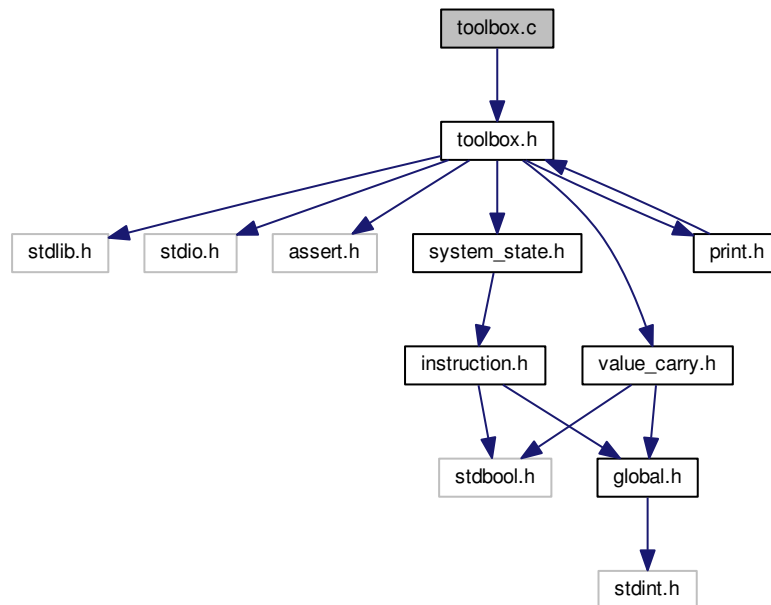
A header to define the [system\\_state\\_t](#) type.

## 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

<i>value</i>	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

<i>machine</i>	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 address is read, prints a message to stdout.
- If another out of bounds address is read, prints an error.

#### Parameters

<i>machine</i>	The current system state.
<i>mem_address</i>	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**

<i>machine</i>	The current system state.
<i>mem_address</i>	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**

<i>value</i>	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**

<i>fname</i>	The filename containing object code to be loaded.
<i>memory</i>	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**

<i>value</i>	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 address is written to, prints a message to stdout.
- If GPIO clear or set address is written to, prints a message to stdout.
- If another out of bounds address is read, prints an error.

##### Parameters

<i>machine</i>	The current system state.
<i>mem_address</i>	The memory address to write to.
<i>word</i>	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

<i>type</i>	The type of shift to use.
<i>shift_amount</i>	The amount to shift by.
<i>value</i>	The value to shift.

##### Returns

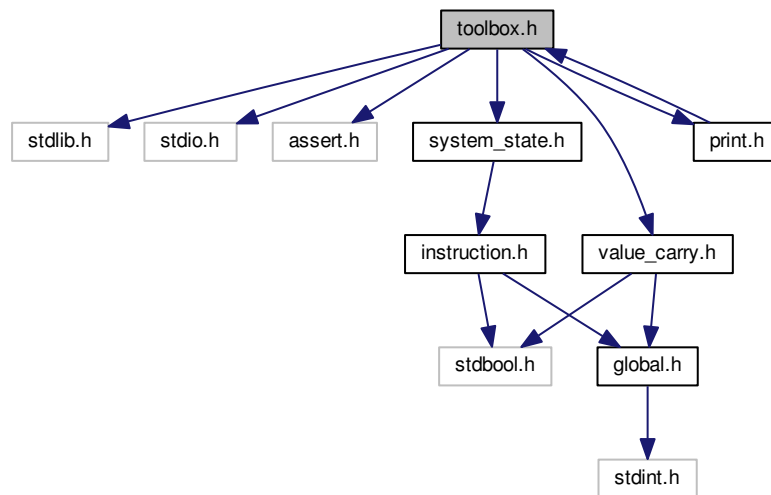
The pointer to the shifted value.

## 4.14 toolbox.h File Reference

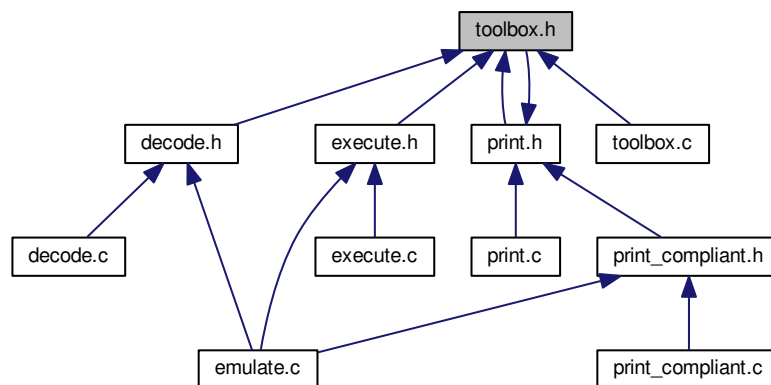
Header file for [toolbox.c](#).

```
#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

<i>value</i>	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

<i>machine</i>	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 address is read, prints a message to stdout.
- If another out of bounds address is read, prints an error.

**Parameters**

<i>machine</i>	The current system state.
<i>mem_address</i>	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**

<i>machine</i>	The current system state.
<i>mem_address</i>	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**

<i>value</i>	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

<i>fname</i>	The filename containing object code to be loaded.
<i>memory</i>	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

<i>value</i>	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 address is written to, prints a message to stdout.
- If GPIO clear or set address is written to, prints a message to stdout.
- If another out of bounds address is read, prints an error.

## Parameters

<i>machine</i>	The current system state.
<i>mem_address</i>	The memory address to write to.
<i>word</i>	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

<i>type</i>	The type of shift to use.
<i>shift_amount</i>	The amount to shift by.
<i>value</i>	The value to shift.

### Returns

The pointer to the shifted value.

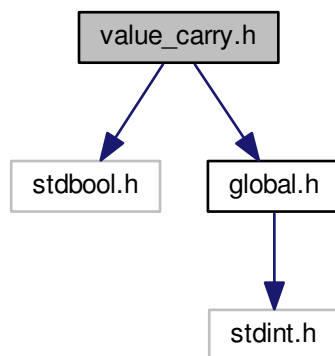
## 4.15 value\_carry.h File Reference

A header to define the `value_carry_t` type.

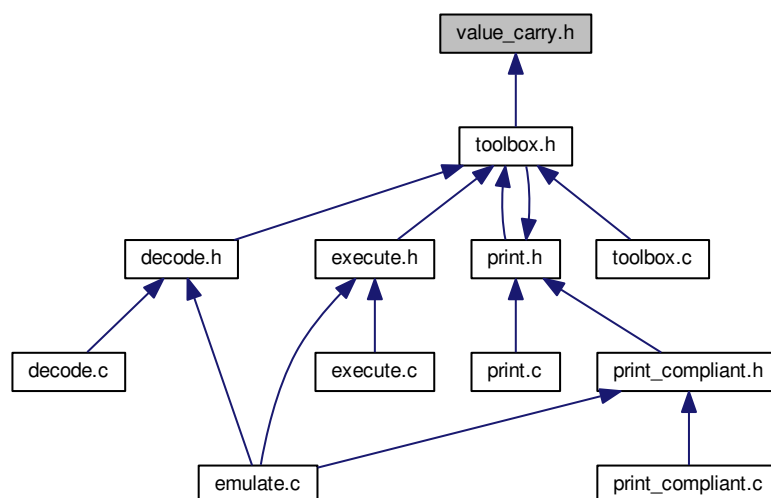
```
#include <stdbool.h>
```

```
#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

ADD  
    global.h, 30

AND  
    global.h, 30

ASR  
    global.h, 30

absolute  
    toolbox.c, 49  
    toolbox.h, 53

address\_t  
    global.h, 28

AL  
    global.h, 29

BRA  
    global.h, 30

branch  
    decode.c, 12  
    decode.h, 16

byte\_t  
    global.h, 28

C  
    global.h, 29

CMP  
    global.h, 30

COMPLIANT\_MODE  
    global.h, 27

CPSR  
    global.h, 27

carry  
    value\_carry\_t, 9

cond  
    instruction\_t, 6

condition  
    execute.c, 21

condition\_t  
    global.h, 29

cpsr\_flags\_t  
    global.h, 29

DPI  
    global.h, 30

data\_processing  
    decode.c, 12  
    decode.h, 16

decode.c, 11  
    branch, 12  
    data\_processing, 12  
    decode\_instruction, 13  
    halt, 13  
    multiply, 13  
    single\_data\_transfer, 13

decode.h, 14  
    branch, 16  
    data\_processing, 16  
    decode\_instruction, 16  
    halt, 17  
    multiply, 17  
    single\_data\_transfer, 17

decode\_instruction  
    decode.c, 13  
    decode.h, 16

decoded\_instruction  
    system\_state\_t, 8

EOR  
    global.h, 30

emulate.c, 18  
    main, 19

EQ  
    global.h, 29

execute  
    execute.c, 21  
    execute.h, 23

execute.c, 20  
    condition, 21  
    execute, 21  
    execute\_branch, 21  
    execute\_dpi, 21  
    execute\_mul, 21  
    execute\_sdt, 22

execute.h, 22  
    execute, 23  
    execute\_branch, 23  
    execute\_dpi, 24  
    execute\_mul, 24  
    execute\_sdt, 24

execute\_branch  
    execute.c, 21  
    execute.h, 23

execute\_dpi  
    execute.c, 21  
    execute.h, 24

execute\_mul  
    execute.c, 21  
    execute.h, 24

execute\_sdt  
    execute.c, 22  
    execute.h, 24

- exit\_program
  - toolbox.c, [49](#)
  - toolbox.h, [53](#)
- fetch\_instruction
  - system\_state\_t, [8](#)
- flag\_0
  - instruction\_t, [6](#)
- flag\_1
  - instruction\_t, [6](#)
- flag\_2
  - instruction\_t, [6](#)
- flag\_3
  - instruction\_t, [6](#)
- GPIO\_ACCESS\_SIZE
  - global.h, [27](#)
- GPIO\_ACCESS\_START
  - global.h, [27](#)
- GPIO\_CLEAR\_SIZE
  - global.h, [27](#)
- GPIO\_CLEAR\_START
  - global.h, [27](#)
- GPIO\_SET\_SIZE
  - global.h, [27](#)
- GPIO\_SET\_START
  - global.h, [28](#)
- GE
  - global.h, [29](#)
- get\_cond
  - print.c, [33](#)
  - print.h, [38](#)
- get\_opcode
  - print.c, [33](#)
  - print.h, [38](#)
- get\_shift
  - print.c, [34](#)
  - print.h, [39](#)
- get\_word
  - toolbox.c, [49](#)
  - toolbox.h, [53](#)
- get\_word\_compliant
  - toolbox.c, [49](#)
  - toolbox.h, [54](#)
- global.h, [24](#)
  - ADD, [30](#)
  - AND, [30](#)
  - ASR, [30](#)
  - address\_t, [28](#)
  - AL, [29](#)
  - BRA, [30](#)
  - byte\_t, [28](#)
  - C, [29](#)
  - CMP, [30](#)
  - COMPLIANT\_MODE, [27](#)
  - CPSR, [27](#)
  - condition\_t, [29](#)
  - cpsr\_flags\_t, [29](#)
  - DPI, [30](#)
  - EOR, [30](#)
  - EQ, [29](#)
  - GPIO\_ACCESS\_SIZE, [27](#)
  - GPIO\_ACCESS\_START, [27](#)
  - GPIO\_CLEAR\_SIZE, [27](#)
  - GPIO\_CLEAR\_START, [27](#)
  - GPIO\_SET\_SIZE, [27](#)
  - GPIO\_SET\_START, [28](#)
  - GE, [29](#)
  - GT, [29](#)
  - instruction\_type\_t, [29](#)
  - LSL, [30](#)
  - LSR, [30](#)
  - LE, [29](#)
  - LT, [29](#)
  - MASK\_FIRST\_4, [28](#)
  - MASK\_FIRST\_6, [28](#)
  - MASK\_FIRST\_8, [28](#)
  - MOV, [30](#)
  - MUL, [30](#)
  - N, [29](#)
  - NUM\_ADDRESSES, [28](#)
  - NUM\_REGISTERS, [28](#)
  - NUL, [30](#)
  - NE, [29](#)
  - ORR, [30](#)
  - opcode\_t, [30](#)
  - PC, [28](#)
  - ROR, [30](#)
  - RSB, [30](#)
  - reg\_address\_t, [29](#)
  - SDB, [30](#)
  - SUB, [30](#)
  - shift\_t, [30](#)
  - TEQ, [30](#)
  - TST, [30](#)
  - V, [29](#)
  - WORD\_SIZE, [28](#)
  - word\_t, [29](#)
  - Z, [29](#)
  - ZER, [30](#)
- GT
  - global.h, [29](#)
- halt
  - decode.c, [13](#)
  - decode.h, [17](#)
- has\_fetched\_instruction
  - system\_state\_t, [8](#)
- immediate\_value
  - instruction\_t, [6](#)
- instruction.h, [31](#)
- instruction\_t, [5](#)
  - cond, [6](#)
  - flag\_0, [6](#)
  - flag\_1, [6](#)
  - flag\_2, [6](#)
  - flag\_3, [6](#)

- immediate\_value, 6
- operation, 6
- rd, 6
- rm, 6
- rn, 6
- rs, 7
- shift\_amount, 7
- shift\_type, 7
- type, 7
- instruction\_type\_t
  - global.h, 29
- is\_negative
  - toolbox.c, 50
  - toolbox.h, 54
- LSL
  - global.h, 30
- LSR
  - global.h, 30
- LE
  - global.h, 29
- load\_file
  - toolbox.c, 50
  - toolbox.h, 54
- LT
  - global.h, 29
- MASK\_FIRST\_4
  - global.h, 28
- MASK\_FIRST\_6
  - global.h, 28
- MASK\_FIRST\_8
  - global.h, 28
- MOV
  - global.h, 30
- MUL
  - global.h, 30
- main
  - emulate.c, 19
- memory
  - system\_state\_t, 8
- multiply
  - decode.c, 13
  - decode.h, 17
- N
  - global.h, 29
- NUM\_ADDRESSES
  - global.h, 28
- NUM\_REGISTERS
  - global.h, 28
- NUL
  - global.h, 30
- NE
  - global.h, 29
- negate
  - toolbox.c, 50
  - toolbox.h, 55
- ORR
  - global.h, 30
- opcode\_t
  - global.h, 30
- operation
  - instruction\_t, 6
- PC
  - global.h, 28
- print.c, 32
  - get\_cond, 33
  - get\_opcode, 33
  - get\_shift, 34
  - print\_array, 34
  - print\_binary\_value, 34
  - print\_decoded\_instruction, 35
  - print\_fetched\_instruction, 35
  - print\_instruction, 35
  - print\_memory, 35
  - print\_registers, 36
  - print\_system\_state, 36
  - print\_value, 36
  - twos\_complement\_to\_long, 36
- print.h, 37
  - get\_cond, 38
  - get\_opcode, 38
  - get\_shift, 39
  - print\_array, 39
  - print\_binary\_value, 39
  - print\_decoded\_instruction, 40
  - print\_fetched\_instruction, 40
  - print\_instruction, 40
  - print\_memory, 40
  - print\_registers, 41
  - print\_system\_state, 41
  - print\_value, 41
  - twos\_complement\_to\_long, 41
- print\_array
  - print.c, 34
  - print.h, 39
- print\_binary\_value
  - print.c, 34
  - print.h, 39
- print\_compliant.c, 42
  - print\_memory\_compliant, 43
  - print\_registers\_compliant, 43
  - print\_system\_state\_compliant, 43
  - print\_value\_compliant, 43
- print\_compliant.h, 43
  - print\_memory\_compliant, 45
  - print\_registers\_compliant, 45
  - print\_system\_state\_compliant, 45
  - print\_value\_compliant, 45
- print\_decoded\_instruction
  - print.c, 35
  - print.h, 40
- print\_fetched\_instruction
  - print.c, 35
  - print.h, 40

- print\_instruction
  - print.c, 35
  - print.h, 40
- print\_memory
  - print.c, 35
  - print.h, 40
- print\_memory\_compliant
  - print\_compliant.c, 43
  - print\_compliant.h, 45
- print\_registers
  - print.c, 36
  - print.h, 41
- print\_registers\_compliant
  - print\_compliant.c, 43
  - print\_compliant.h, 45
- print\_system\_state
  - print.c, 36
  - print.h, 41
- print\_system\_state\_compliant
  - print\_compliant.c, 43
  - print\_compliant.h, 45
- print\_value
  - print.c, 36
  - print.h, 41
- print\_value\_compliant
  - print\_compliant.c, 43
  - print\_compliant.h, 45
- ROR
  - global.h, 30
- RSB
  - global.h, 30
- rd
  - instruction\_t, 6
- reg\_address\_t
  - global.h, 29
- registers
  - system\_state\_t, 8
- rm
  - instruction\_t, 6
- rn
  - instruction\_t, 6
- rs
  - instruction\_t, 7
- SDT
  - global.h, 30
- SUB
  - global.h, 30
- set\_word
  - toolbox.c, 50
  - toolbox.h, 55
- shift\_amount
  - instruction\_t, 7
- shift\_t
  - global.h, 30
- shift\_type
  - instruction\_t, 7
- shifter
  - toolbox.c, 51
  - toolbox.h, 55
- single\_data\_transfer
  - decode.c, 13
  - decode.h, 17
- system\_state.h, 46
- system\_state\_t, 7
  - decoded\_instruction, 8
  - fetch\_instruction, 8
  - has\_fetch\_instruction, 8
  - memory, 8
  - registers, 8
- TEQ
  - global.h, 30
- TST
  - global.h, 30
- toolbox.c, 47
  - absolute, 49
  - exit\_program, 49
  - get\_word, 49
  - get\_word\_compliant, 49
  - is\_negative, 50
  - load\_file, 50
  - negate, 50
  - set\_word, 50
  - shifter, 51
- toolbox.h, 51
  - absolute, 53
  - exit\_program, 53
  - get\_word, 53
  - get\_word\_compliant, 54
  - is\_negative, 54
  - load\_file, 54
  - negate, 55
  - set\_word, 55
  - shifter, 55
- twos\_complement\_to\_long
  - print.c, 36
  - print.h, 41
- type
  - instruction\_t, 7
- V
  - global.h, 29
- value
  - value\_carry\_t, 9
- value\_carry.h, 56
- value\_carry\_t, 9
  - carry, 9
  - value, 9
- WORD\_SIZE
  - global.h, 28
- word\_t
  - global.h, 29
- Z
  - global.h, 29

## ZER

global.h, [30](#)