OneUp Wi-11 Simulator

Generated by Doxygen 1.7.2

Thu Jan 20 2011 21:59:20

Contents

Chapter 1

Main Page

Chapter 2

Class Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 iInterpreter Class Reference

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

- iInterpreter.h

5.2 iMemory Class Reference

Public Member Functions

- virtual void **setAddress** (const iWord &) const =0
- virtual void **setSize** (const int lowerbound, const int upperbound) const =0
- virtual bool **Initialize** () const =0
- virtual Word **Load** (const iWord &) const =0
- virtual bool **Store** (const iWord &address, const iWord &value)=0

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

- iMemory.h

5.3 iRegister Class Reference

Public Member Functions

- virtual Word **getValue** () const =0
- virtual void **Add** (const iWord &)=0
- virtual Register **Add** (const iRegister &) const =0
- virtual void **operator+** (const iWord &)=0

- virtual Register **operator+** (const iRegister &) const =0
- virtual void **Subtract** (const iWord &)=0
- virtual Register **Subtract** (const iRegister &) const =0
- virtual void **operator-** (const iWord &)=0
- virtual Register **operator-** (const iRegister &) const =0
- virtual void **And** (const iWord &)=0
- virtual Register **And** (const iRegister &) const =0
- virtual void **Or** (const iWord &)=0
- virtual Register **Or** (const iRegister &) const =0
- virtual void **Not** ()=0
- virtual Register **Not** () const =0
- virtual void **Store** (const iWord &)=0
- virtual void **Store** (const iRegister &)=0
- virtual Register & **operator=** (const iWord &)=0
- virtual Register & **operator=** (const Register)=0
- virtual Register & **operator++** ()=0
- virtual Register & **operator++** (int)=0

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

- iRegister.h

5.4 iSimulator Class Reference

Public Member Functions

- virtual bool **Initialize** (const char ∗)=0
- virtual bool **Add** (const REGISTER_ID DR, const REGISTER_ID SR1, const REGISTER_-ID SR2)=0
- virtual bool **Add** (const REGISTER_ID DR, const REGISTER_ID SR1, const iWord &immediate)=0
- virtual bool **And** (const REGISTER_ID DR, const REGISTER_ID SR1, const REGISTER_-ID SR2)=0
- virtual bool **And** (const REGISTER_ID DR, const REGISTER_ID SR1, const iWord &immediate)=0
- virtual bool **Branch** (const iWord &address)=0
- virtual bool **Debug** ()=0
- virtual bool **JSR** (const iWord &)=0
- virtual bool **JSRR** (const iWord &baseR, const iWord &address)=0
- virtual bool **Load** (const REGISTER_ID DR, const iWord &address)=0
- virtual bool **LDI** (const REGISTER_ID DR, const iWord &address)=0
- virtual bool **LDR** (const REGISTER_ID DR, const iWord &baseR, const iWord &address)=0
- virtual bool **Not** (const REGISTER_ID DR, const REGISTER_ID SR)=0
- virtual bool **Ret** ()=0

- virtual bool **Store** (const REGISTER_ID DR, const iWord &address)=0
- virtual bool **STI** (const REGISTER_ID DR, const iWord &address)=0
- virtual bool **STR** (const REGISTER_ID DR, const iWord &baseR, const iWord &address)=0
- virtual bool **Trap** (const iWord &address)=0

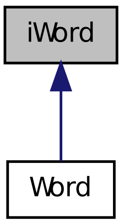
The documentation for this class was generated from the following file:

- iSimulator.h

5.5 iWord Class Reference

The iWord interface class defines the a "word" of data on the Wi-11 Machine.

Inheritance diagram for iWord:



Public Member Functions

- virtual int toInt () const =0

  "To non-negative Integer"

- virtual int toInt2Complement () const =0

  "To Integer as 2's Complement"

- virtual std::string toStr () const =0

  "To String"

- virtual std::string toHex () const =0

  "To Hexadecimal"

- virtual bool fromInt (int)=0

"From Integer"

- virtual bool fromStr (const std::string &)=0

  "From String"

- virtual bool fromHex (const std::string &)=0

  "From Hexadecimal"

- virtual Word Add (const iWord &) const =0

  Adds two words.

- virtual Word operator+ (const iWord &) const =0

  A standard addition operator.

- virtual Word Subtract (const iWord &) const =0

  Subtracts two words.

- virtual Word operator- (const iWord &) const =0

  A standard subtraction operator.

- virtual Word And (const iWord &) const =0

  "And"s the bits of two words.

- virtual Word **Or** (const iWord &) const =0
- virtual Word **Not** () const =0
- virtual void copy (const iWord &)=0

  Copies a word.

- virtual Word & operator= (const Word)=0

  A standard assignment operator.

- virtual iWord & operator++ ()=0
- virtual iWord & operator++ (int)=0

  A standard post-increment operator.

- virtual bool operator[] (int) const =0

  An accessor to the "i"th bit of the value.

- virtual void **print** () const =0

5.5.1 Detailed Description

The iWord interface class defines the a "word" of data on the Wi-11 Machine. The methods present in this inteface are meant to mimic the functionality of the Wi-11 machine, allowing for simplified execution of the instructions therein. As the size of a "word" depends on the architecture, classes implementing this interface should define the word length to be 16 bits in length.

5.5.2 Member Function Documentation

5.5.2.1 virtual int iWord::toInt ( ) const `[pure virtual]`

"To non-negative Integer"

Postcondition

The value of the word is not changed.

Returns

The bits of the word interpreted as a positive integer value.

Implemented in Word.

5.5.2.2 virtual int iWord::toInt2Complement ( ) const `[pure virtual]`

"To Integer as 2's Complement"

Postcondition

The value of the word is not changed.

Returns

The bits of the word interpreted as a signed (2's complement) integer value.

Implemented in Word.

5.5.2.3 virtual std::string iWord::toStr ( ) const `[pure virtual]`

"To String"

Postcondition

The value of the word is not changed.

Returns

"[" + <16 characters: either 1's or 0's> + "]"

Examples:

If the object holds a (2's comp.) value 4: [0000000000000100]
If the object holds a (2's comp.) value -1: [1111111111111111]

Implemented in Word.

5.5.2.4 virtual std::string iWord::toHex ( ) const `[pure virtual]`

"To Hexadecimal"

Postcondition

The value of the word is not changed.

Returns

"0x" + <4 characters in the range [0-9],[A-F]>

Examples:

If the object holds (2's comp.) value 8: 0x0008
If the object holds (2's comp.) value -2: 0xFFFE

Implemented in Word.

5.5.2.5 virtual bool iWord::fromInt (int) `[pure virtual]`

"From Integer"

Parameters

`in`	*value*	The value to be stored into the word.

Postcondition

"value" is not changed.

Returns

True if and only if "value" can be represented in 16 bits

When this function returns "False", the value of the word is unchanged.

Otherwise, the word now holds the value "value".

Implemented in Word.

5.5.2.6 virtual bool iWord::fromStr (const std::string &) `[pure virtual]`

"From String"

Parameters

`in`	*str*	A string of characters meant to represent a "word" to be stored.

Postcondition

"str" is not changed.

Returns

True if and only if "str" is well-formed (as defined in toStr()).

When this function returns "False", the value of the word is unchanged.

Otherwise, the word now holds the value "str".

Implemented in Word.

5.5.2.7 virtual bool iWord::fromHex (const std::string &) `[pure virtual]`

"From Hexadecimal"

Parameters

`in`	*str*	A string of characters meant to represent a "word" to be stored.

Postcondition

"str" is not changed.

Returns

True if and only if "str" is well-formed (as defined in toHex()).

When this function returns "False", the value of the word is unchanged.

Otherwise, the word now holds the value "str".

Implemented in Word.

5.5.2.8 virtual Word iWord::Add (const iWord &) const `[pure virtual]`

Adds two words.

Parameters

`in`	*w*	A word value to be added.

Postcondition

Both "w" and the calling object do not change.

Returns

A new "Word" object containing result of adding "w" and the calling object.

Note

The addition is carried out with no regard to logical overflow.

Implemented in Word.

5.5.2.9 virtual Word iWord::operator+ (const iWord &) const `[pure virtual]`

A standard addition operator.

Note

"result = p + w" is equivalent to "result = p.Add(w)".

Implemented in Word.

5.5.2.10 virtual Word iWord::Subtract (const iWord &) const `[pure virtual]`

Subtracts two words.

Parameters

`in`	*w*	A word value to be subtracted.

Postcondition

Both "w" and the calling object do not change.

Returns

A new "Word" object containing the result of subtracting "w" from the calling object.

Note

The subtraction is carried out with no regard for logical overflow.

Implemented in Word.

5.5.2.11 virtual Word iWord::operator- (const iWord &) const `[pure virtual]`

A standard subtraction operator.

Note

"result = p - w" is equivalent to "result = p.Subtract(w)".

Implemented in Word.

5.5.2.12 virtual Word iWord::And (const iWord &) const `[pure virtual]`

"And"s the bits of two words.

Parameters

`in`	*w*	A word value to be "and"ed.

Postcondition

Both "w" and the calling object do not change.

Returns

A new "Word" object containing the result of performing a bit-wise and on "w" and the calling object.

Implemented in Word.

5.5.2.13 virtual void iWord::copy (const iWord &) `[pure virtual]`

Copies a word.

Parameters

`out`	*The*	value to be copied.

Postcondition

The caller equals that parameter.

Equivalent to the assignment "caller = parameter".

Implemented in Word.

5.5.2.14 virtual Word& iWord::operator= (const *Word*) `[pure virtual]`

A standard assignment operator.

Parameters

`in`	*The*	value to be copied.

Returns

A copy of the parameter.

The return value and parameter here must be declared as "Word"s as C++ does not work well with polymorphic assignment operators.

Implemented in Word.

5.5.2.15 virtual iWord& iWord::operator++ ( ) `[pure virtual]`

A standard pre-increment operator.

Returns

A reference to itself.

The object increments its value BEFORE the execution of the current line.

Implemented in Word.

5.5.2.16 virtual iWord& iWord::operator++ (int) `[pure virtual]`

A standard post-increment operator.

Returns

A reference to itself.

The object increments its value AFTER the execution of the current line.

Implemented in Word.

5.5.2.17 virtual bool iWord::operator[] (int) const `[pure virtual]`

An accessor to the "i"th bit of the value.

Parameters

`in`	*The*	index of the bit in question.

Precondition

The index must be less than the size of a word, ie. 16.

Returns

True <=> 1, False <=> 0.

The number of the bits starts at zero and rises into the more significant bits. Examples: If the object "num" holds a value of 4 (0...100 in binary), num[0] = 0, num[1] = 0, num[2] = 1. If it holds a value of 1 (0...001 in binary) num[0] = 1, num[1] = 0, num[2] = 0, etc. If it holds a negative value (Starting with a 1 in 2's complement), num[15] = 1.

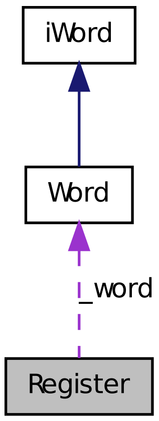
Implemented in Word.

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

- iWord.h

5.6 Register Class Reference

Collaboration diagram for Register:



Public Member Functions

- **Register** (const Word w)
- Word **getValue** () const
- void **Add** (const iWord &)
- Register **Add** (const iRegister &) const
- void **operator+** (const iWord &)
- Register **operator+** (const iRegister &) const
- void **Subtract** (const iWord &)
- Register **Subtract** (const iRegister &) const
- void **operator-** (const iWord &)
- Register **operator-** (const iRegister &) const
- void **And** (const iWord &)
- Register **And** (const iRegister &) const
- void **Or** (const iWord &)
- Register **Or** (const iRegister &) const
- void **Not** ()
- Register **Not** () const
- void **Store** (const iWord &)
- void **Store** (const iRegister &)
- Register & **operator=** (const iWord &)
- Register & **operator=** (const Register)

- Register & **operator++** ()
- Register & **operator++** (int)

Private Attributes

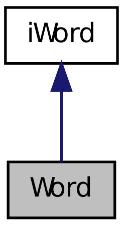
- Word **_word**

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

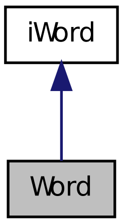
- Register.h
- Register.cpp

5.7 Word Class Reference

Inheritance diagram for Word:



Collaboration diagram for Word:



Public Member Functions

- int toInt () const

  "To non-negative Integer"

- int toInt2Complement () const

  "To Integer as 2's Complement"

- std::string toStr () const

  "To String"

- std::string toHex () const

  "To Hexadecimal"

- bool fromInt (int)

  "From Integer"

- bool fromStr (const std::string &)

  "From String"

- bool fromHex (const std::string &)

  "From Hexadecimal"

- Word Add (const iWord &) const

  Adds two words.

- Word operator+ (const iWord &) const

  A standard addition operator.

- Word Subtract (const iWord &) const

  Subtracts two words.

- Word operator- (const iWord &) const

  A standard subtraction operator.

- Word And (const iWord &) const

  "And"s the bits of two words.

- Word **Or** (const iWord &) const
- Word **Not** () const
- void copy (const iWord &)

  Copies a word.

- Word & operator= (const Word)

  A standard assignment operator.

- iWord & operator++ ()
- iWord & operator++ (int)

  A standard post-increment operator.
- bool operator[] (const int) const

  An accessor to the "i"th bit of the value.
- void **print** () const

Private Member Functions

- bool **_hasBit** (int) const

Private Attributes

- unsigned short **_value**

5.7.1 Member Function Documentation

5.7.1.1 int Word::toInt ( ) const `[virtual]`

"To non-negative Integer"

Postcondition

The value of the word is not changed.

Returns

The bits of the word interpreted as a positive integer value.

Implements iWord.

5.7.1.2 int Word::toInt2Complement ( ) const `[virtual]`

"To Integer as 2's Complement"

Postcondition

The value of the word is not changed.

Returns

The bits of the word interpreted as a signed (2's complement) integer value.

Implements iWord.

5.7.1.3 string Word::toStr ( ) const `[virtual]`

"To String"

Postcondition

The value of the word is not changed.

Returns

"[" + <16 characters: either 1's or 0's> + "]"

Examples:

If the object holds a (2's comp.) value 4: [0000000000000100]
If the object holds a (2's comp.) value -1: [1111111111111111]

Implements iWord.

5.7.1.4 string Word::toHex ( ) const `[virtual]`

"To Hexadecimal"

Postcondition

The value of the word is not changed.

Returns

"0x" + <4 characters in the range [0-9],[A-F]>

Examples:

If the object holds (2's comp.) value 8: 0x0008
If the object holds (2's comp.) value -2: 0xFFFE

Implements iWord.

5.7.1.5 bool Word::fromInt (int) `[virtual]`

"From Integer"

Parameters

`in`	*value*	The value to be stored into the word.

Postcondition

"value" is not changed.

Returns

True if and only if "value" can be represented in 16 bits

When this function returns "False", the value of the word is unchanged.

Otherwise, the word now holds the value "value".

Implements iWord.

5.7.1.6 bool Word::fromStr (const std::string &) `[virtual]`

"From String"

Parameters

`in`	*str*	A string of characters meant to represent a "word" to be stored.

Postcondition

"str" is not changed.

Returns

True if and only if "str" is well-formed (as defined in toStr()).

When this function returns "False", the value of the word is unchanged.

Otherwise, the word now holds the value "str".

Implements iWord.

5.7.1.7 bool Word::fromHex (const std::string &) `[virtual]`

"From Hexadecimal"

Parameters

`in`	*str*	A string of characters meant to represent a "word" to be stored.

Postcondition

"str" is not changed.

Returns

True if and only if "str" is well-formed (as defined in toHex()).

When this function returns "False", the value of the word is unchanged.

Otherwise, the word now holds the value "str".

Implements iWord.

5.7.1.8 Word Word::Add (const iWord &) const [virtual]

Adds two words.

Parameters

`in`	*w*	A word value to be added.

Postcondition

Both "w" and the calling object do not change.

Returns

A new "Word" object containing result of adding "w" and the calling object.

Note

The addition is carried out with no regard to logical overflow.

Implements iWord.

5.7.1.9 Word Word::operator+ (const iWord &) const [virtual]

A standard addition operator.

Note

"result = p + w" is equivalent to "result = p.Add(w)".

Implements iWord.

5.7.1.10 Word Word::Subtract (const iWord &) const [virtual]

Subtracts two words.

Parameters

`in`	*w*	A word value to be subtracted.

Postcondition

Both "w" and the calling object do not change.

Returns

A new "Word" object containing the result of subtracting "w" from the calling object.

Note

The subtraction is carried out with no regard for logical overflow.

Implements iWord.

5.7.1.11 Word Word::operator- (const iWord &) const [virtual]

A standard subtraction operator.

Note

"result = p - w" is equivalent to "result = p.Subtract(w)".

Implements iWord.

5.7.1.12 Word Word::And (const iWord &) const [virtual]

"And"s the bits of two words.

Parameters

in	*w*	A word value to be "and"ed.

Postcondition

Both "w" and the calling object do not change.

Returns

A new "Word" object containing the result of performing a bit-wise and on "w" and the calling object.

Implements iWord.

5.7.1.13 void Word::copy (const iWord &) [virtual]

Copies a word.

Parameters

out	*The*	value to be copied.

Postcondition

The caller equals that parameter.

Equivalent to the assignment "caller = parameter".

Implements iWord.

5.7.1.14 Word & Word::operator= (const *Word*) [virtual]

A standard assignment operator.

Parameters

in	*The*	value to be copied.

Returns

A copy of the parameter.

The return value and parameter here must be declared as "Word"s as C++ does not work well with polymorphic assignment operators.

Implements iWord.

5.7.1.15 iWord & Word::operator++ ( ) `[virtual]`

A standard pre-increment operator.

Returns

A reference to itself.

The object increments its value BEFORE the execution of the current line.

Implements iWord.

5.7.1.16 iWord & Word::operator++ (int) `[virtual]`

A standard post-increment operator.

Returns

A reference to itself.

The object increments its value AFTER the execution of the current line.

Implements iWord.

5.7.1.17 bool Word::operator[] (const) const `[virtual]`

An accessor to the "i"th bit of the value.

Parameters

in	*The*	index of the bit in question.

Precondition

The index must be less than the size of a word, ie. 16.

Returns

True <=> 1, False <=> 0.

The number of the bits starts at zero and rises into the more significant bits. Examples: If the object "num" holds a value of 4 (0...100 in binary), num[0] = 0, num[1] = 0, num[2] = 1. If it holds a value of 1 (0...001 in binary) num[0] = 1, num[1] = 0, num[2] = 0, etc. If it holds a negative value (Starting with a 1 in 2's complement), num[15] = 1.

Implements iWord.

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

- Word.h
- Word.cpp

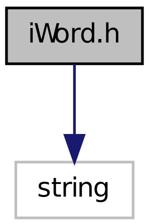
Chapter 6

File Documentation

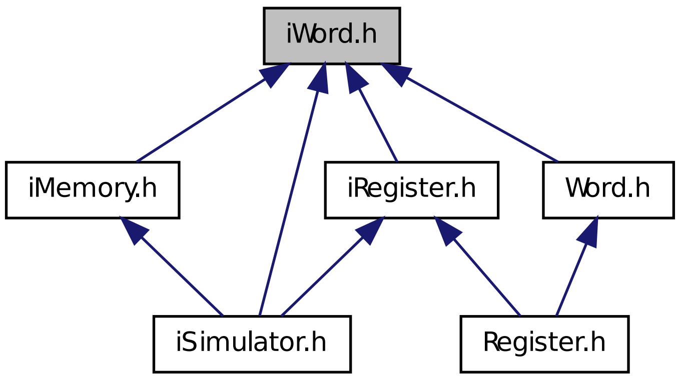
6.1 iWord.h File Reference

The interface implemented by the "Word" class.

Include dependency graph for iWord.h:



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



Classes

- class iWord

    The iWord interface class defines the a "word" of data on the Wi-11 Machine.

6.1.1 Detailed Description

The interface implemented by the "Word" class.

Author

Joshua Green
Andrew Groot

Defines the operations and signatures by which the "Word" class should operate. The signatures, while intended to be coded to the interface, are done as to this as C++ allows.

Index