

OneUp Wi-11 Simulator

Generated by Doxygen 1.7.2

Fri Jan 21 2011 20:29:39

Contents

1	Main Page	1
1.1	Introduction	1
1.2	Object Files	1
1.2.1	The Header Record	1
1.2.2	Text Records	2
1.2.3	The End Record	2
1.3	Interactions	2
2	Directory Hierarchy	3
2.1	Directories	3
3	Class Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Directory Documentation	11
6.1	code/ Directory Reference	11
6.2	code/MemoryTest/ Directory Reference	12
6.3	code/test/ Directory Reference	13
7	Class Documentation	15
7.1	iDecoder Class Reference	15
7.2	iInterpreter Class Reference	15
7.3	iLoader Class Reference	15
7.4	iMemory Class Reference	16
7.5	Instruction Struct Reference	16
7.6	iObjParser Class Reference	16
7.7	iRegister Class Reference	17
7.7.1	Detailed Description	18
7.7.2	Member Function Documentation	18
7.7.2.1	GetValue	18
7.7.2.2	Add	19
7.7.2.3	Add	19
7.7.2.4	operator+	19
7.7.2.5	Subtract	20

7.7.2.6	Subtract	20
7.7.2.7	operator-	20
7.7.2.8	And	20
7.7.2.9	And	21
7.7.2.10	Or	21
7.7.2.11	Or	21
7.7.2.12	Not	22
7.7.2.13	Not	22
7.7.2.14	Store	22
7.7.2.15	Store	23
7.7.2.16	operator=	23
7.7.2.17	operator=	23
7.7.2.18	operator++	23
7.7.2.19	operator++	24
7.8	iSimulator Class Reference	24
7.9	iWi11 Class Reference	24
7.10	iWord Class Reference	25
7.10.1	Detailed Description	28
7.10.2	Member Function Documentation	29
7.10.2.1	ToInt	29
7.10.2.2	ToInt2Complement	29
7.10.2.3	ToStr	29
7.10.2.4	ToHex	30
7.10.2.5	FromInt	30
7.10.2.6	FromStr	30
7.10.2.7	FromHex	31
7.10.2.8	Add	31
7.10.2.9	operator+	32
7.10.2.10	Subtract	32
7.10.2.11	operator-	32
7.10.2.12	And	32
7.10.2.13	Or	33
7.10.2.14	Not	33
7.10.2.15	Copy	33
7.10.2.16	operator=	34
7.10.2.17	operator++	34
7.10.2.18	operator++	34
7.10.2.19	operator[]	35
7.10.2.20	toInt	35
7.10.2.21	toInt2Complement	35
7.10.2.22	toStr	36
7.10.2.23	toHex	36
7.10.2.24	fromInt	36
7.10.2.25	fromStr	37
7.10.2.26	fromHex	37
7.10.2.27	Add	38
7.10.2.28	operator+	38
7.10.2.29	Subtract	38
7.10.2.30	operator-	39
7.10.2.31	And	39

7.10.2.32	Or	39
7.10.2.33	Not	39
7.10.2.34	copy	40
7.10.2.35	operator=	40
7.10.2.36	operator++	40
7.10.2.37	operator++	41
7.10.2.38	operator[]	41
7.11	Memory Class Reference	42
7.12	ObjectData Struct Reference	43
7.13	Register Class Reference	43
7.13.1	Member Function Documentation	46
7.13.1.1	GetValue	46
7.13.1.2	Add	46
7.13.1.3	Add	46
7.13.1.4	operator+	47
7.13.1.5	Subtract	47
7.13.1.6	Subtract	47
7.13.1.7	operator-	47
7.13.1.8	And	48
7.13.1.9	And	48
7.13.1.10	Or	48
7.13.1.11	Or	49
7.13.1.12	Not	49
7.13.1.13	Not	49
7.13.1.14	Store	49
7.13.1.15	Store	50
7.13.1.16	operator=	50
7.13.1.17	operator=	50
7.13.1.18	operator++	50
7.13.1.19	operator++	51
7.14	ResultDecoder Class Reference	51
7.15	Word Class Reference	51
7.15.1	Member Function Documentation	55
7.15.1.1	_hasBit	55
7.15.1.2	toInt	55
7.15.1.3	toInt2Complement	55
7.15.1.4	toStr	56
7.15.1.5	toHex	56
7.15.1.6	fromInt	56
7.15.1.7	fromStr	57
7.15.1.8	fromHex	57
7.15.1.9	Add	58
7.15.1.10	operator+	58
7.15.1.11	Subtract	58
7.15.1.12	operator-	59
7.15.1.13	And	59
7.15.1.14	Or	59
7.15.1.15	Not	60
7.15.1.16	copy	60
7.15.1.17	operator=	60

7.15.1.18	operator++	61
7.15.1.19	operator++	61
7.15.1.20	operator[]	61
7.15.1.21	_HasBit	62
7.15.1.22	ToInt	62
7.15.1.23	ToInt2Complement	62
7.15.1.24	ToStr	62
7.15.1.25	ToHex	63
7.15.1.26	FromInt	63
7.15.1.27	FromStr	64
7.15.1.28	FromHex	64
7.15.1.29	Add	64
7.15.1.30	operator+	65
7.15.1.31	Subtract	65
7.15.1.32	operator-	65
7.15.1.33	And	66
7.15.1.34	Or	66
7.15.1.35	Not	66
7.15.1.36	Copy	67
7.15.1.37	operator=	67
7.15.1.38	operator++	67
7.15.1.39	operator++	67
7.15.1.40	operator[]	68
7.15.2	Member Data Documentation	68
7.15.2.1	_value	68
8	File Documentation	69
8.1	iRegister.h File Reference	69
8.1.1	Detailed Description	70
8.2	Register.h File Reference	70
8.2.1	Detailed Description	71

Chapter 1

Main Page

1.1 Introduction

The "Wi-11 Machine" is a simple, 16-bit computer architecture. It has 8 general purpose registers, 3 condition code registers (CCRs), and a program counter (PC). This software package is meant to emulate its execution, as well as present the user with information regarding the state of the machine after each instruction is executed. However, before one can delve into the behind-the-scenes details, one must understand the environment. In particular, an understanding the object file syntax and the interactions between the components used in this project is necessary.

1.2 Object Files

The object files (usually file_name.o) that this simulator accepts are ascii text files with the following structure:

- One [Header Record](#)
- Several [Text Records](#)
- One [End Record](#)

1.2.1 The Header Record

The Header Record is a single line that prepares the system for the storing the instructions to come.

Components

- A capital 'H'. This designates that it is the Header Record.
- A 6 character "segment name" (anything will do).

- A 4-digit Hexadecimal value that corresponds to the "load address" of the program. Instructions can be written starting at this address.
- A second 4-digit Hexadecimal value that denotes the length of the program-load segment (the size of memory into which the instructions will be loaded).

At a glance: There is an 'H', a segment name, the first location where instructions can be written, and the number of memory locations for instructions.

1.2.2 Text Records

Following the Header Record are several Text Records. Each Text Record corresponds to a single machine instruction and, like the header record, is on a single line.

Components

- A capital 'T'. This designates that it is a Text Record.
- A 4-digit hexadecimal value -- The location in memory at which the instruction will be stored.
- A second 4-digit Hexadecimal value -- The encoding of the instruction to be stored.

At a glance: There is a T, the location to store the instruction, and the instruction itself.

1.2.3 The End Record

The End Record is, as the name would suggest, the last line of the line. Its purpose is to denote the end of instructions to be written and to give an initial value for the PC.

Components

- The End Record begins with a capital 'E'.
- Next, and last, a 4-digit hexadecimal value to be put into the PC.

At a glance: There is an E, and the location in memory from which the first instruction should be fetched.

1.3 Interactions

Chapter 2

Directory Hierarchy

2.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

code	11
MemoryTest	12
test	13

Chapter 3

Class Index

3.1 Class Hierarchy

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

iDecoder	15
iInterpreter	15
iLoader	15
iMemory	16
Memory	42
Memory	42
Instruction	16
iObjParser	16
iRegister	17
Register	43
iSimulator	24
iWi11	24
iWord	25
Word	51
Word	51
ObjectData	43
ResultDecoder	51

Chapter 4

Class Index

4.1 Class List

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

iDecoder	15
iInterpreter	15
iLoader	15
iMemory	16
Instruction	16
iObjParser	16
iRegister (Defines a "register" in the Wi-11 machine)	17
iSimulator	24
iWi11	24
iWord (Defines a "word" of data on the Wi-11 Machine)	25
Memory	42
ObjectData	43
Register	43
ResultDecoder	51
Word	51

Chapter 5

File Index

5.1 File List

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

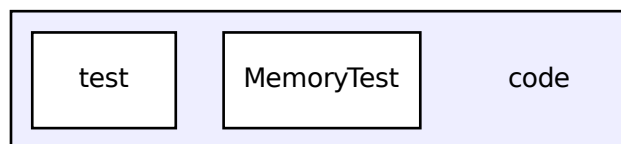
iDecoder.h	??
iInterpreter.h	??
iLoader.h	??
iMemory.h	??
MemoryTest/iMemory.h	??
iObjParser.h	??
iRegister.h (Definition of a "register" in the Wi-11 machine)	69
iSimulator.h	??
iWi11.h	??
iWord.h	??
MemoryTest/iWord.h	??
Memory.h	??
MemoryTest/Memory.h	??
Register.h (Definition of private data for the "Register" class)	70
MemoryTest/ResultCodes.h	??
ResultCodes.h	??
MemoryTest/Word.h	??
Word.h	??

Chapter 6

Directory Documentation

6.1 code/ Directory Reference

Directory dependency graph for code/:



Directories

- directory [MemoryTest](#)
- directory [test](#)

Files

- file `iDecoder.h`
- file `iInterpreter.h`
- file `iLoader.h`
- file `iMemory.h`
- file `iObjParser.h`

- file [iRegister.h](#)

Definition of a "register" in the Wi-11 machine.

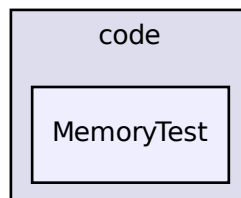
- file [iSimulator.h](#)
- file [iWi11.h](#)
- file [iWord.h](#)
- file [Main.cpp](#)
- file [Memory.cpp](#)
- file [Memory.h](#)
- file [Register.cpp](#)
- file [Register.h](#)

Definition of private data for the "Register" class.

- file [ResultCodes.h](#)
- file [Word.cpp](#)
- file [Word.h](#)

6.2 code/MemoryTest/ Directory Reference

Directory dependency graph for code/MemoryTest/:

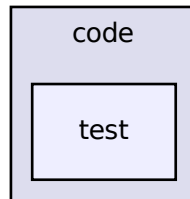


Files

- file [MemoryTest/iMemory.h](#)
- file [MemoryTest/iWord.h](#)
- file [MemoryTest/Memory.cpp](#)
- file [MemoryTest/Memory.h](#)
- file [MemoryTest.cpp](#)
- file [MemoryTest/ResultCodes.h](#)
- file [MemoryTest/Word.cpp](#)
- file [MemoryTest/Word.h](#)

6.3 code/test/ Directory Reference

Directory dependency graph for code/test/:



Files

- file **RegisterTest.cpp**
- file **WordTest.cpp**

Chapter 7

Class Documentation

7.1 iDecoder Class Reference

Public Member Functions

- virtual [Instruction](#) DecodeInstruction (const [iWord](#) &) const =0

7.2 iInterpreter Class Reference

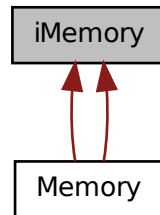
7.3 iLoader Class Reference

Public Member Functions

- virtual [iLoader](#) ([iMemory](#) *) =0
- virtual Codes::RESULT Load (const char *filename, [iWord](#) &PC_address) =0

7.4 iMemory Class Reference

Inheritance diagram for iMemory:



Public Member Functions

- virtual `Codes::RESULT Reserve` (const `iWord` &initial_address, const `iWord` &length)=0
- virtual `Word Load` (const `iWord` &) const =0
- virtual `Codes::RESULT Store` (const `iWord` &address, const `Word` &value)=0
- virtual `Codes::RESULT Reserve` (const `iWord` &initial_address, const `iWord` &length)=0
- virtual `Word Load` (const `iWord` &) const =0
- virtual `Codes::RESULT Store` (const `iWord` &address, const `Word` &value)=0

7.5 Instruction Struct Reference

Public Attributes

- `INSTRUCTION_TYPE` type
- `std::vector< Word > data`

7.6 iObjParser Class Reference

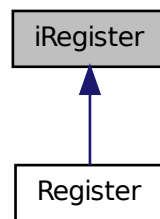
Public Member Functions

- virtual `Codes::Result Initialize` (const char *)=0
- virtual `ObjectData GetNext` ()=0

7.7 iRegister Class Reference

Defines a "register" in the Wi-11 machine.

Inheritance diagram for iRegister:



Public Member Functions

- virtual [Word GetValue](#) () const =0
Retrieves a copy of the word of data store in the register.
- virtual void [Add](#) (const [iWord](#) &w)=0
Adds a word of data to the calling object.
- virtual [Register Add](#) (const [iRegister](#) &r) const =0
Adds a word of data to the calling object.
- virtual [Register operator+](#) (const [iRegister](#) &r) const =0
A standard add operator.
- virtual void [Subtract](#) (const [iWord](#) &w)=0
Subtracts a word of data from the calling object.
- virtual [Register Subtract](#) (const [iRegister](#) &r) const =0
Subtracts a word of data from the calling object.
- virtual [Register operator-](#) (const [iRegister](#) &r) const =0
A standard subtraction operator.
- virtual void [And](#) (const [iWord](#) &w)=0
Performs a bit-wise and.

- virtual [Register And](#) (const [iRegister](#) &r) const =0
Performs a bit-wise and.
- virtual void [Or](#) (const [iWord](#) &w)=0
Performs a bit-wise "or".
- virtual [Register Or](#) (const [iRegister](#) &r) const =0
Performs a bit-wise or.
- virtual void [Not](#) ()=0
Performs a bit-wise not.
- virtual [Register Not](#) () const =0
Performs a bit-wise not.
- virtual void [Store](#) (const [iWord](#) &w)=0
Stores a word of data.
- virtual void [Store](#) (const [iRegister](#) &r)=0
Stores a copy of another register.
- virtual [Register & operator=](#) (const [iWord](#) &w)=0
A standard assignment operator.
- virtual [Register & operator=](#) (const [Register](#) r)=0
A standard assignment operator.
- virtual [Register & operator++](#) ()=0
A standard pre-increment operator.
- virtual [Register & operator++](#) (int)=0
A standard post-increment operator.

7.7.1 Detailed Description

Defines a "register" in the Wi-11 machine. The methods present in this interface are meant to mimic the functionality of the Wi-11 machine, allowing for simplified execution of the instructions therein. This interface class will serve as a base from which the general purpose registers and program counter of the Wi-11 can be defined.

7.7.2 Member Function Documentation

7.7.2.1 virtual Word [iRegister::GetValue](#) () const [pure virtual]

Retrieves a copy of the word of data store in the register.

Postcondition

The value of the calling object is not changed.

Returns

A new [Word](#) object holding the value that is stored in the register.

Implemented in [Register](#).

7.7.2.2 virtual void iRegister::Add (const iWord & w) [pure virtual]

Adds a word of data to the calling object.

Parameters

in	w	The value to be added.
----	---	------------------------

Postcondition

The calling object equals its previous value plus the value of "w"; "w", however, will remain unchanged.

Implemented in [Register](#).

7.7.2.3 virtual Register iRegister::Add (const iRegister & r) const [pure virtual]

Adds a word of data to the calling object.

Parameters

in	r	The value to be added.
----	---	------------------------

Postcondition

Both the calling object and "r" will not be changed.

Returns

A new [Register](#) object holding the value of the calling object plus the value in "r".

Implemented in [Register](#).

7.7.2.4 virtual Register iRegister::operator+ (const iRegister & r) const [pure virtual]

A standard add operator.

Note

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

Implemented in [Register](#).

7.7.2.5 `virtual void iRegister::Subtract (const iWord & w) [pure virtual]`

Subtracts a word of data from the calling object.

Parameters

<code>in</code>	<code>w</code>	The value to be subtracted.
-----------------	----------------	-----------------------------

Postcondition

The calling object equals its previous value minus the value of "w"; "w", however, will remain unchanged.

Implemented in [Register](#).

7.7.2.6 `virtual Register iRegister::Subtract (const iRegister & r) const [pure virtual]`

Subtracts a word of data from the calling object.

Parameters

<code>in</code>	<code>r</code>	The value to be subtracted.
-----------------	----------------	-----------------------------

Postcondition

Both the calling object and "r" will not be changed.

Returns

A new [Register](#) object holding the value of the calling object minus the value in "r".

Implemented in [Register](#).

7.7.2.7 `virtual Register iRegister::operator- (const iRegister & r) const [pure virtual]`

A standard subtraction operator.

Note

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

Implemented in [Register](#).

7.7.2.8 `virtual void iRegister::And (const iWord & w) [pure virtual]`

Performs a bit-wise and.

Parameters

<i>in</i>	<i>w</i>	The value to be "and"ed.
-----------	----------	--------------------------

Postcondition

The calling object equals its previous value bit-wise and'ed with *w*.

Implemented in [Register](#).

7.7.2.9 virtual Register iRegister::And (const iRegister & *r*) const [pure virtual]

Performs a bit-wise and.

Parameters

<i>in</i>	<i>r</i>	The value to be "and"ed.
-----------	----------	--------------------------

Postcondition

Both the calling object and *r* are not changed.

Returns

A new [Register](#) object holding the value of the calling object bit-wise and'ed with *r*.

Implemented in [Register](#).

7.7.2.10 virtual void iRegister::Or (const iWord & *w*) [pure virtual]

Performs a bit-wise "or".

Parameters

<i>in</i>	<i>w</i>	The value to be "or"ed.
-----------	----------	-------------------------

Postcondition

The calling object equals its previous value bit-wise or'ed with *w*.

Implemented in [Register](#).

7.7.2.11 virtual Register iRegister::Or (const iRegister & *r*) const [pure virtual]

Performs a bit-wise or.

Parameters

<i>in</i>	<i>r</i>	The value to be "or"ed.
-----------	----------	-------------------------

Postcondition

Both the calling object and `r` are not changed.

Returns

A new [Register](#) object holding the value of the calling object bit-wise or'ed with `r`.

Implemented in [Register](#).

7.7.2.12 virtual void iRegister::Not () [pure virtual]

Performs a bit-wise not.

Postcondition

The calling object's bits are all flipped (e.g. 1001 -> 0110).

Implemented in [Register](#).

7.7.2.13 virtual Register iRegister::Not () const [pure virtual]

Performs a bit-wise not.

Postcondition

The calling object is not changed.

Returns

A new [Register](#) object holding the bit-wise not of the calling object.

Implemented in [Register](#).

7.7.2.14 virtual void iRegister::Store (const iWord & w) [pure virtual]

Stores a word of data.

Parameters

<code>in</code>	<code>w</code>	The value to be store.
-----------------	----------------	------------------------

Postcondition

The calling object's value is now "`w`".

Implemented in [Register](#).

7.7.2.15 `virtual void iRegister::Store (const iRegister & r) [pure virtual]`

Stores a copy of another register.

Parameters

<code>in</code>	<code>r</code>	The register to be copied.
-----------------	----------------	----------------------------

Postcondition

The calling object's value is now "r".

Implemented in [Register](#).

7.7.2.16 `virtual Register& iRegister::operator= (const iWord & w) [pure virtual]`

A standard assignment operator.

Note

"r = w" is equivalent to "r.Store(w)"

Implemented in [Register](#).

7.7.2.17 `virtual Register& iRegister::operator= (const Register r) [pure virtual]`

A standard assignment operator.

Note

"r1 = r2" is equivalent to "r1.Store(r2)"

Implemented in [Register](#).

7.7.2.18 `virtual Register& iRegister::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 [Register](#).

7.7.2.19 virtual Register& iRegister::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 [Register](#).

7.8 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

7.9 iWi11 Class Reference

Public Member Functions

- virtual bool **LoadObj** (const char *)=0

- virtual void **DisplayMemory** () const =0
- virtual void **DisplayRegisters** () const =0
- virtual bool **ExecuteNext** (bool verbose=false)=0

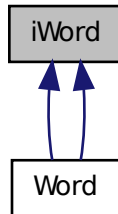
Private Member Functions

- virtual [iRegister](#) & **_GetRegister** (const Decoder::REGISTER_ID &)=0
- virtual Codes::RESULT **_Add** (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2)=0
- virtual Codes::RESULT **_Add** (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const [iWord](#) &immediate)=0
- virtual Codes::RESULT **_And** (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2)=0
- virtual Codes::RESULT **_And** (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const [iWord](#) &immediate)=0
- virtual Codes::RESULT **_Branch** (const [iWord](#) &address)=0
- virtual Codes::RESULT **_Debug** ()=0
- virtual Codes::RESULT **_JSR** (const [iWord](#) &)=0
- virtual Codes::RESULT **_JSRR** (const [iWord](#) &baseR, const [iWord](#) &address)=0
- virtual Codes::RESULT **_Load** (const Decoder::REGISTER_ID DR, const [iWord](#) &address)=0
- virtual Codes::RESULT **_Loadl** (const Decoder::REGISTER_ID DR, const [iWord](#) &address)=0
- virtual Codes::RESULT **_LoadR** (const Decoder::REGISTER_ID DR, Decoder::REGISTER_ID baseR, const [iWord](#) &address)=0
- virtual Codes::RESULT **_Not** (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR)=0
- virtual Codes::RESULT **_Ret** ()=0
- virtual Codes::RESULT **_Store** (const Decoder::REGISTER_ID SR1, const [iWord](#) &address)=0
- virtual Codes::RESULT **_STI** (const Decoder::REGISTER_ID SR1, const [iWord](#) &address)=0
- virtual Codes::RESULT **_STR** (const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID baseR, const [iWord](#) &address)=0
- virtual Codes::RESULT **_Trap** (const [iWord](#) &code)=0

7.10 iWord Class Reference

Defines 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 value)=0
"From Integer"
- virtual bool **FromStr** (const std::string &str)=0
"From String"
- virtual bool **FromHex** (const std::string &str)=0
"From Hexadecimal"
- virtual **Word Add** (const **iWord** &w) const =0
Adds two words.
- virtual **Word operator+** (const **iWord** &w) const =0
A standard addition operator.

- virtual **Word Subtract** (const **iWord** &w) const =0
Subtracts two words.
- virtual **Word operator-** (const **iWord** &w) const =0
A standard subtraction operator.
- virtual **Word And** (const **iWord** &w) const =0
"And"s the bits of two words.
- virtual **Word Or** (const **iWord** &w) const =0
"Or"s the bits of two words.
- virtual **Word Not** () const =0
"Not"s the bits of a word.
- virtual void **Copy** (const **iWord** &w)=0
Copies a word.
- virtual **Word & operator=** (const **Word** w)=0
A standard assignment operator.
- virtual **iWord & operator++** ()=0
A standard pre-increment operator.
- virtual **iWord & operator++** (int)=0
A standard post-increment operator.
- virtual bool **operator[]** (const int i) const =0
An accessor to the 'i'th bit of the value.
- 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 value)=0
"From Integer"
- virtual bool **fromStr** (const std::string &str)=0

"From String"

- virtual bool `fromHex` (const std::string &str)=0
"From Hexadecimal"
- virtual `Word Add` (const `iWord` &w) const =0
Adds two words.
- virtual `Word operator+` (const `iWord` &w) const =0
A standard addition operator.
- virtual `Word Subtract` (const `iWord` &w) const =0
Subtracts two words.
- virtual `Word operator-` (const `iWord` &w) const =0
A standard subtraction operator.
- virtual `Word And` (const `iWord` &w) const =0
"And"s the bits of two words.
- virtual `Word Or` (const `iWord` &w) const =0
"Or"s the bits of two words.
- virtual `Word Not` () const =0
"Not"s the bits of a word.
- virtual void `copy` (const `iWord` &w)=0
Copies a word.
- virtual `Word & operator=` (const `Word` w)=0
A standard assignment operator.
- virtual `iWord & operator++` ()=0
A standard pre-increment operator.
- virtual `iWord & operator++` (int)=0
A standard post-increment operator.
- virtual bool `operator[]` (const int i) const =0
An accessor to the 'i'th bit of the value.

7.10.1 Detailed Description

Defines a "word" of data on the Wi-11 Machine. The methods present in this interface 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.

7.10.2 Member Function Documentation

7.10.2.1 `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](#).

7.10.2.2 `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](#).

7.10.2.3 `virtual std::string iWord::ToStr () const [pure virtual]`

"To String"

Postcondition

The value of the word is not changed.

Returns

16 characters: each either a 1 or 0

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](#).

7.10.2.4 `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: "0xFFFFE"

Implemented in [Word](#).

7.10.2.5 `virtual bool iWord::FromInt (int value)` `[pure virtual]`

"From Integer"

Parameters

<i>in</i>	<i>value</i>	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](#).

7.10.2.6 `virtual bool iWord::FromStr (const std::string & str)` `[pure virtual]`

"From String"

Parameters

<i>in</i>	<i>str</i>	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](#).

7.10.2.7 virtual bool iWord::FromHex (const std::string & str) [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](#).

7.10.2.8 virtual Word iWord::Add (const iWord & w) 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](#), and [Word](#).

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

A standard addition operator.

Note

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

Implemented in [Word](#), and [Word](#).

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

Subtracts two words.

Parameters

<code>in</code>	<code>w</code>	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](#), and [Word](#).

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

A standard subtraction operator.

Note

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

Implemented in [Word](#), and [Word](#).

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

"And"s the bits of two words.

Parameters

<code>in</code>	<code>w</code>	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](#), and [Word](#).

7.10.2.13 `virtual Word iWord::Or (const iWord & w) const` [pure virtual]

"Or"s the bits of two words.

Parameters

<code>in</code>	<code>w</code>	A word value to be "or"ed.
-----------------	----------------	----------------------------

Postcondition

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

Returns

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

Implemented in [Word](#), and [Word](#).

7.10.2.14 `virtual Word iWord::Not () const` [pure virtual]

"Not"s the bits of a word.

Postcondition

The calling object do not change.

Returns

A new "Word" object containing the result of performing a bit-wise not on the calling object.

Implemented in [Word](#), and [Word](#).

7.10.2.15 `virtual void iWord::Copy (const iWord & w)` [pure virtual]

Copies a word.

Parameters

out	w	The value to be copied.
-----	---	-------------------------

Postcondition

The caller equals that parameter.

Equivalent to the assignment "caller = parameter".

Implemented in [Word](#).

7.10.2.16 virtual Word& iWord::operator=(const Word w) [pure virtual]

A standard assignment operator.

Parameters

in	w	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](#), and [Word](#).

7.10.2.17 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](#), and [Word](#).

7.10.2.18 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](#), and [Word](#).

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

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

Parameters

<code>in</code>	<code>i</code>	The index of the bit in question.
-----------------	----------------	-----------------------------------

Precondition

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

Returns

True \Leftrightarrow 1, False \Leftrightarrow 0.

The number of the bits starts at zero and rises into the more significant bits.

Examples:

If the object holds a value of 4 (0...100 in binary): `num[2] = 1`.

If it holds a value of 1 (0...001 in binary): `num[0] = 1`.

If it holds a negative value (Starting with a 1 in 2's complement): `num[15] = 1`.

Implemented in [Word](#), and [Word](#).

7.10.2.20 `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](#).

7.10.2.21 `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](#).

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

"To String"

Postcondition

The value of the word is not changed.

Returns

16 characters: each either a 1 or 0

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](#).

7.10.2.23 `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: "0xFFFFE"

Implemented in [Word](#).

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

"From Integer"

Parameters

<i>in</i>	<i>value</i>	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](#).

7.10.2.25 virtual bool iWord::fromStr (const std::string & *str*) [pure virtual]

"From String"

Parameters

<i>in</i>	<i>str</i>	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](#).

7.10.2.26 virtual bool iWord::fromHex (const std::string & *str*) [pure virtual]

"From Hexadecimal"

Parameters

<i>in</i>	<i>str</i>	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](#).

7.10.2.27 virtual Word iWord::Add (const iWord & w) 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](#), and [Word](#).

7.10.2.28 virtual Word iWord::operator+ (const iWord & w) const [pure virtual]

A standard addition operator.

Note

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

Implemented in [Word](#), and [Word](#).

7.10.2.29 virtual Word iWord::Subtract (const iWord & w) 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](#), and [Word](#).

7.10.2.30 virtual Word iWord::operator- (const iWord & w) const [pure virtual]

A standard subtraction operator.

Note

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

Implemented in [Word](#), and [Word](#).

7.10.2.31 virtual Word iWord::And (const iWord & w) 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](#), and [Word](#).

7.10.2.32 virtual Word iWord::Or (const iWord & w) const [pure virtual]

"Or"s the bits of two words.

Parameters

in	w	A word value to be "or"ed.
----	---	----------------------------

Postcondition

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

Returns

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

Implemented in [Word](#), and [Word](#).

7.10.2.33 virtual Word iWord::Not () const [pure virtual]

"Not"s the bits of a word.

Postcondition

The calling object do not change.

Returns

A new "Word" object containing the result of performing a bit-wise not on the calling object.

Implemented in [Word](#), and [Word](#).

7.10.2.34 virtual void iWord::copy (const iWord & w) [pure virtual]

Copies a word.

Parameters

out	w	The value to be copied.
-----	---	-------------------------

Postcondition

The caller equals that parameter.

Equivalent to the assignment "caller = parameter".

Implemented in [Word](#).

7.10.2.35 virtual Word& iWord::operator= (const Word w) [pure virtual]

A standard assignment operator.

Parameters

in	w	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](#), and [Word](#).

7.10.2.36 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](#), and [Word](#).

7.10.2.37 `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](#), and [Word](#).

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

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

Parameters

<code>in</code>	<code>i</code>	The index of the bit in question.
-----------------	----------------	-----------------------------------

Precondition

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

Returns

True \Leftrightarrow 1, False \Leftrightarrow 0.

The number of the bits starts at zero and rises into the more significant bits.

Examples:

If the object holds a value of 4 (0...100 in binary): `num[2] = 1`.

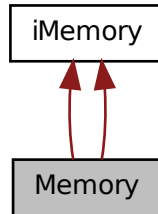
If it holds a value of 1 (0...001 in binary): `num[0] = 1`.

If it holds a negative value (Starting with a 1 in 2's complement): `num[15] = 1`.

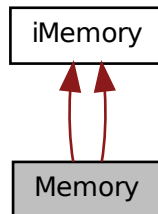
Implemented in [Word](#), and [Word](#).

7.11 Memory Class Reference

Inheritance diagram for Memory:



Collaboration diagram for Memory:



Public Member Functions

- virtual Codes::RESULT **Reserve** (const `iWord` &initial_address, const `iWord` &length)
- virtual `Word Load` (const `iWord` &) const
- virtual Codes::RESULT **Store** (const `iWord` &address, const `Word` &value)
- virtual Codes::RESULT **Reserve** (const `iWord` &initial_address, const `iWord` &length)
- virtual `Word Load` (const `iWord` &) const
- virtual Codes::RESULT **Store** (const `iWord` &address, const `Word` &value)

Private Attributes

- `std::vector< Word * > _bounded_memory`
- `std::vector< int > _segment_offsets`
- `std::vector< int > _segment_lengths`
- `std::map< int, Word > _unbounded_memory`

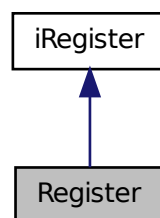
7.12 ObjectData Struct Reference

Public Attributes

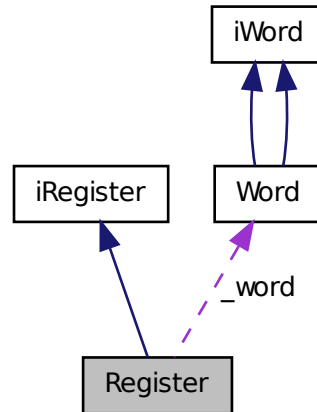
- `char type`
- `std::vector< std::string > data`

7.13 Register Class Reference

Inheritance diagram for Register:



Collaboration diagram for Register:



Public Member Functions

- **Register** (const **Word** w)
- **Word GetValue** () const
Retrieves a copy of the word of data store in the register.
- void **Add** (const **iWord** &w)
Adds a word of data to the calling object.
- **Register Add** (const **iRegister** &r) const
Adds a word of data to the calling object.
- **Register operator+** (const **iRegister** &r) const
A standard add operator.
- void **Subtract** (const **iWord** &w)
Subtracts a word of data from the calling object.
- **Register Subtract** (const **iRegister** &r) const
Subtracts a word of data from the calling object.
- **Register operator-** (const **iRegister** &r) const
A standard subtraction operator.

- void **And** (const **iWord** &w)
Performs a bit-wise and.
- **Register And** (const **iRegister** &r) const
Performs a bit-wise and.
- void **Or** (const **iWord** &w)
Performs a bit-wise "or".
- **Register Or** (const **iRegister** &r) const
Performs a bit-wise or.
- void **Not** ()
Performs a bit-wise not.
- **Register Not** () const
Performs a bit-wise not.
- void **Store** (const **iWord** &w)
Stores a word of data.
- void **Store** (const **iRegister** &r)
Stores a copy of another register.
- **Register & operator=** (const **iWord** &w)
A standard assignment operator.
- **Register & operator=** (const **Register** r)
A standard assignment operator.
- **Register & operator++** ()
A standard pre-increment operator.
- **Register & operator++** (int)
A standard post-increment operator.

Private Attributes

- **Word _word**
The word of data held in the register.

7.13.1 Member Function Documentation

7.13.1.1 `Word Register::GetValue () const` [virtual]

Retrieves a copy of the word of data store in the register.

Postcondition

The value of the calling object is not changed.

Returns

A new [Word](#) object holding the value that is stored in the register.

Implements [iRegister](#).

7.13.1.2 `void Register::Add (const iWord & w)` [virtual]

Adds a word of data to the calling object.

Parameters

<code>in</code>	<code>w</code>	The value to be added.
-----------------	----------------	------------------------

Postcondition

The calling object equals its previous value plus the value of "w"; "w", however, will remain unchanged.

Implements [iRegister](#).

7.13.1.3 `Register Register::Add (const iRegister & r) const` [virtual]

Adds a word of data to the calling object.

Parameters

<code>in</code>	<code>r</code>	The value to be added.
-----------------	----------------	------------------------

Postcondition

Both the calling object and "r" will not be changed.

Returns

A new [Register](#) object holding the value of the calling object plus the value in "r".

Implements [iRegister](#).

7.13.1.4 Register Register::operator+ (const iRegister & r) const [virtual]

A standard add operator.

Note

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

Implements [iRegister](#).

7.13.1.5 void Register::Subtract (const iWord & w) [virtual]

Subtracts a word of data from the calling object.

Parameters

in	w	The value to be subtracted.
----	---	-----------------------------

Postcondition

The calling object equals its previous value minus the value of "w"; "w", however, will remain unchanged.

Implements [iRegister](#).

7.13.1.6 Register Register::Subtract (const iRegister & r) const [virtual]

Subtracts a word of data from the calling object.

Parameters

in	r	The value to be subtracted.
----	---	-----------------------------

Postcondition

Both the calling object and "r" will not be changed.

Returns

A new [Register](#) object holding the value of the calling object minus the value in "r".

Implements [iRegister](#).

7.13.1.7 Register Register::operator- (const iRegister & r) const [virtual]

A standard subtraction operator.

Note

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

Implements [iRegister](#).

7.13.1.8 void Register::And (const iWord & w) [virtual]

Performs a bit-wise and.

Parameters

in	w	The value to be "and"ed.
----	---	--------------------------

Postcondition

The calling object equals its previous value bit-wise and'ed with w.

Implements [iRegister](#).

7.13.1.9 Register Register::And (const iRegister & r) const [virtual]

Performs a bit-wise and.

Parameters

in	r	The value to be "and"ed.
----	---	--------------------------

Postcondition

Both the calling object and r are not changed.

Returns

A new [Register](#) object holding the value of the calling object bit-wise and'ed with r.

Implements [iRegister](#).

7.13.1.10 void Register::Or (const iWord & w) [virtual]

Performs a bit-wise "or".

Parameters

in	w	The value to be "or"ed.
----	---	-------------------------

Postcondition

The calling object equals its previous value bit-wise or'ed with w.

Implements [iRegister](#).

7.13.1.11 Register Register::Or (const iRegister & *r*) const [virtual]

Performs a bit-wise or.

Parameters

in	<i>r</i>	The value to be "or"ed.
----	----------	-------------------------

Postcondition

Both the calling object and *r* are not changed.

Returns

A new [Register](#) object holding the value of the calling object bit-wise or'ed with *r*.

Implements [iRegister](#).

7.13.1.12 void Register::Not () [virtual]

Performs a bit-wise not.

Postcondition

The calling object's bits are all flipped (e.g. 1001 -> 0110).

Implements [iRegister](#).

7.13.1.13 Register Register::Not () const [virtual]

Performs a bit-wise not.

Postcondition

The calling object is not changed.

Returns

A new [Register](#) object holding the bit-wise not of the calling object.

Implements [iRegister](#).

7.13.1.14 void Register::Store (const iWord & *w*) [virtual]

Stores a word of data.

Parameters

in	<i>w</i>	The value to be store.
----	----------	------------------------

Postcondition

The calling object's value is now "w".

Implements [iRegister](#).

7.13.1.15 void Register::Store (const iRegister & r) [virtual]

Stores a copy of another register.

Parameters

in	r	The register to be copied.
----	---	----------------------------

Postcondition

The calling object's value is now "r".

Implements [iRegister](#).

7.13.1.16 Register & Register::operator= (const iWord & w) [virtual]

A standard assignment operator.

Note

"r = w" is equivalent to "r.Store(w)"

Implements [iRegister](#).

7.13.1.17 Register & Register::operator= (const Register r) [virtual]

A standard assignment operator.

Note

"r1 = r2" is equivalent to "r1.Store(r2)"

Implements [iRegister](#).

7.13.1.18 Register & Register::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 [iRegister](#).

7.13.1.19 Register & Register::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 [iRegister](#).

7.14 ResultDecoder Class Reference

Public Member Functions

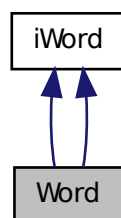
- std::string **Find** (const Codes::RESULT &) const
- std::string **Find** (const Codes::RESULT &) const

Static Private Attributes

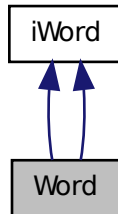
- static std::map< Codes::RESULT, std::string > **_codes**

7.15 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 value)`
"From Integer"
- `bool fromStr (const std::string &str)`
"From String"
- `bool fromHex (const std::string &str)`
"From Hexadecimal"
- `Word Add (const iWord &w) const`
Adds two words.
- `Word operator+ (const iWord &w) const`
A standard addition operator.

- **Word Subtract** (const **iWord** &w) const
Subtracts two words.
- **Word operator-** (const **iWord** &w) const
A standard subtraction operator.
- **Word And** (const **iWord** &w) const
"And"s the bits of two words.
- **Word Or** (const **iWord** &w) const
"Or"s the bits of two words.
- **Word Not** () const
"Not"s the bits of a word.
- void **copy** (const **iWord** &w)
Copies a word.
- **Word & operator=** (const **Word** w)
A standard assignment operator.
- **iWord & operator++** ()
A standard pre-increment operator.
- **iWord & operator++** (int)
A standard post-increment operator.
- bool **operator[]** (const int i) const
An accessor to the 'i'th bit of the value.
- 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 value)
"From Integer"
- bool **FromStr** (const std::string &str)

"From String"

- bool [FromHex](#) (const std::string &str)

"From Hexadecimal"

- [Word Add](#) (const [iWord](#) &w) const

Adds two words.

- [Word operator+](#) (const [iWord](#) &w) const

A standard addition operator.

- [Word Subtract](#) (const [iWord](#) &w) const

Subtracts two words.

- [Word operator-](#) (const [iWord](#) &w) const

A standard subtraction operator.

- [Word And](#) (const [iWord](#) &w) const

"And"s the bits of two words.

- [Word Or](#) (const [iWord](#) &w) const

"Or"s the bits of two words.

- [Word Not](#) () const

"Not"s the bits of a word.

- void [Copy](#) (const [iWord](#) &w)

Copies a word.

- [Word & operator=](#) (const [Word](#) w)

A standard assignment operator.

- [iWord & operator++](#) ()

A standard pre-increment operator.

- [iWord & operator++](#) (int)

A standard post-increment operator.

- bool [operator\[\]](#) (const int i) const

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

Private Member Functions

- `bool _hasBit (int) const`
Tests for powers of two in binary representation.
- `bool _HasBit (int) const`
Tests for powers of two in binary representation.

Private Attributes

- `unsigned short _value`
Used to store the "word" of data.

7.15.1 Member Function Documentation

7.15.1.1 `bool Word::hasBit (int i) const` [private]

Tests for powers of two in binary representation.

Parameters

<code>i</code>	The index of the digit desired from the binary representation of <code>_word</code> .
----------------	---

Returns

True if and only if the `i`'th bit is 1.

The indexing of the bits works as defined in [operator\[\]\(\)](#).

7.15.1.2 `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](#).

7.15.1.3 `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](#).

7.15.1.4 string Word::toStr () const [virtual]

"To String"

Postcondition

The value of the word is not changed.

Returns

16 characters: each either a 1 or 0

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](#).

7.15.1.5 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: "0xFFFFE"

Implements [iWord](#).

7.15.1.6 bool Word::fromInt (int value) [virtual]

"From Integer"

Parameters

<i>in</i>	<i>value</i>	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](#).

7.15.1.7 bool Word::fromStr (const std::string & *str*) [virtual]

"From String"

Parameters

<i>in</i>	<i>str</i>	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](#).

7.15.1.8 bool Word::fromHex (const std::string & *str*) [virtual]

"From Hexadecimal"

Parameters

<i>in</i>	<i>str</i>	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](#).

7.15.1.9 Word Word::Add (const iWord & w) 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](#).

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

A standard addition operator.

Note

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

Implements [iWord](#).

7.15.1.11 Word Word::Subtract (const iWord & w) 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](#).

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

A standard subtraction operator.

Note

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

Implements [iWord](#).

7.15.1.13 Word Word::And (const iWord & w) 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](#).

7.15.1.14 Word Word::Or (const iWord & w) const [virtual]

"Or"s the bits of two words.

Parameters

in	w	A word value to be "or"ed.
----	---	----------------------------

Postcondition

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

Returns

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

Implements [iWord](#).

7.15.1.15 Word Word::Not () const [virtual]

"Not"s the bits of a word.

Postcondition

The calling object do not change.

Returns

A new "Word" object containing the result of performing a bit-wise not on the calling object.

Implements [iWord](#).

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

Copies a word.

Parameters

out	w	The value to be copied.
-----	---	-------------------------

Postcondition

The caller equals that parameter.

Equivalent to the assignment "caller = parameter".

Implements [iWord](#).

7.15.1.17 Word & Word::operator= (const Word w) [virtual]

A standard assignment operator.

Parameters

in	w	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](#).

7.15.1.18 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](#).

7.15.1.19 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](#).

7.15.1.20 bool Word::operator[] (const int i) const [virtual]

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

Parameters

in	i	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 holds a value of 4 (0...100 in binary): num[2] = 1.

If it holds a value of 1 (0...001 in binary): num[0] = 1.

If it holds a negative value (Starting with a 1 in 2's complement): num[15] = 1.

Implements [iWord](#).

7.15.1.21 `bool Word::HasBit (int i) const` [private]

Tests for powers of two in binary representation.

Parameters

<i>i</i>	The index of the digit desired from the binary representation of <code>_word</code> .
----------	---

Returns

True if and only if the *i*'th bit is 1.

The indexing of the bits works as defined in [operator\[\]\(\)](#).

7.15.1.22 `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](#).

7.15.1.23 `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](#).

7.15.1.24 `string Word::ToStr () const` [virtual]

"To String"

Postcondition

The value of the word is not changed.

Returns

16 characters: each either a 1 or 0

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](#).

7.15.1.25 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](#).

7.15.1.26 bool Word::FromInt (int value) [virtual]

"From Integer"

Parameters

<i>in</i>	<i>value</i>	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](#).

7.15.1.27 `bool Word::FromStr (const std::string & str)` [virtual]

"From String"

Parameters

<i>in</i>	<i>str</i>	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](#).

7.15.1.28 `bool Word::FromHex (const std::string & str)` [virtual]

"From Hexadecimal"

Parameters

<i>in</i>	<i>str</i>	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](#).

7.15.1.29 `Word Word::Add (const iWord & w) const` [virtual]

Adds two words.

Parameters

<i>in</i>	<i>w</i>	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](#).

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

A standard addition operator.

Note

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

Implements [iWord](#).

7.15.1.31 Word Word::Subtract (const iWord & w) 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](#).

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

A standard subtraction operator.

Note

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

Implements [iWord](#).

7.15.1.33 Word Word::And (const iWord & w) 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](#).

7.15.1.34 Word Word::Or (const iWord & w) const [virtual]

"Or"s the bits of two words.

Parameters

in	w	A word value to be "or"ed.
----	---	----------------------------

Postcondition

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

Returns

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

Implements [iWord](#).

7.15.1.35 Word Word::Not () const [virtual]

"Not"s the bits of a word.

Postcondition

The calling object do not change.

Returns

A new "Word" object containing the result of performing a bit-wise not on the calling object.

Implements [iWord](#).

7.15.1.36 `void Word::Copy (const iWord & w) [virtual]`

Copies a word.

Parameters

out	<i>w</i>	The value to be copied.
-----	----------	-------------------------

Postcondition

The caller equals that parameter.

Equivalent to the assignment "caller = parameter".

Implements [iWord](#).

7.15.1.37 `Word& Word::operator= (const Word w) [virtual]`

A standard assignment operator.

Parameters

in	<i>w</i>	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](#).

7.15.1.38 `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](#).

7.15.1.39 `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](#).

7.15.1.40 bool Word::operator[] (const int i) const [virtual]

An accessor to the *i*'th bit of the value.

Parameters

<i>in</i>	<i>i</i>	The index of the bit in question.
-----------	----------	-----------------------------------

Precondition

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

Returns

True \Leftrightarrow 1, False \Leftrightarrow 0.

The number of the bits starts at zero and rises into the more significant bits.

Examples:

If the object holds a value of 4 (0...100 in binary): num[2] = 1.

If it holds a value of 1 (0...001 in binary): num[0] = 1.

If it holds a negative value (Starting with a 1 in 2's complement): num[15] = 1.

Implements [iWord](#).

7.15.2 Member Data Documentation**7.15.2.1 unsigned short Word::_value [private]**

Used to store the "word" of data.

The type "unsigned short" was chosen because in c++, shorts are 16bits (the same size as our words) and having it unsigned allows for easy "reading" as a positive int or a 2's complement int.

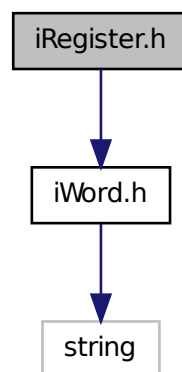
Chapter 8

File Documentation

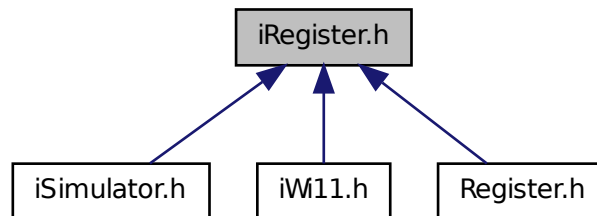
8.1 iRegister.h File Reference

Definition of a "register" in the Wi-11 machine.

Include dependency graph for iRegister.h:



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



Classes

- class [iRegister](#)

Defines a "register" in the Wi-11 machine.

8.1.1 Detailed Description

Definition of a "register" in the Wi-11 machine.

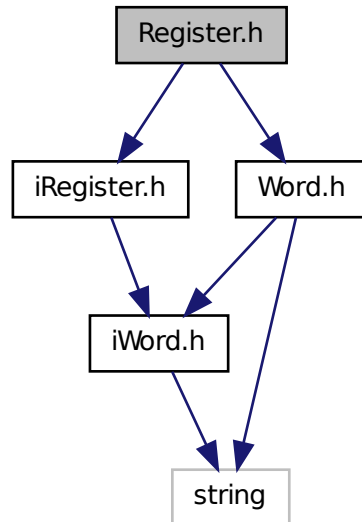
Author

Joshua Green
Andrew Groot

8.2 Register.h File Reference

Definition of private data for the "Register" class.

Include dependency graph for Register.h:



Classes

- class [Register](#)

8.2.1 Detailed Description

Definition of private data for the "Register" class.

Author

Andrew Groot

Index

`_HasBit`
 Word, 62
`_hasBit`
 Word, 55
`_value`
 Word, 68

`Add`
 iRegister, 19
 iWord, 31, 37
 Register, 46
 Word, 58, 64

`And`
 iRegister, 20, 21
 iWord, 32, 39
 Register, 48
 Word, 59, 66

`code/ Directory Reference, 11`
`code/MemoryTest/ Directory Reference, 12`
`code/test/ Directory Reference, 13`

`Copy`
 iWord, 33
 Word, 67

`copy`
 iWord, 40
 Word, 60

`FromHex`
 iWord, 31
 Word, 64

`fromHex`
 iWord, 37
 Word, 57

`FromInt`
 iWord, 30
 Word, 63

`fromInt`
 iWord, 36
 Word, 56

`FromStr`
 iWord, 30
 Word, 64

`fromStr`
 iWord, 37
 Word, 57

`GetValue`
 iRegister, 18
 Register, 46

`iDecoder, 15`
`iInterpreter, 15`
`iLoader, 15`
`iMemory, 16`
`Instruction, 16`
`iObjParser, 16`
`iRegister, 17`
 Add, 19
 And, 20, 21
 GetValue, 18
 Not, 22
 operator+, 19
 operator++, 23
 operator-, 20
 operator=, 23
 Or, 21
 Store, 22
 Subtract, 20

`iRegister.h, 69`
`iSimulator, 24`
`iWi11, 24`
`iWord, 25`
 Add, 31, 37
 And, 32, 39
 Copy, 33
 copy, 40
 FromHex, 31
 fromHex, 37
 FromInt, 30
 fromInt, 36
 FromStr, 30

- fromStr, [37](#)
 - Not, [33](#), [39](#)
 - operator+, [31](#), [38](#)
 - operator++, [34](#), [40](#), [41](#)
 - operator-, [32](#), [38](#)
 - operator=, [34](#), [40](#)
 - Or, [33](#), [39](#)
 - Subtract, [32](#), [38](#)
 - ToHex, [29](#)
 - toHex, [36](#)
 - ToInt, [29](#)
 - toInt, [35](#)
 - ToInt2Complement, [29](#)
 - toInt2Complement, [35](#)
 - ToStr, [29](#)
 - toStr, [35](#)
- Memory, [42](#)
- Not
- iRegister, [22](#)
 - iWord, [33](#), [39](#)
 - Register, [49](#)
 - Word, [60](#), [66](#)
- ObjectData, [43](#)
- operator+
- iRegister, [19](#)
 - iWord, [31](#), [38](#)
 - Register, [46](#)
 - Word, [58](#), [65](#)
- operator++
- iRegister, [23](#)
 - iWord, [34](#), [40](#), [41](#)
 - Register, [50](#)
 - Word, [61](#), [67](#)
- operator-
- iRegister, [20](#)
 - iWord, [32](#), [38](#)
 - Register, [47](#)
 - Word, [59](#), [65](#)
- operator=
- iRegister, [23](#)
 - iWord, [34](#), [40](#)
 - Register, [50](#)
 - Word, [60](#), [67](#)
- Or
- iRegister, [21](#)
 - iWord, [33](#), [39](#)
 - Register, [48](#)
- Word, [59](#), [66](#)
- Register, [43](#)
- Add, [46](#)
 - And, [48](#)
 - GetValue, [46](#)
 - Not, [49](#)
 - operator+, [46](#)
 - operator++, [50](#)
 - operator-, [47](#)
 - operator=, [50](#)
 - Or, [48](#)
 - Store, [49](#), [50](#)
 - Subtract, [47](#)
- Register.h, [70](#)
- ResultDecoder, [51](#)
- Store
- iRegister, [22](#)
 - Register, [49](#), [50](#)
- Subtract
- iRegister, [20](#)
 - iWord, [32](#), [38](#)
 - Register, [47](#)
 - Word, [58](#), [65](#)
- ToHex
- iWord, [29](#)
 - Word, [63](#)
- toHex
- iWord, [36](#)
 - Word, [56](#)
- ToInt
- iWord, [29](#)
 - Word, [62](#)
- toInt
- iWord, [35](#)
 - Word, [55](#)
- ToInt2Complement
- iWord, [29](#)
 - Word, [62](#)
- toInt2Complement
- iWord, [35](#)
 - Word, [55](#)
- ToStr
- iWord, [29](#)
 - Word, [62](#)
- toStr
- iWord, [35](#)
 - Word, [56](#)

Word, [51](#)
 _hasBit, [62](#)
 hasBit, [55](#)
 _value, [68](#)
Add, [58](#), [64](#)
And, [59](#), [66](#)
Copy, [67](#)
copy, [60](#)
FromHex, [64](#)
fromHex, [57](#)
FromInt, [63](#)
fromInt, [56](#)
FromStr, [64](#)
fromStr, [57](#)
Not, [60](#), [66](#)
operator+, [58](#), [65](#)
operator++, [61](#), [67](#)
operator-, [59](#), [65](#)
operator=, [60](#), [67](#)
Or, [59](#), [66](#)
Subtract, [58](#), [65](#)
ToHex, [63](#)
toHex, [56](#)
ToInt, [62](#)
toInt, [55](#)
ToInt2Complement, [62](#)
toInt2Complement, [55](#)
ToStr, [62](#)
toStr, [56](#)