

OneUp Wi-11 Simulator

Generated by Doxygen 1.7.2

Sat Jan 22 2011 01:05:32

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	3
1.3.1	Components	3
1.3.2	Wi11 Instruction Set	3
2	Directory Hierarchy	5
2.1	Directories	5
3	Class Index	7
3.1	Class Hierarchy	7
4	Class Index	9
4.1	Class List	9
5	File Index	11
5.1	File List	11
6	Directory Documentation	13
6.1	code/ Directory Reference	13
6.2	code/MemoryTest/ Directory Reference	14
6.3	code/test/ Directory Reference	15
7	Class Documentation	17
7.1	iDecoder Class Reference	17
7.2	iInterpreter Class Reference	17
7.3	iLoader Class Reference	17
7.4	iMemory Class Reference	18
7.5	Instruction Struct Reference	18
7.6	iObjParser Class Reference	18
7.7	iRegister Class Reference	19
7.7.1	Detailed Description	20
7.7.2	Member Function Documentation	20
7.7.2.1	GetValue	20
7.7.2.2	Add	21
7.7.2.3	Add	21

7.7.2.4	operator+	21
7.7.2.5	Subtract	22
7.7.2.6	Subtract	22
7.7.2.7	operator-	22
7.7.2.8	And	22
7.7.2.9	And	23
7.7.2.10	Or	23
7.7.2.11	Or	23
7.7.2.12	Not	24
7.7.2.13	Not	24
7.7.2.14	Store	24
7.7.2.15	Store	25
7.7.2.16	operator=	25
7.7.2.17	operator=	25
7.7.2.18	operator++	25
7.7.2.19	operator++	26
7.8	iSimulator Class Reference	26
7.9	iWi11 Class Reference	26
7.9.1	Detailed Description	29
7.9.2	Constructor & Destructor Documentation	29
7.9.2.1	iWi11	29
7.9.3	Member Function Documentation	29
7.9.3.1	_GetRegister	29
7.9.3.2	_Add	30
7.9.3.3	_Add	30
7.9.3.4	_And	31
7.9.3.5	_And	31
7.9.3.6	_Branch	32
7.9.3.7	_Debug	32
7.9.3.8	_JSR	32
7.9.3.9	_JSRR	33
7.9.3.10	_Load	33
7.9.3.11	_Loadl	34
7.9.3.12	_LoadR	34
7.9.3.13	_Not	35
7.9.3.14	_Ret	35
7.9.3.15	_Store	36
7.9.3.16	_STl	36
7.9.3.17	_STR	36
7.9.3.18	_Trap	37
7.9.3.19	LoadObj	38
7.9.3.20	DisplayMemory	38
7.9.3.21	DisplayRegisters	38
7.9.3.22	ExecuteNext	39
7.10	iWord Class Reference	39
7.10.1	Detailed Description	42
7.10.2	Member Function Documentation	42
7.10.2.1	ToInt	42
7.10.2.2	ToInt2Complement	43
7.10.2.3	ToStr	43

7.10.2.4	ToHex	43
7.10.2.5	FromInt	44
7.10.2.6	FromStr	44
7.10.2.7	FromHex	44
7.10.2.8	Add	45
7.10.2.9	operator+	45
7.10.2.10	Subtract	45
7.10.2.11	operator-	46
7.10.2.12	And	46
7.10.2.13	Or	46
7.10.2.14	Not	47
7.10.2.15	Copy	47
7.10.2.16	operator=	47
7.10.2.17	operator++	48
7.10.2.18	operator++	48
7.10.2.19	operator[]	48
7.10.2.20	toInt	49
7.10.2.21	toInt2Complement	49
7.10.2.22	toStr	49
7.10.2.23	toHex	50
7.10.2.24	fromInt	50
7.10.2.25	fromStr	50
7.10.2.26	fromHex	51
7.10.2.27	Add	51
7.10.2.28	operator+	52
7.10.2.29	Subtract	52
7.10.2.30	operator-	52
7.10.2.31	And	52
7.10.2.32	Or	53
7.10.2.33	Not	53
7.10.2.34	copy	53
7.10.2.35	operator=	54
7.10.2.36	operator++	54
7.10.2.37	operator++	54
7.10.2.38	operator[]	55
7.11	Memory Class Reference	55
7.12	ObjectData Struct Reference	56
7.13	Register Class Reference	57
7.13.1	Member Function Documentation	59
7.13.1.1	GetValue	59
7.13.1.2	Add	59
7.13.1.3	Add	60
7.13.1.4	operator+	60
7.13.1.5	Subtract	60
7.13.1.6	Subtract	60
7.13.1.7	operator-	61
7.13.1.8	And	61
7.13.1.9	And	61
7.13.1.10	Or	62
7.13.1.11	Or	62

7.13.1.12	Not	62
7.13.1.13	Not	62
7.13.1.14	Store	63
7.13.1.15	Store	63
7.13.1.16	operator=	63
7.13.1.17	operator=	63
7.13.1.18	operator++	64
7.13.1.19	operator++	64
7.14	ResultDecoder Class Reference	64
7.15	Wi11 Class Reference	65
7.15.1	Member Function Documentation	68
7.15.1.1	_GetRegister	68
7.15.1.2	_Add	68
7.15.1.3	_Add	69
7.15.1.4	_And	69
7.15.1.5	_And	70
7.15.1.6	_Branch	70
7.15.1.7	_Debug	70
7.15.1.8	_JSR	71
7.15.1.9	_JSRR	71
7.15.1.10	_Load	71
7.15.1.11	_Loadl	72
7.15.1.12	_LoadR	72
7.15.1.13	_Not	73
7.15.1.14	_Ret	73
7.15.1.15	_Store	74
7.15.1.16	_STI	74
7.15.1.17	_STR	75
7.15.1.18	_Trap	75
7.15.1.19	LoadObj	76
7.15.1.20	DisplayMemory	76
7.15.1.21	DisplayRegisters	77
7.15.1.22	ExecuteNext	77
7.16	Word Class Reference	78
7.16.1	Member Function Documentation	81
7.16.1.1	_hasBit	81
7.16.1.2	toInt	82
7.16.1.3	toInt2Complement	82
7.16.1.4	toStr	82
7.16.1.5	toHex	83
7.16.1.6	fromInt	83
7.16.1.7	fromStr	83
7.16.1.8	fromHex	84
7.16.1.9	Add	84
7.16.1.10	operator+	85
7.16.1.11	Subtract	85
7.16.1.12	operator-	85
7.16.1.13	And	85
7.16.1.14	Or	86
7.16.1.15	Not	86

7.16.1.16	copy	86
7.16.1.17	operator=	87
7.16.1.18	operator++	87
7.16.1.19	operator++	87
7.16.1.20	operator[]	88
7.16.1.21	_HasBit	88
7.16.1.22	ToInt	88
7.16.1.23	ToInt2Complement	89
7.16.1.24	ToStr	89
7.16.1.25	ToHex	89
7.16.1.26	FromInt	90
7.16.1.27	FromStr	90
7.16.1.28	FromHex	90
7.16.1.29	Add	91
7.16.1.30	operator+	91
7.16.1.31	Subtract	91
7.16.1.32	operator-	92
7.16.1.33	And	92
7.16.1.34	Or	92
7.16.1.35	Not	93
7.16.1.36	Copy	93
7.16.1.37	operator=	93
7.16.1.38	operator++	94
7.16.1.39	operator++	94
7.16.1.40	operator[]	94
7.16.2	Member Data Documentation	95
7.16.2.1	_value	95
8	File Documentation	97
8.1	iRegister.h File Reference	97
8.1.1	Detailed Description	98
8.2	iWi11.h File Reference	98
8.2.1	Detailed Description	100
8.3	Register.h File Reference	100
8.3.1	Detailed Description	101
8.4	Wi11.h File Reference	101
8.4.1	Detailed Description	102

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 of 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

1.3.1 Components

1.3.2 Wi11 Instruction Set

Chapter 2

Directory Hierarchy

2.1 Directories

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

code	13
MemoryTest	14
test	15

Chapter 3

Class Index

3.1 Class Hierarchy

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

iDecoder	17
iInterpreter	17
iLoader	17
iMemory	18
Memory	55
Memory	55
Instruction	18
iObjParser	18
iRegister	19
Register	57
iSimulator	26
iWi11	26
Wi11	65
iWord	39
Word	78
Word	78
ObjectData	56
ResultDecoder	64

Chapter 4

Class Index

4.1 Class List

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

iDecoder	17
iInterpreter	17
iLoader	17
iMemory	18
Instruction	18
iObjParser	18
iRegister (Defines a "register" in the Wi-11 machine)	19
iSimulator	26
iWi11 (Defines the internal logic of the Wi-11)	26
iWord (Defines a "word" of data on the Wi-11 Machine)	39
Memory	55
ObjectData	56
Register	57
ResultDecoder	64
Wi11	65
Word	78

Chapter 5

File Index

5.1 File List

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

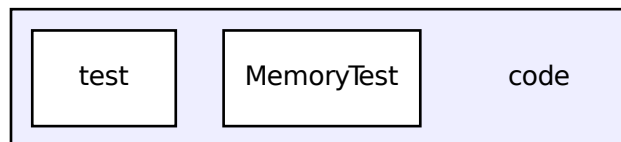
iDecoder.h	??
iInterpreter.h	??
iLoader.h	??
iMemory.h	??
MemoryTest/iMemory.h	??
iObjParser.h	??
iRegister.h (Definition of a "register" in the Wi-11 machine)	97
iSimulator.h	??
iWi11.h (Definition of the Wi-11 machine simulator)	98
iWord.h	??
MemoryTest/iWord.h	??
Memory.h	??
MemoryTest/Memory.h	??
Register.h (Definition of private data for the "Register" class)	100
MemoryTest/ResultCodes.h	??
ResultCodes.h	??
Wi11.h (Definition of the private data for the "Wi11" class)	101
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](#)

Definition of the Wi-11 machine simulator.

- 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.

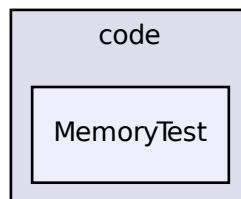
- file [ResultCodes.h](#)
- file [Wi11.h](#)

Definition of the private data for the "Wi11" class.

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

6.2 code/MemoryTest/ Directory Reference

Directory dependency graph for code/MemoryTest/:



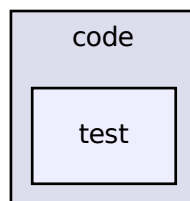
Files

- 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

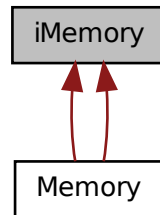
7.3 iLoader Class Reference

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

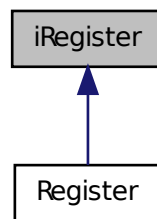
Public Member Functions

- 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

in	w	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

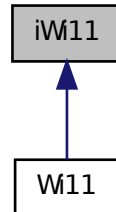
Public Member Functions

- 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

Defines the internal logic of the Wi-11.

Inheritance diagram for iWi11:



Public Member Functions

- virtual `iWi11` ()=0
Creates and organizes the componts of the `Wi11` machine.
- virtual bool `LoadObj` (const char *filename)=0
Loads the object file and sets up memory as it describes.
- virtual void `DisplayMemory` () const =0
Prints the state of memory to standard out.
- virtual void `DisplayRegisters` () const =0
Prints the state of every register to standard out.
- virtual bool `ExecuteNext` (bool verbose=false)=0
Executes the instruction pointed to by the PC.

Private Member Functions

- virtual `iRegister` & `_GetRegister` (const Decoder::REGISTER_ID &id)=0
Retrieves a reference to the register corresponding to "id".
- virtual Codes::RESULT `_Add` (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2)=0
Adds two registers and stores the result in a third.
- virtual Codes::RESULT `_Add` (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const `iWord` &immediate)=0

Adds a constant to a register and stores the result in another.

- virtual Codes::RESULT [_And](#) (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2)=0

Bit-wise ands two registers and stores the result in a third.

- virtual Codes::RESULT [_And](#) (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const [iWord](#) &immediate)=0

Bit-wise ands a register with a constant and stores the result in another register.

- virtual Codes::RESULT [_Branch](#) (const [iWord](#) &address)=0

Changes the last 9 bits of the PC.

- virtual Codes::RESULT [_Debug](#) ()=0

Deprecated?

- virtual Codes::RESULT [_JSR](#) (const [iWord](#) &w)=0

Initiate a jump to a subroutine (alter the PC).

- virtual Codes::RESULT [_JSRR](#) (const [iWord](#) &baseR, const [iWord](#) &address)=0

Initiate a jump to a subroutine (alter the PC). param[in] baseR A register whose value acts as a base address.

- virtual Codes::RESULT [_Load](#) (const Decoder::REGISTER_ID DR, const [iWord](#) &address)=0

Loads a word in memory into a register.

- virtual Codes::RESULT [_LoadI](#) (const Decoder::REGISTER_ID DR, const [iWord](#) &address)=0

Performs an indirect load.

- virtual Codes::RESULT [_LoadR](#) (const Decoder::REGISTER_ID DR, Decoder::REGISTER_ID baseR, const [iWord](#) &address)=0

Performs a register-relative load.

- virtual Codes::RESULT [_Not](#) (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR)=0

Bit-wise nots a register and stores the result in another.

- virtual Codes::RESULT [_Ret](#) ()=0

Return from a subroutine.

- virtual Codes::RESULT [_Store](#) (const Decoder::REGISTER_ID SR1, const [iWord](#) &address)=0

Stores a register's value into memory at a specified address.

- virtual `Codes::RESULT _STI (const Decoder::REGISTER_ID SR1, const iWord &address)=0`
Performs an indirect store.
- virtual `Codes::RESULT _STR (const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID baseR, const iWord &address)=0`
Performs a register-relative store.
- virtual `Codes::RESULT _Trap (const iWord &code)=0`
Branches to a trap vector.

7.9.1 Detailed Description

Defines the internal logic of the Wi-11.

The methods present in this interface are meant to simulate the Wi-11's fetch-execute loop. Any implementation of this will be expected to house 8 private instances of the [Register](#) class as general purpose registers and each of these should have an associated REGISTER_ID enum token. A reference to an [iMemory](#) class is also necessary.

The implementers of a super class will also have to incorporate some sort of interaction with a CCR structure. An interface for this interaction is not provided.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 virtual iWi11::iWi11 () [pure virtual]

Creates and organizes the componts of the [Wi11](#) machine.
Initializes the general purpose registers, CCR, and memory.

7.9.3 Member Function Documentation

7.9.3.1 virtual iRegister& iWi11::_GetRegister (const Decoder::REGISTER_ID & id) [private, pure virtual]

Retrieves a reference to the register corresponding to "id".

Parameters

<code>in</code>	<code>id</code>	A REGISTER_ID corresponding to one of the private registers.
-----------------	-----------------	--

Returns

A reference to the id'd register.

Implemented in [Wi11](#).

```
7.9.3.2 virtual Codes::RESULT iWi11::_Add ( const Decoder::REGISTER_ID DR, const
      Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2 ) [private,
      pure virtual]
```

Adds two registers and stores the result in a third.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR1</i>	The first source register.
in	<i>SR2</i>	The second source register.

Postcondition

SR1 and SR2 are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implemented in [Wi11](#).

```
7.9.3.3 virtual Codes::RESULT iWi11::_Add ( const Decoder::REGISTER_ID DR, const
      Decoder::REGISTER_ID SR1, const iWord & immediate ) [private, pure
      virtual]
```

Adds a constant to a register and stores the result in another.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR1</i>	The source register.
in	<i>immediate</i>	The immediate value.

Postcondition

SR1 and "immediate" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implemented in [Wi11](#).

```
7.9.3.4 virtual Codes::RESULT iWi11::And ( const Decoder::REGISTER_ID DR, const
      Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2 ) [private,
      pure virtual]
```

Bit-wise ands two registers and stores the result in a third.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR1</i>	The first source register.
in	<i>SR2</i>	The second source register.

Postcondition

SR1 and SR2 are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implemented in [Wi11](#).

```
7.9.3.5 virtual Codes::RESULT iWi11::And ( const Decoder::REGISTER_ID DR, const
      Decoder::REGISTER_ID SR1, const iWord & immediate ) [private, pure
      virtual]
```

Bit-wise ands a register with a constant and stores the result in another register.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR1</i>	The source register.
in	<i>immediate</i>	The immediate value.

Postcondition

SR1 and "immediate" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implemented in [Wi11](#).

7.9.3.6 `virtual Codes::RESULT iWi11::_Branch (const iWord & address) [private, pure virtual]`

Changes the last 9 bits of the PC.

Parameters

<i>in</i>	<i>address</i>	The 9 bits to become the end of the PC.
-----------	----------------	---

Postcondition

"address" is not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Implemented in [Wi11](#).

7.9.3.7 `virtual Codes::RESULT iWi11::_Debug () [private, pure virtual]`

Deprecated?

Does nothing.

Implemented in [Wi11](#).

7.9.3.8 `virtual Codes::RESULT iWi11::_JSR (const iWord & w) [private, pure virtual]`

Initiate a jump to a subroutine (alter the PC).

Parameters

<i>in</i>	<i>w</i>	A 9 bit offset for the PC.
-----------	----------	----------------------------

Postcondition

The PC has "w" as its 9 least significant bits.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

If the link bit was set for this instruction, R7 will hold the old value of the PC. How-

ever, the CCR will not be altered for this instruction, despite R7 being altered.

Implemented in [Wi11](#).

7.9.3.9 virtual Codes::RESULT iWi11::_JSRR (const iWord & *baseR*, const iWord & *address*)
[private, pure virtual]

Initiate a jump to a subroutine (alter the PC). param[in] baseR A register whose value acts as a base address.

Parameters

in	<i>address</i>	A 6 bit offset to the base address.
----	----------------	-------------------------------------

Postcondition

The PC is the value in baseR plus the value in address.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

If the link bit was set for this instruction, R7 will hold the old value of the PC. However, the CCR will not be altered for this instruction, despite R7 being altered.

Implemented in [Wi11](#).

7.9.3.10 virtual Codes::RESULT iWi11::_Load (const Decoder::REGISTER_ID *DR*, const iWord & *address*) [private, pure virtual]

Loads a word in memory into a register.

Parameters

out	<i>DR</i>	The destination register.
in	<i>address</i>	When concatenated with the PC, forms address in memory from which to load.

Postcondition

[Memory](#) and "address" have not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implemented in [Wi11](#).

7.9.3.11 `virtual Codes::RESULT iWi11::LoadI (const Decoder::REGISTER_ID DR, const iWord & address) [private, pure virtual]`

Performs an indirect load.

Parameters

out	<i>DR</i>	The destination register.
in	<i>address</i>	A 9-bit offset to the PC.

Postcondition

[Memory](#) and "address" have not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Works similar to `_Load()` but when memory is read, it uses the address found to again access memory. In this indirect way, a load can be made from anywhere in [Memory](#).

Note

Updates the CCR.

Implemented in [Wi11](#).

7.9.3.12 `virtual Codes::RESULT iWi11::LoadR (const Decoder::REGISTER_ID DR, Decoder::REGISTER_ID baseR, const iWord & address) [private, pure virtual]`

Performs a register-relative load.

Parameters

out	<i>DR</i>	The destination register.
in	<i>baseR</i>	A register whose value works as a base address.
in	<i>address</i>	An 6-bit index from the base address.

Postcondition

[Memory](#), "baseR", and "address" have no changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Loads from "baseR" plus "address".

Note

Updates the CCR.

Implemented in [Wi11](#).

7.9.3.13 `virtual Codes::RESULT iWi11::Not (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR) [private, pure virtual]`

Bit-wise nots a register and stores the result in another.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR</i>	The source register.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implemented in [Wi11](#).

7.9.3.14 `virtual Codes::RESULT iWi11::Ret () [private, pure virtual]`

Return from a subroutine.

Postcondition

The PC now holds the value that was (and still is) in R7.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

This can be used to jump anywhere in memory. However, this is not the intended usage.

Updates the CCR.

Implemented in [Wi11](#).

7.9.3.15 `virtual Codes::RESULT iWi11::_Store (const Decoder::REGISTER_ID SR1, const iWord & address) [private, pure virtual]`

Stores a register's value into memory at a specified address.

Parameters

in	<i>SR1</i>	The source register (holds the data to be stored).
in	<i>address</i>	When concatenated with the PC, forms the address for the store.

Postcondition

SR1 and "address" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Implemented in [Wi11](#).

7.9.3.16 `virtual Codes::RESULT iWi11::_STI (const Decoder::REGISTER_ID SR1, const iWord & address) [private, pure virtual]`

Performs an indirect store.

Parameters

in	<i>SR1</i>	The source register (holds the data to be stored).
in	<i>address</i>	A 9-bit offset to the PC.

Postcondition

"SR1" and "address" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Works similar to `_Store()` but when memory is read, it uses the address found to again access memory. In this indirect way, a store can be made to anywhere in [Memory](#).

Implemented in [Wi11](#).

7.9.3.17 `virtual Codes::RESULT iWi11::_STR (const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID baseR, const iWord & address) [private, pure virtual]`

Performs a register-relative store.

Parameters

in	<i>SR1</i>	The source register (holds the data to be stored).
----	------------	--

in	<i>baseR</i>	A register whose value acts as a base address.
in	<i>address</i>	A 6-bit index from the base address.

Postcondition

SR1, baseR, and "address" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Implemented in [Wi11](#).

7.9.3.18 `virtual Codes::RESULT iWi11::Trap (const iWord & code)` [private, pure virtual]

Branches to a trap vector.

Parameters

in	<i>code</i>	The trap code.
----	-------------	----------------

Postcondition

"code" is not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

The traps are as follows:

- 0x21 - OUT - Write the character formed from the eight least significant bits of R0 to standard out.
- 0x22 - PUTS - Write the a string to standard out starting at the address pointed to by R0 and ending at a null character.
- 0x23 - IN - Prompt for, and read, a single character from standard in. Re-print it and store its ascii value in R0 (with leading zeros).
- 0x25 - HALT - End execution and print an appropriate message to standard out.
- 0x31 - INN - Prompt for, and read, a positive decimal number from standard in. Re-print it and store it in R0 (the number must in 16-bit range).
- 0x43 - RND - Generate a random number and store it in R0.

Note

Traps 0x23, 0x31, and 0x43 all update the CCR.

Standard in is the keyboard.
Standard out is the console.

Implemented in [Wi11](#).

7.9.3.19 virtual bool iWi11::LoadObj (const char * *filename*) [pure virtual]

Loads the object file and sets up memory as it describes.

Parameters

in	<i>filename</i>	The name of the object file.
----	-----------------	------------------------------

Postcondition

"filename" is not changed.

Returns

True if and only if the load was successful.

If "false" is returned, prints an appropriate error message to the user.

Note

This function can be called multiple times. Each time the PC is overwritten.

Implemented in [Wi11](#).

7.9.3.20 virtual void iWi11::DisplayMemory () const [pure virtual]

Prints the state of memory to standard out.

Postcondition

The calling object is not changed.

Implemented in [Wi11](#).

7.9.3.21 virtual void iWi11::DisplayRegisters () const [pure virtual]

Prints the state of every register to standard out.

Postcondition

The calling object is not changed.

The values of all 8 general purpose registers, the CCR, and PC are all printed.

Implemented in [Wi11](#).

7.9.3.22 `virtual bool iWi11::ExecuteNext (bool verbose = false)` [pure virtual]

Executes the instruction pointed to by the PC.

Parameters

<code>in</code>	<code>verbose</code>	If true, machine state information is displayed after each step.
-----------------	----------------------	--

Returns

True if and only if the end of the program have been reached.

This function is the brains of the operation, so to speak. Almost the entire fetch-execute loop of the Wi-11 is present here. In particular, this function must interpret the instructions and manage the CCRs.

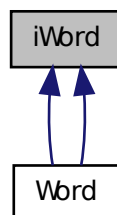
For a complete list of the instructions, see [Wi-11 Instructions](#).

Implemented in [Wi11](#).

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: "0xFFFF"

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

<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](#).

7.10.2.8 virtual Word iWord::Add (const iWord & w) const [pure virtual]

Adds two words.

Parameters

<code>in</code>	<code>w</code>	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

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](#).

7.10.2.14 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

<code>in</code>	<code>w</code>	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 <=> 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.

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

<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.

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

<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](#).

7.10.2.30 `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`.

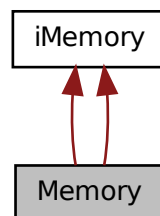
If it holds a value of 1 (0...001 in binary): `num[0] = 1`.

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

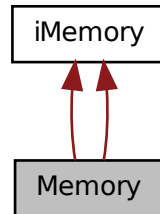
Implemented in [Word](#), and [Word](#).

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`

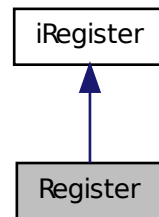
7.12 ObjectData Struct Reference

Public Attributes

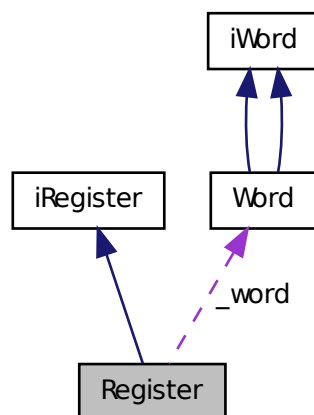
- char `type`
- std::vector< std::string > `data`

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

<i>in</i>	<i>r</i>	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

<i>in</i>	<i>w</i>	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

<i>in</i>	<i>r</i>	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	r	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	<i>r</i>	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

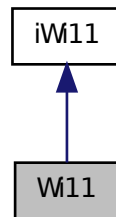
- std::string Find (const Codes::RESULT &) const
- std::string Find (const Codes::RESULT &) const

Static Private Attributes

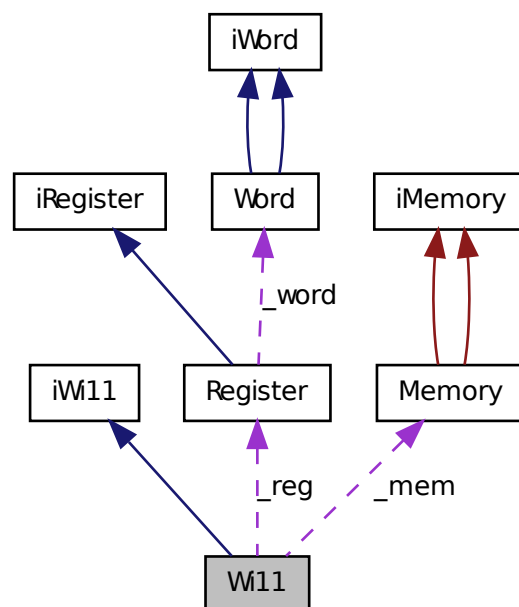
- static std::map< Codes::RESULT, std::string > _codes

7.15 Wi11 Class Reference

Inheritance diagram for Wi11:



Collaboration diagram for Wi11:



Public Member Functions

- bool [LoadObj](#) (const char *filename)
Loads the object file and sets up memory as it describes.
- void [DisplayMemory](#) () const
Prints the state of memory to standard out.
- void [DisplayRegisters](#) () const
Prints the state of every register to standard out.
- bool [ExecuteNext](#) (bool verbose=false)
Executes the instruction pointed to by the PC.

Private Member Functions

- [iRegister](#) & [_GetRegister](#) (const Decoder::REGISTER_ID &id)
Retrieves a reference to the register corresponding to "id".
- Codes::RESULT [_Add](#) (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2)
Adds two registers and stores the result in a third.
- Codes::RESULT [_Add](#) (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const [iWord](#) &immediate)
Adds a constant to a register and stores the result in another.
- Codes::RESULT [_And](#) (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2)
Bit-wise ands two registers and stores the result in a third.
- Codes::RESULT [_And](#) (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const [iWord](#) &immediate)
Bit-wise ands a register with a constant and stores the result in another register.
- Codes::RESULT [_Branch](#) (const [iWord](#) &address)
Changes the last 9 bits of the PC.
- Codes::RESULT [_Debug](#) ()
Deprecated?
- Codes::RESULT [_JSR](#) (const [iWord](#) &w)
Initiate a jump to a subroutine (alter the PC).
- Codes::RESULT [_JSRR](#) (const [iWord](#) &baseR, const [iWord](#) &address)

Initiate a jump to a subroutine (alter the PC). param[in] baseR A register whose value acts as a base address.

- Codes::RESULT [_Load](#) (const Decoder::REGISTER_ID DR, const [iWord](#) &address)
Loads a word in memory into a register.
- Codes::RESULT [_LoadI](#) (const Decoder::REGISTER_ID DR, const [iWord](#) &address)
Performs an indirect load.
- Codes::RESULT [_LoadR](#) (const Decoder::REGISTER_ID DR, Decoder::REGISTER_ID baseR, const [iWord](#) &address)
Performs a register-relative load.
- Codes::RESULT [_Not](#) (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR)
Bit-wise nots a register and stores the result in another.
- Codes::RESULT [_Ret](#) ()
Return from a subroutine.
- Codes::RESULT [_Store](#) (const Decoder::REGISTER_ID SR1, const [iWord](#) &address)
Stores a register's value into memory at a specified address.
- Codes::RESULT [_STI](#) (const Decoder::REGISTER_ID SR1, const [iWord](#) &address)
Performs an indirect store.
- Codes::RESULT [_STR](#) (const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID baseR, const [iWord](#) &address)
Performs a register-relative store.
- Codes::RESULT [_Trap](#) (const [iWord](#) &code)
Branches to a trap vector.

Private Attributes

- [Memory _mem](#)
Wi-11's memory.
- [Register _reg](#) [8]
8 general purpose registers.

- bool [_pos](#)
CCR, true iff positive.
- bool [_zero](#)
CCR, true iff zero.

7.15.1 Member Function Documentation

7.15.1.1 `iRegister& Wi11::_GetRegister (const Decoder::REGISTER_ID & id)` [`private`, `virtual`]

Retrieves a reference to the register corresponding to "id".

Parameters

<code>in</code>	<code>id</code>	A REGISTER_ID corresponding to one of the private registers.
-----------------	-----------------	--

Returns

A reference to the id'd register.

Implements [iWi11](#).

7.15.1.2 `Codes::RESULT Wi11::Add (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2)` [`private`, `virtual`]

Adds two registers and stores the result in a third.

Parameters

<code>out</code>	<code>DR</code>	The destination register.
<code>in</code>	<code>SR1</code>	The first source register.
<code>in</code>	<code>SR2</code>	The second source register.

Postcondition

SR1 and SR2 are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implements [iWi11](#).

7.15.1.3 `Codes::RESULT Wi11::Add (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const iWord & immediate) [private, virtual]`

Adds a constant to a register and stores the result in another.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR1</i>	The source register.
in	<i>immediate</i>	The immediate value.

Postcondition

SR1 and "immediate" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implements [iWi11](#).

7.15.1.4 `Codes::RESULT Wi11::And (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID SR2) [private, virtual]`

Bit-wise ands two registers and stores the result in a third.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR1</i>	The first source register.
in	<i>SR2</i>	The second source register.

Postcondition

SR1 and *SR2* are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implements [iWi11](#).

7.15.1.5 Codes::RESULT Wi11::And (const Decoder::REGISTER_ID *DR*, const Decoder::REGISTER_ID *SR1*, const iWord & *immediate*) [private, virtual]

Bit-wise ands a register with a constant and stores the result in another register.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR1</i>	The source register.
in	<i>immediate</i>	The immediate value.

Postcondition

SR1 and "immediate" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implements [iWi11](#).

7.15.1.6 Codes::RESULT Wi11::Branch (const iWord & *address*) [private, virtual]

Changes the last 9 bits of the PC.

Parameters

in	<i>address</i>	The 9 bits to become the end of the PC.
----	----------------	---

Postcondition

"address" is not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Implements [iWi11](#).

7.15.1.7 Codes::RESULT Wi11::Debug () [private, virtual]

Deprecated?

Does nothing.

Implements [iWi11](#).

7.15.1.8 Codes::RESULT Wi11::JSR (const iWord & *w*) [private, virtual]

Initiate a jump to a subroutine (alter the PC).

Parameters

<i>in</i>	<i>w</i>	A 9 bit offset for the PC.
-----------	----------	----------------------------

Postcondition

The PC has "*w*" as its 9 least significant bits.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

If the link bit was set for this instruction, R7 will hold the old value of the PC. However, the CCR will not be altered for this instruction, despite R7 being altered.

Implements [iWi11](#).

7.15.1.9 Codes::RESULT Wi11::JSRR (const iWord & *baseR*, const iWord & *address*) [private, virtual]

Initiate a jump to a subroutine (alter the PC). param[*in*] *baseR* A register whose value acts as a base address.

Parameters

<i>in</i>	<i>address</i>	A 6 bit offset to the base address.
-----------	----------------	-------------------------------------

Postcondition

The PC is the value in *baseR* plus the value in *address*.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

If the link bit was set for this instruction, R7 will hold the old value of the PC. However, the CCR will not be altered for this instruction, despite R7 being altered.

Implements [iWi11](#).

7.15.1.10 Codes::RESULT Wi11::Load (const Decoder::REGISTER_ID *DR*, const iWord & *address*) [private, virtual]

Loads a word in memory into a register.

Parameters

out	<i>DR</i>	The destination register.
in	<i>address</i>	When concatenated with the PC, forms address in memory from which to load.

Postcondition

[Memory](#) and "address" have not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implements [iWi11](#).

7.15.1.11 `Codes::RESULT Wi11::_LoadI (const Decoder::REGISTER_ID DR, const iWord & address) [private, virtual]`

Performs an indirect load.

Parameters

out	<i>DR</i>	The destination register.
in	<i>address</i>	A 9-bit offset to the PC.

Postcondition

[Memory](#) and "address" have not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Works similar to `_Load()` but when memory is read, it uses the address found to again access memory. In this indirect way, a load can be made from anywhere in [Memory](#).

Note

Updates the CCR.

Implements [iWi11](#).

7.15.1.12 `Codes::RESULT Wi11::_LoadR (const Decoder::REGISTER_ID DR, Decoder::REGISTER_ID baseR, const iWord & address) [private, virtual]`

Performs a register-relative load.

Parameters

out	<i>DR</i>	The destination register.
in	<i>baseR</i>	A register whose value works as a base address.
in	<i>address</i>	An 6-bit index from the base address.

Postcondition

[Memory](#), "baseR", and "address" have no changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Loads from "baseR" plus "address".

Note

Updates the CCR.

Implements [iWi11](#).

7.15.1.13 `Codes::RESULT Wi11::Not (const Decoder::REGISTER_ID DR, const Decoder::REGISTER_ID SR) [private, virtual]`

Bit-wise nots a register and stores the result in another.

Parameters

out	<i>DR</i>	The destination register.
in	<i>SR</i>	The source register.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

Updates the CCR.

Implements [iWi11](#).

7.15.1.14 `Codes::RESULT Wi11::Ret () [private, virtual]`

Return from a subroutine.

Postcondition

The PC now holds the value that was (and still is) in R7.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Note

This can be used to jump anywhere in memory. However, this is not the intended usage.

Updates the CCR.

Implements [iWi11](#).

7.15.1.15 `Codes::RESULT Wi11::_Store (const Decoder::REGISTER_ID SR1, const iWord & address) [private, virtual]`

Stores a register's value into memory at a specified address.

Parameters

in	<i>SR1</i>	The source register (holds the data to be stored).
in	<i>address</i>	When concatenated with the PC, forms the address for the store.

Postcondition

SR1 and "address" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Implements [iWi11](#).

7.15.1.16 `Codes::RESULT Wi11::_STI (const Decoder::REGISTER_ID SR1, const iWord & address) [private, virtual]`

Performs an indirect store.

Parameters

in	<i>SR1</i>	The source register (holds the data to be stored).
in	<i>address</i>	A 9-bit offset to the PC.

Postcondition

"SR1" and "address" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Works similar to `_Store()` but when memory is read, it uses the address found to again access memory. In this indirect way, a store can be made to anywhere in [Memory](#).

Implements [iWi11](#).

7.15.1.17 `Codes::RESULT Wi11::STR (const Decoder::REGISTER_ID SR1, const Decoder::REGISTER_ID baseR, const iWord & address) [private, virtual]`

Performs a register-relative store.

Parameters

in	<i>SR1</i>	The source register (holds the data to be stored).
in	<i>baseR</i>	A register whose value acts as a base address.
in	<i>address</i>	A 6-bit index from the base address.

Postcondition

SR1, *baseR*, and "address" are not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

Implements [iWi11](#).

7.15.1.18 `Codes::RESULT Wi11::Trap (const iWord & code) [private, virtual]`

Branches to a trap vector.

Parameters

in	<i>code</i>	The trap code.
----	-------------	----------------

Postcondition

"code" is not changed.

Returns

SUCCESS or, if something went wrong, an appropriate error code.

The traps are as follows:

- 0x21 - OUT - Write the character formed from the eight least significant bits of R0 to standard out.
- 0x22 - PUTS - Write the a string to standard out starting at the address pointed to by R0 and ending at a null character.
- 0x23 - IN - Prompt for, and read, a single character from standard in. Re-print it and store its ascii value in R0 (with leading zeros).

- 0x25 - HALT - End execution and print an appropriate message to standard out.
- 0x31 - INN - Prompt for, and read, a positive decimal number from standard in. Re-print it and store it in R0 (the number must in 16-bit range).
- 0x43 - RND - Generate a random number and store it in R0.

Note

Traps 0x23, 0x31, and 0x43 all update the CCR.

Standard in is the keyboard.
Standard out is the console.

Implements [iWi11](#).

7.15.1.19 bool Wi11::LoadObj (const char * filename) [virtual]

Loads the object file and sets up memory as it describes.

Parameters

<i>in</i>	<i>filename</i>	The name of the object file.
-----------	-----------------	------------------------------

Postcondition

"filename" is not changed.

Returns

True if and only if the load was successful.

If "false" is returned, prints an appropriate error message to the user.

Note

This function can be called multiple times. Each time the PC is overwritten.

Implements [iWi11](#).

7.15.1.20 void Wi11::DisplayMemory () const [virtual]

Prints the state of memory to standard out.

Postcondition

The calling object is not changed.

Implements [iWi11](#).

7.15.1.21 void Wi11::DisplayRegisters () const [virtual]

Prints the state of every register to standard out.

Postcondition

The calling object is not changed.

The values of all 8 general purpose registers, the CCR, and PC are all printed.

Implements [iWi11](#).

7.15.1.22 bool Wi11::ExecuteNext (bool *verbose* = false) [virtual]

Executes the instruction pointed to by the PC.

Parameters

<i>in</i>	<i>verbose</i>	If true, machine state information is displayed after each step.
-----------	----------------	--

Returns

True if and only if the end of the program have been reached.

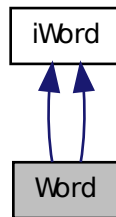
This function is the brains of the operation, so to speak. Almost the entire fetch-execute loop of the Wi-11 is present here. In particular, this function must interpret the instructions and manage the CCRs.

For a complete list of the instructions, see [Wi-11 Instructions](#).

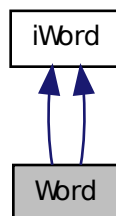
Implements [iWi11](#).

7.16 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.

7.16.1 Member Function Documentation

7.16.1.1 `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[]()`.

7.16.1.2 `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](#).

7.16.1.3 `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](#).

7.16.1.4 `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](#).

7.16.1.5 `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: "0xFFFE"

Implements [iWord](#).

7.16.1.6 `bool Word::fromInt (int value)` `[virtual]`

"From Integer"

Parameters

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

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

"From String"

Parameters

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

7.16.1.8 bool Word::fromHex (const std::string & str) [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](#).

7.16.1.9 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](#).

7.16.1.10 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](#).

7.16.1.11 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](#).

7.16.1.12 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](#).

7.16.1.13 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](#).

7.16.1.14 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](#).

7.16.1.15 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](#).

7.16.1.16 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](#).

7.16.1.17 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](#).

7.16.1.18 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](#).

7.16.1.19 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](#).

7.16.1.20 `bool Word::operator[] (const int i) const` `[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`.

Implements [iWord](#).

7.16.1.21 `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\[\]\(\)](#).

7.16.1.22 `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](#).

7.16.1.23 `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](#).

7.16.1.24 `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](#).

7.16.1.25 `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: "0xFFFF"

Implements [iWord](#).

7.16.1.26 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](#).

7.16.1.27 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](#).

7.16.1.28 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](#).

7.16.1.29 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](#).

7.16.1.30 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](#).

7.16.1.31 Word Word::Subtract (const iWord & w) const [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.

Implements [iWord](#).

7.16.1.32 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](#).

7.16.1.33 Word Word::And (const iWord & w) const [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.

Implements [iWord](#).

7.16.1.34 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](#).

7.16.1.35 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](#).

7.16.1.36 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](#).

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

A standard assignment operator.

Parameters

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

7.16.1.38 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](#).

7.16.1.39 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](#).

7.16.1.40 bool Word::operator[] (const int i) const [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.

Implements [iWord](#).

7.16.2 Member Data Documentation

7.16.2.1 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.

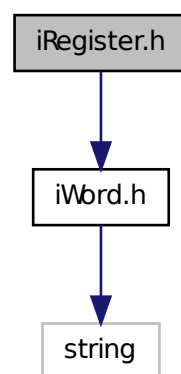
Chapter 8

File Documentation

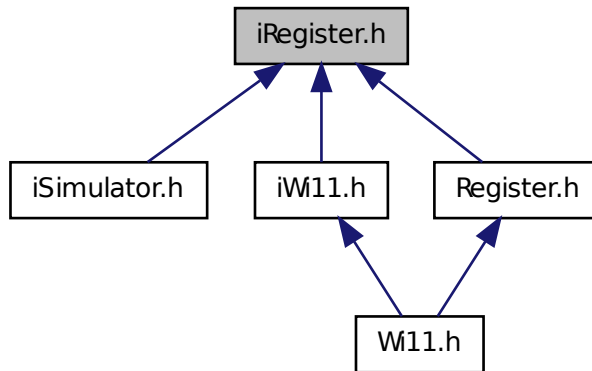
8.1 iRegister.h File Reference

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.

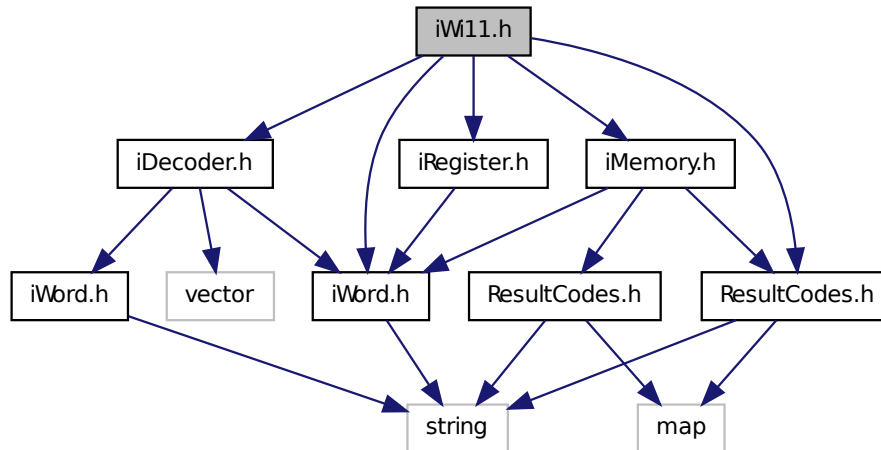
Author

Joshua Green
Andrew Groot

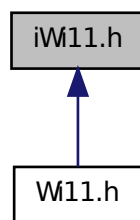
8.2 iWi11.h File Reference

Definition of the Wi-11 machine simulator.

Include dependency graph for iWi11.h:



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



Classes

- class [iWi11](#)

Defines the internal logic of the Wi-11.

8.2.1 Detailed Description

Definition of the Wi-11 machine simulator.

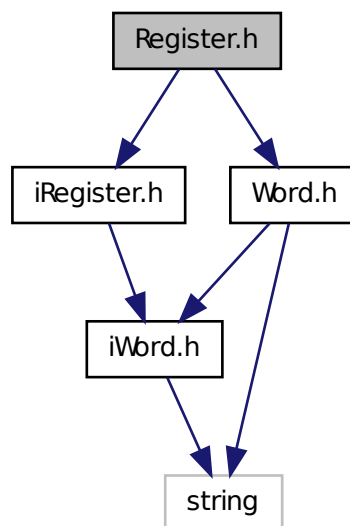
Author

Joshua Green
Andrew Groot

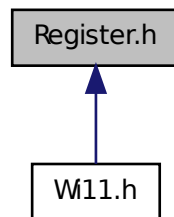
8.3 Register.h File Reference

Definition of private data for the "Register" class.

Include dependency graph for Register.h:



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



Classes

- class [Register](#)

8.3.1 Detailed Description

Definition of private data for the "Register" class.

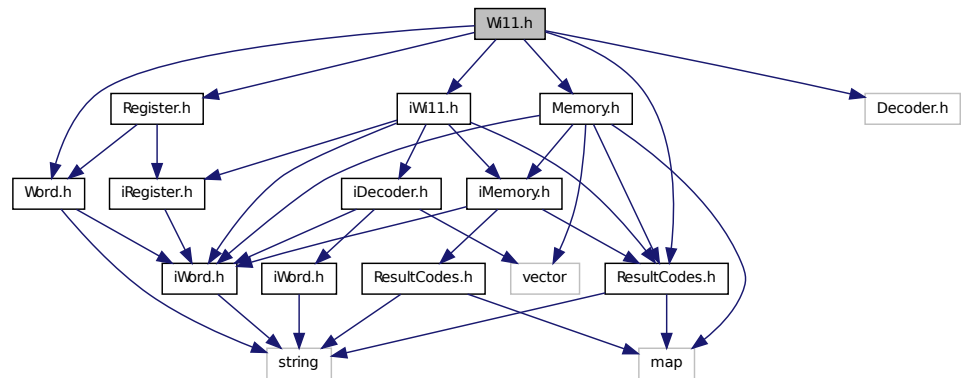
Author

Andrew Groot

8.4 Wi11.h File Reference

Definition of the private data for the "Wi11" class.

Include dependency graph for Wi11.h:



Classes

- class [Wi11](#)

8.4.1 Detailed Description

Definition of the private data for the "Wi11" class.

Author

Andrew Groot

Index

- `_Add`
 - `iWi11`, [30](#)
 - `Wi11`, [68](#)
 - `_And`
 - `iWi11`, [31](#)
 - `Wi11`, [69](#)
 - `_Branch`
 - `iWi11`, [32](#)
 - `Wi11`, [70](#)
 - `_Debug`
 - `iWi11`, [32](#)
 - `Wi11`, [70](#)
 - `_GetRegister`
 - `iWi11`, [29](#)
 - `Wi11`, [68](#)
 - `_HasBit`
 - `Word`, [88](#)
 - `_JSR`
 - `iWi11`, [32](#)
 - `Wi11`, [70](#)
 - `_JSRR`
 - `iWi11`, [33](#)
 - `Wi11`, [71](#)
 - `_Load`
 - `iWi11`, [33](#)
 - `Wi11`, [71](#)
 - `_Loadl`
 - `iWi11`, [34](#)
 - `Wi11`, [72](#)
 - `_LoadR`
 - `iWi11`, [34](#)
 - `Wi11`, [72](#)
 - `_Not`
 - `iWi11`, [35](#)
 - `Wi11`, [73](#)
 - `_Ret`
 - `iWi11`, [35](#)
 - `Wi11`, [73](#)
 - `_STI`
 - `iWi11`, [36](#)
 - `Wi11`, [74](#)

- `_STR`
 - `iWi11`, [36](#)
 - `Wi11`, [75](#)
 - `_Store`
 - `iWi11`, [35](#)
 - `Wi11`, [74](#)
 - `_Trap`
 - `iWi11`, [37](#)
 - `Wi11`, [75](#)
 - `_hasBit`
 - `Word`, [81](#)
 - `_value`
 - `Word`, [95](#)
 - `Add`
 - `iRegister`, [21](#)
 - `iWord`, [45](#), [51](#)
 - `Register`, [59](#)
 - `Word`, [84](#), [91](#)
 - `And`
 - `iRegister`, [22](#), [23](#)
 - `iWord`, [46](#), [52](#)
 - `Register`, [61](#)
 - `Word`, [85](#), [92](#)

- `code/ Directory Reference`, [13](#)
 - `code/MemoryTest/ Directory Reference`, [14](#)
 - `code/test/ Directory Reference`, [15](#)

- `Copy`
 - `iWord`, [47](#)
 - `Word`, [93](#)
 - `copy`
 - `iWord`, [53](#)
 - `Word`, [86](#)

- `DisplayMemory`
 - `iWi11`, [38](#)
 - `Wi11`, [76](#)
 - `DisplayRegisters`
 - `iWi11`, [38](#)
 - `Wi11`, [76](#)

- ExecuteNext
 - iWi11, [38](#)
 - Wi11, [77](#)
- FromHex
 - iWord, [44](#)
 - Word, [90](#)
- fromHex
 - iWord, [51](#)
 - Word, [84](#)
- FromInt
 - iWord, [43](#)
 - Word, [89](#)
- fromInt
 - iWord, [50](#)
 - Word, [83](#)
- FromStr
 - iWord, [44](#)
 - Word, [90](#)
- fromStr
 - iWord, [50](#)
 - Word, [83](#)
- GetValue
 - iRegister, [20](#)
 - Register, [59](#)
- iDecoder, [17](#)
- iInterpreter, [17](#)
- iLoader, [17](#)
- iMemory, [18](#)
- Instruction, [18](#)
- iObjParser, [18](#)
- iRegister, [19](#)
 - Add, [21](#)
 - And, [22, 23](#)
 - GetValue, [20](#)
 - Not, [24](#)
 - operator+, [21](#)
 - operator++, [25](#)
 - operator-, [22](#)
 - operator=, [25](#)
 - Or, [23](#)
 - Store, [24](#)
 - Subtract, [22](#)
- iRegister.h, [97](#)
- iSimulator, [26](#)
- iWi11, [26](#)
 - _Add, [30](#)
 - _And, [31](#)
 - _Branch, [32](#)
 - _Debug, [32](#)
 - _GetRegister, [29](#)
 - _JSR, [32](#)
 - _JSRR, [33](#)
 - _Load, [33](#)
 - _LoadI, [34](#)
 - _LoadR, [34](#)
 - _Not, [35](#)
 - _Ret, [35](#)
 - _STI, [36](#)
 - _STR, [36](#)
 - _Store, [35](#)
 - _Trap, [37](#)
 - DisplayMemory, [38](#)
 - DisplayRegisters, [38](#)
 - ExecuteNext, [38](#)
 - iWi11, [29](#)
 - LoadObj, [38](#)
- iWi11.h, [98](#)
- iWord, [39](#)
 - Add, [45, 51](#)
 - And, [46, 52](#)
 - Copy, [47](#)
 - copy, [53](#)
 - FromHex, [44](#)
 - fromHex, [51](#)
 - FromInt, [43](#)
 - fromInt, [50](#)
 - FromStr, [44](#)
 - fromStr, [50](#)
 - Not, [47, 53](#)
 - operator+, [45, 52](#)
 - operator++, [48, 54](#)
 - operator-, [46, 52](#)
 - operator=, [47, 54](#)
 - Or, [46, 53](#)
 - Subtract, [45, 52](#)
 - ToHex, [43](#)
 - toHex, [50](#)
 - ToInt, [42](#)
 - toInt, [49](#)
 - ToInt2Complement, [42](#)
 - toInt2Complement, [49](#)
 - ToStr, [43](#)
 - toStr, [49](#)
- LoadObj
 - iWi11, [38](#)
 - Wi11, [76](#)

- Memory, [55](#)
- Not
 - iRegister, [24](#)
 - iWord, [47](#), [53](#)
 - Register, [62](#)
 - Word, [86](#), [93](#)
- ObjectData, [56](#)
- operator+
 - iRegister, [21](#)
 - iWord, [45](#), [52](#)
 - Register, [60](#)
 - Word, [84](#), [91](#)
- operator++
 - iRegister, [25](#)
 - iWord, [48](#), [54](#)
 - Register, [64](#)
 - Word, [87](#), [94](#)
- operator-
 - iRegister, [22](#)
 - iWord, [46](#), [52](#)
 - Register, [61](#)
 - Word, [85](#), [92](#)
- operator=
 - iRegister, [25](#)
 - iWord, [47](#), [54](#)
 - Register, [63](#)
 - Word, [87](#), [93](#)
- Or
 - iRegister, [23](#)
 - iWord, [46](#), [53](#)
 - Register, [61](#), [62](#)
 - Word, [86](#), [92](#)
- Register, [57](#)
 - Add, [59](#)
 - And, [61](#)
 - GetValue, [59](#)
 - Not, [62](#)
 - operator+, [60](#)
 - operator++, [64](#)
 - operator-, [61](#)
 - operator=, [63](#)
 - Or, [61](#), [62](#)
 - Store, [63](#)
 - Subtract, [60](#)
- Register.h, [100](#)
- ResultDecoder, [64](#)
- Store
 - iRegister, [24](#)
 - Register, [63](#)
- Subtract
 - iRegister, [22](#)
 - iWord, [45](#), [52](#)
 - Register, [60](#)
 - Word, [85](#), [91](#)
- ToHex
 - iWord, [43](#)
 - Word, [89](#)
- toHex
 - iWord, [50](#)
 - Word, [82](#)
- ToInt
 - iWord, [42](#)
 - Word, [88](#)
- toInt
 - iWord, [49](#)
 - Word, [81](#)
- ToInt2Complement
 - iWord, [42](#)
 - Word, [88](#)
- toInt2Complement
 - iWord, [49](#)
 - Word, [82](#)
- ToStr
 - iWord, [43](#)
 - Word, [89](#)
- toStr
 - iWord, [49](#)
 - Word, [82](#)
- Wi11, [65](#)
 - _Add, [68](#)
 - _And, [69](#)
 - _Branch, [70](#)
 - _Debug, [70](#)
 - _GetRegister, [68](#)
 - _JSR, [70](#)
 - _JSRR, [71](#)
 - _Load, [71](#)
 - _LoadI, [72](#)
 - _LoadR, [72](#)
 - _Not, [73](#)
 - _Ret, [73](#)
 - _STI, [74](#)
 - _STR, [75](#)
 - _Store, [74](#)
 - _Trap, [75](#)

- DisplayMemory, [76](#)
- DisplayRegisters, [76](#)
- ExecuteNext, [77](#)
- LoadObj, [76](#)
- Wi11.h, [101](#)
- Word, [78](#)
 - _HasBit, [88](#)
 - _hasBit, [81](#)
 - _value, [95](#)
 - Add, [84](#), [91](#)
 - And, [85](#), [92](#)
 - Copy, [93](#)
 - copy, [86](#)
 - FromHex, [90](#)
 - fromHex, [84](#)
 - FromInt, [89](#)
 - fromInt, [83](#)
 - FromStr, [90](#)
 - fromStr, [83](#)
 - Not, [86](#), [93](#)
 - operator+, [84](#), [91](#)
 - operator++, [87](#), [94](#)
 - operator-, [85](#), [92](#)
 - operator=, [87](#), [93](#)
 - Or, [86](#), [92](#)
 - Subtract, [85](#), [91](#)
 - ToHex, [89](#)
 - toHex, [82](#)
 - ToInt, [88](#)
 - toInt, [81](#)
 - ToInt2Complement, [88](#)
 - toInt2Complement, [82](#)
 - ToStr, [89](#)
 - toStr, [82](#)