

# OneUp Wi-11 Simulator - Programmer's Guide

Generated by Doxygen 1.7.2

Sun Jan 23 2011 18:47:48



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
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	Interaction . . . . .	3
1.3.1	Components . . . . .	3
1.3.1.1	Loading . . . . .	3
1.3.1.2	Executing . . . . .	3
1.3.2	Wi-11 Instruction Set . . . . .	4
<b>2</b>	<b>Namespace Index</b>	<b>5</b>
2.1	Namespace List . . . . .	5
<b>3</b>	<b>Class Index</b>	<b>7</b>
3.1	Class Hierarchy . . . . .	7
<b>4</b>	<b>Class Index</b>	<b>9</b>
4.1	Class List . . . . .	9
<b>5</b>	<b>File Index</b>	<b>11</b>
5.1	File List . . . . .	11
<b>6</b>	<b>Namespace Documentation</b>	<b>13</b>
6.1	Codes Namespace Reference . . . . .	13
6.1.1	Detailed Description . . . . .	13
6.2	Decoder_Directory Namespace Reference . . . . .	13
6.2.1	Detailed Description . . . . .	14
<b>7</b>	<b>Class Documentation</b>	<b>15</b>
7.1	Decoder Class Reference . . . . .	15
7.2	iDecoder Class Reference . . . . .	15
7.2.1	Detailed Description . . . . .	15
7.2.2	Member Function Documentation . . . . .	16
7.2.2.1	DecodeInstruction . . . . .	16

7.3	iLoader Class Reference	16
7.3.1	Detailed Description	16
7.3.2	Member Function Documentation	16
7.3.2.1	Load	16
7.4	iMemory Class Reference	17
7.4.1	Detailed Description	17
7.4.2	Member Function Documentation	17
7.4.2.1	Reserve	17
7.4.2.2	Load	18
7.4.2.3	Store	18
7.5	Instruction Struct Reference	18
7.5.1	Detailed Description	19
7.5.2	Member Data Documentation	19
7.5.2.1	data	19
7.6	iObjParser Class Reference	20
7.6.1	Detailed Description	20
7.6.2	Member Function Documentation	20
7.6.2.1	Initialize	20
7.6.2.2	GetNext	21
7.7	iRegister Class Reference	21
7.7.1	Detailed Description	23
7.7.2	Member Function Documentation	23
7.7.2.1	GetValue	23
7.7.2.2	Add	23
7.7.2.3	Add	23
7.7.2.4	operator+	24
7.7.2.5	Subtract	24
7.7.2.6	Subtract	24
7.7.2.7	operator-	25
7.7.2.8	And	25
7.7.2.9	And	25
7.7.2.10	Or	26
7.7.2.11	Or	26
7.7.2.12	Not	26
7.7.2.13	Not	26
7.7.2.14	Store	27
7.7.2.15	Store	27
7.7.2.16	operator=	27
7.7.2.17	operator=	27
7.7.2.18	operator++	27
7.7.2.19	operator++	28
7.8	iWi11 Class Reference	28
7.8.1	Detailed Description	30
7.8.2	Member Function Documentation	31
7.8.2.1	_GetRegister	31
7.8.2.2	_Add	31

7.8.2.3	_Add	31
7.8.2.4	_And	32
7.8.2.5	_And	32
7.8.2.6	_Branch	33
7.8.2.7	_Debug	33
7.8.2.8	_JSR	33
7.8.2.9	_JSRR	34
7.8.2.10	_Load	34
7.8.2.11	_LoadI	35
7.8.2.12	_LoadR	35
7.8.2.13	_Not	36
7.8.2.14	_Ret	36
7.8.2.15	_Store	37
7.8.2.16	_STI	37
7.8.2.17	_STR	38
7.8.2.18	_Trap	38
7.8.2.19	LoadObj	39
7.8.2.20	DisplayMemory	40
7.8.2.21	DisplayRegisters	40
7.8.2.22	ExecuteNext	40
7.9	iWord Class Reference	40
7.9.1	Detailed Description	42
7.9.2	Member Function Documentation	42
7.9.2.1	ToInt	42
7.9.2.2	ToInt2Complement	43
7.9.2.3	ToStr	43
7.9.2.4	ToHex	43
7.9.2.5	FromInt	44
7.9.2.6	FromStr	44
7.9.2.7	FromHex	44
7.9.2.8	Add	45
7.9.2.9	operator+	45
7.9.2.10	Subtract	45
7.9.2.11	operator-	46
7.9.2.12	And	46
7.9.2.13	Or	46
7.9.2.14	Not	47
7.9.2.15	Copy	47
7.9.2.16	operator=	47
7.9.2.17	operator++	48
7.9.2.18	operator++	48
7.9.2.19	operator[]	48
7.9.2.20	SetBit	49
7.10	Loader Class Reference	49
7.10.1	Detailed Description	50
7.10.2	Constructor & Destructor Documentation	50

7.10.2.1	Loader	50
7.11	Memory Class Reference	50
7.11.1	Detailed Description	51
7.12	ObjectData Struct Reference	52
7.12.1	Detailed Description	52
7.13	ObjParser Class Reference	52
7.13.1	Detailed Description	53
7.13.2	Member Function Documentation	54
7.13.2.1	Initialize	54
7.13.2.2	GetNext	54
7.14	Register Class Reference	54
7.14.1	Detailed Description	56
7.15	ResultDecoder Class Reference	56
7.15.1	Detailed Description	56
7.15.2	Member Function Documentation	57
7.15.2.1	Find	57
7.15.3	Member Data Documentation	57
7.15.3.1	_codes	57
7.16	Wi11 Class Reference	57
7.16.1	Detailed Description	60
7.16.2	Constructor & Destructor Documentation	60
7.16.2.1	Wi11	60
7.17	Wi11::CCR Struct Reference	60
7.17.1	Detailed Description	61
7.18	Word Class Reference	61
7.18.1	Detailed Description	62
7.18.2	Member Function Documentation	62
7.18.2.1	_HasBit	62
7.18.3	Member Data Documentation	63
7.18.3.1	_value	63
<b>8</b>	<b>File Documentation</b>	<b>65</b>
8.1	iDecoder.h File Reference	65
8.1.1	Detailed Description	66
8.2	iLoader.h File Reference	67
8.2.1	Detailed Description	67
8.3	iMemory.h File Reference	68
8.3.1	Detailed Description	68
8.4	iObjParser.h File Reference	69
8.4.1	Detailed Description	69
8.5	iRegister.h File Reference	70
8.5.1	Detailed Description	70
8.6	iWi11.h File Reference	71
8.6.1	Detailed Description	71
8.7	iWord.h File Reference	72
8.7.1	Detailed Description	72

---

8.8	Loader.h File Reference . . . . .	72
8.8.1	Detailed Description . . . . .	73
8.9	Memory.h File Reference . . . . .	74
8.9.1	Detailed Description . . . . .	74
8.10	ObjParser.cpp File Reference . . . . .	75
8.10.1	Detailed Description . . . . .	75
8.11	ObjParser.h File Reference . . . . .	76
8.11.1	Detailed Description . . . . .	76
8.12	Register.h File Reference . . . . .	77
8.12.1	Detailed Description . . . . .	77
8.13	ResultCodes.h File Reference . . . . .	78
8.13.1	Detailed Description . . . . .	79
8.14	Wi11.h File Reference . . . . .	79
8.14.1	Detailed Description . . . . .	80
8.15	Word.cpp File Reference . . . . .	80
8.15.1	Detailed Description . . . . .	80
8.16	Word.h File Reference . . . . .	81
8.16.1	Detailed Description . . . . .	82





# Chapter 1

## Introduction

### 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). The Wi-11 Simulator 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 Interaction

The components described in this document are, for the most part, representative of the actual hardware components that would be present in the Wi-11 machine. The following section describes these components and their interactions. After that, a list of the [instructions](#) that the Wi-11 can execute (along with their encodings) completes the introduction to this simulator. The rest of the document details the workings of each component and provides the reader with the knowledge necessary for altering, fixing, or even just understanding the code itself.

### 1.3.1 Components

The Wi-11 Simulator uses 5 major components (for a visual, see interactions). The main function, however, is only aware of one: [Wi11](#). It creates one [Wi11](#) object and uses it to parse object files, decode the instructions, and execute them. In order to perform these tasks it first creates [Loader](#), [Memory](#), [Decoder](#), and [Register](#) objects. The [Register](#) objects correspond all those mentioned in the [Introduction](#), with the exception of the CCRs which are declared as their own entity.

#### Note

The [Word](#) class is not described below but nearly all transfers of data and mathematical operations are performed using (an) object(s) of this type.

#### 1.3.1.1 Loading

The [Loader](#) object, receiving a pointer to memory and a filename, creates an [ObjParser](#) object (the fifth major component). The [ObjParser](#) pulls the relevant data from the file and the [Loader](#) puts it into memory. After some input by the user is accepted (assuming the simulator is in debug mode), the [Wi11](#) is ready to begin executing instructions.

#### 1.3.1.2 Executing

The [Wi11](#) component executes instructions in a way very similar to how an actual Wi-11 machine would execute them. It first has the [Memory](#) object return the instruction referenced by the current value of the PC. After incrementing the PC, the raw instruction is given to the [Decoder](#). The [Decoder](#) returns an [Instruction](#) object that allows the [Wi11](#) to call one of its many private functions that correspond (one-to-one) to each kind of

instruction. This process is then repeated until either the HALT trap code is found or the user-specified instruction limit is reached.

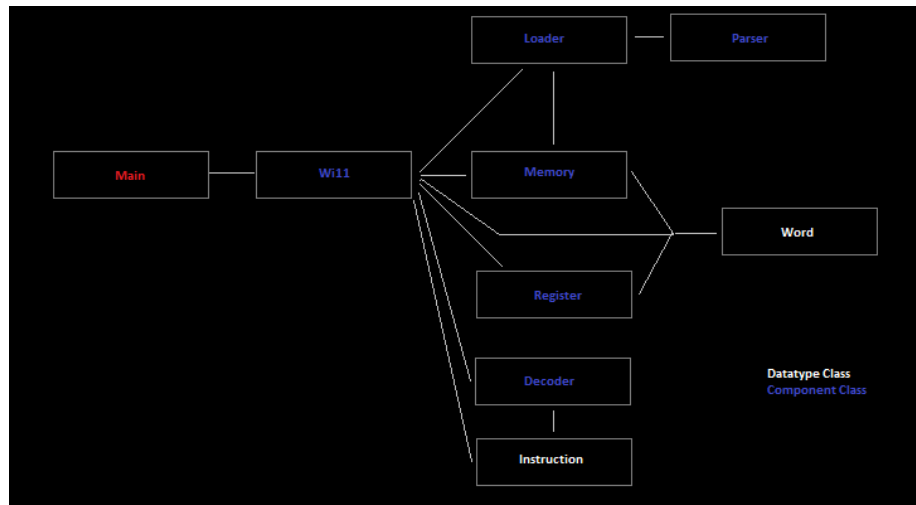


Figure 1.1: This diagram shows the awareness of each component with those operating below it.

### 1.3.2 Wi-11 Instruction Set

# Chapter 2

## Namespace Index

### 2.1 Namespace List

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

<a href="#">Codes</a> (Values corresponding to the results of Wi-11 function calls ) . . . . .	13
<a href="#">Decoder_Directory</a> (Declares register id's and instruction types for each register and instruction ) . . . . .	13



# Chapter 3

## Class Index

### 3.1 Class Hierarchy

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

Decoder . . . . .	15
iDecoder . . . . .	15
iLoader . . . . .	16
Loader . . . . .	49
iMemory . . . . .	17
Memory . . . . .	50
Instruction . . . . .	18
iObjParser . . . . .	20
ObjParser . . . . .	52
iRegister . . . . .	21
Register . . . . .	54
iWi11 . . . . .	28
Wi11 . . . . .	57
iWord . . . . .	40
Word . . . . .	61
ObjectData . . . . .	52
ResultDecoder . . . . .	56
Wi11::CCR . . . . .	60





## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">Decoder</a> . . . . .	15
<a href="#">iDecoder</a> (Defines how Wi-11 instructions are decoded) . . . . .	15
<a href="#">iLoader</a> (Defines how the Wi-11 initializes memory) . . . . .	16
<a href="#">iMemory</a> (Defines the functionality of memory in the Wi-11 machine) . . . . .	17
<a href="#">Instruction</a> (Container to simplify interactions with Wi-11 instructions) . . . . .	18
<a href="#">iObjParser</a> (Defines how object files are processed) . . . . .	20
<a href="#">iRegister</a> (Defines a "register" in the Wi-11 machine) . . . . .	21
<a href="#">iWi11</a> (Defines the internal logic of the Wi-11) . . . . .	28
<a href="#">iWord</a> (Defines a "word" of data on the Wi-11 Machine) . . . . .	40
<a href="#">Loader</a> (Implements <a href="#">iLoader</a> ) . . . . .	49
<a href="#">Memory</a> (Implements <a href="#">iMemory</a> ) . . . . .	50
<a href="#">ObjectData</a> (A simple encoding of a "record") . . . . .	52
<a href="#">ObjParser</a> (Implements <a href="#">iObjParser</a> ) . . . . .	52
<a href="#">Register</a> (Implements <a href="#">iRegister</a> ) . . . . .	54
<a href="#">ResultDecoder</a> (Finds the messages associated with a given result code) . . . . .	56
<a href="#">Wi11</a> (Implements <a href="#">iWi11</a> ) . . . . .	57
<a href="#">Wi11::CCR</a> (Condition code registers: negative, zero, positive) . . . . .	60
<a href="#">Word</a> (Implements <a href="#">iWord</a> ) . . . . .	61



## Chapter 5

# File Index

### 5.1 File List

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

<b>Decoder.h</b>	??
<a href="#">iDecoder.h</a> (Definition of the Wi-11 instruction decoder )	65
<a href="#">iLoader.h</a> (Definition of the Wi-11 program loader )	67
<a href="#">iMemory.h</a> (Definition of Wi-11 memory )	68
<a href="#">iObjParser.h</a> (Definition of the Object File Parser )	69
<a href="#">iRegister.h</a> (Definition of a "register" in the Wi-11 machine )	70
<a href="#">iWi11.h</a> (Definition of the Wi-11 machine simulator )	71
<a href="#">iWord.h</a> (Definition of a "word" of data )	72
<a href="#">Loader.h</a> (Definition of the private data for the "Loader" class )	72
<a href="#">Memory.h</a> (Definition of private data for the "Memory" class )	74
<a href="#">ObjParser.cpp</a> (Implements the declarations in "ObjParser.h" )	75
<a href="#">ObjParser.h</a> (Definition of private data for the "ObjParser" class )	76
<a href="#">Register.h</a> (Definition of private data for the "Register" class )	77
<a href="#">ResultCodes.h</a> (Definition of the Wi-11's run-time messages )	78
<a href="#">Wi11.h</a> (Definition of the private data for the "Wi11" class )	79
<a href="#">Word.cpp</a> (Implements the delcarations in "Word.h" )	80
<a href="#">Word.h</a> (Definition of private data for the "Word" class )	81



## Chapter 6

# Namespace Documentation

### 6.1 Codes Namespace Reference

Values corresponding to the results of Wi-11 function calls.

#### Enumerations

- enum **RESULT** {  
    **ERROR\_0**, **SUCCESS**, **HALT**, **UNDEFINED**,  
    **INVALID\_OBJECT\_FILE**, **INVALID\_DATA\_ENTRY**, **OUT\_OF\_BOUNDS**, **NOT\_HEX**,  
    **FILE\_NOT\_FOUND**, **INVALID\_TRAP\_CODE** }

#### 6.1.1 Detailed Description

Values corresponding to the results of Wi-11 function calls. An enum is used for efficiency. The code can be returned up the collaboration hierarchy quickly so that, if necessary, the program can print an appropriate error message

#### Note

[ResultDecoder](#) can be used to do a look-up of the error message.

### 6.2 Decoder\_Directory Namespace Reference

Declares register id's and instruction types for each register and instruction.

## Enumerations

- enum INSTRUCTION\_TYPE {  
    ADD, AND, BRx, DBUG,  
    JSR, JSRR, LD, LDI,  
    LDR, LEA, NOT, RET,  
    ST, STI, STR, TRAP,  
    ERROR }  
• enum REGISTER\_ID {  
    R0, R1, R2, R3,  
    R4, R5, R6, R7,  
    PC }

### 6.2.1 Detailed Description

Declares register id's and instruction types for each register and instruction. With these definitions, the process of executing instructions is made easier as REGISTER\_ID's and INSTRUCTION\_TYPE's can be used instead of strings.

## Chapter 7

# Class Documentation

### 7.1 Decoder Class Reference

#### Public Member Functions

- [Instruction DecodeInstruction](#) (const [iWord](#) &) const

### 7.2 iDecoder Class Reference

Defines how Wi-11 instructions are decoded.

#### Public Member Functions

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

*Translates the binary instruction into more usable objects.*

#### 7.2.1 Detailed Description

Defines how Wi-11 instructions are decoded. This could be a struct or even a function. It is declared as an object for consistency purposes.

## 7.2.2 Member Function Documentation

**7.2.2.1** `virtual Instruction iDecoder::DecodeInstruction ( const iWord & inst ) const` `[pure virtual]`

Translates the binary instruction into more usable objects.

### Parameters

<i>in</i>	<i>inst</i>	The instruction to be translated.
-----------	-------------	-----------------------------------

### Returns

An [Instruction](#) object as specified in [its documentation](#).

## 7.3 iLoader Class Reference

Defines how the Wi-11 initializes memory.

### Public Member Functions

- `virtual Codes::RESULT Load (const char *filename, iWord &PC_address) const =0`  
*Perform the loads to memory (storing the instructions).*

### 7.3.1 Detailed Description

Defines how the Wi-11 initializes memory. This class loads the instruction from the object file into memory.

## 7.3.2 Member Function Documentation

**7.3.2.1** `virtual Codes::RESULT iLoader::Load ( const char * filename, iWord & PC_address ) const` `[pure virtual]`

Perform the loads to memory (storing the instructions).

### Parameters

<i>in</i>	<i>filename</i>	The name of the object file to be read.
<i>out</i>	<i>PC_address</i>	The value to be stored in the PC to start execution. SUCCESS or, if something goes wrong, an appropriate error code.



**Note**

Multiple object files can be loaded using this, but the PC will be overwritten every time, so only the last End Record will matter (HOWEVER: the End Records still need to be present in each file).

## 7.4 iMemory Class Reference

Defines the functionality of memory in the Wi-11 machine.

**Public Member Functions**

- virtual std::vector< std::vector< **Word** > > **GetUsedMemory** () const =0
- virtual **Word Load** (const **iWord** &w) const =0  
*Performs a load.*
- virtual Codes::RESULT **Reserve** (const **iWord** &initial\_address, const **iWord** &length)=0  
*Reserves an initial section of memory for instructions.*
- virtual Codes::RESULT **Store** (const **iWord** &address, const **Word** &value)=0  
*Performs a store.*

### 7.4.1 Detailed Description

Defines the functionality of memory in the Wi-11 machine. Its size is limited only by addressability ( $2^{16-1}$  16-bit words). It is meant to be implemented in such a way that the memory initialized for instructions can be accessed in constant time while addresses outside this range are accessed in  $n \log n$  time.

### 7.4.2 Member Function Documentation

**7.4.2.1** virtual Codes::RESULT iMemory::Reserve ( const iWord & *initial\_address*, const iWord & *length* ) [pure virtual]

Reserves an initial section of memory for instructions.

**Parameters**

in	<i>initial_address</i>	The smallest address for the instruction memory.
in	<i>length</i>	The number of addresses to reserve.

**Returns**

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

The memory reserved here is dynamically allocated and provides constant-time access to addresses "initial\_address" through "initial\_address"+"length"-1.

**7.4.2.2 virtual Word iMemory::Load ( const iWord & w ) const [pure virtual]**

Performs a load.

**Parameters**

in	w	The address from which to load data.
----	---	--------------------------------------

**Returns**

The data stored a address "w".

**Note**

If "w" is in the range created by Reserve(), it can be accessed in constant time. Otherwise, a maximum of  $n \log n$  time is required if n is the size of memory initialized outside of these boundaries.

**7.4.2.3 virtual Codes::RESULT iMemory::Store ( const iWord & address, const Word & value ) [pure virtual]**

Peforms a store.

**Parameters**

in	address	The address to store the data.
in	value	The data to store at "address".

**Returns**

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

**Note**

The efficiency constraints in Load() apply here as well.

## 7.5 Instruction Struct Reference

Container to simplify interactions with Wi-11 instructions.

## Public Attributes

- `std::vector< Word > data`  
*The arguemnts to the operation (including unnecessary bits).*
- `Decoder_Directory::INSTRUCTION_TYPE type`  
*The type of instruction.*

### 7.5.1 Detailed Description

Container to simplify interactions with Wi-11 instructions.

### 7.5.2 Member Data Documentation

#### 7.5.2.1 `std::vector<Word> Instruction::data`

The arguemnts to the operation (including unnecessary bits).

#### Example:

The add instruction comes in two forms:

- `dest_reg = source_reg_1 + source_reg_2` For this form, the encoding (as ordered) is as follows:
  - `dest_reg`
  - `source_reg_1`
  - a 0
  - 2 unused bits
  - `source_reg_2` These segments are each an element of the data vector.
- `dest_reg = source_reg + immediate_value` For this form, the encoding (as ordered) is as follows:
  - op code
  - `dest_reg`
  - `source_reg_1`
  - a 1
  - a 5-bit immediate value These segments are also each an element of the data vector.

In short, any division specified in [Wi-11 Instruction Set](#) will be an element of the data vector.

#### Note

Both of the overloaded instructions (ADD and AND) can be differentiated by the number of divisions:

- ADD with two registers has 5
- ADD with a register and immediate has 4 and
- AND with two registers has 5
- AND with a register and immediate has 4 Thus the fifth bit (either a 1 or 0) is not needed to determine the variation of the instruction (HOWEVER: the 1 or 0 is still included).

## 7.6 iObjParser Class Reference

Defines how object files are processed.

### Public Member Functions

- virtual [ObjectData](#) [GetNext](#) ()=0  
*Pre-processes the next line of the object file.*
- virtual Codes::RESULT [Initialize](#) (const char \*filename)=0  
*Attempts to open th object file.*

### 7.6.1 Detailed Description

Defines how object files are processed.

### 7.6.2 Member Function Documentation

**7.6.2.1** virtual Codes::RESULT iObjParser::Initialize ( const char \* *filename* ) [pure virtual]

Attempts to open th object file.

#### Parameters

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

**Returns**

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

If another file is open, closes that file first before attempting to open the new one.

Implemented in [ObjParser](#).

**7.6.2.2 virtual ObjectData iObjParser::GetNext ( ) [pure virtual]**

Pre-processes the next line of the object file.

**Precondition**

Initialize must have successfully opened a file.

**Returns**

The encoding of the next instruction.

If there is an error parsing the entry:

- [ObjectData.type](#) = 0;
- [ObjectData.data](#) = [the faulty encoding]

Implemented in [ObjParser](#).

## 7.7 iRegister Class Reference

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

**Public Member Functions**

- 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 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 `Word GetValue () const =0`  
*Retrieves a copy of the word of data store in the register.*
- virtual void `Not ()=0`  
*Performs a bit-wise not.*
- virtual `Register Not () const =0`  
*Performs a bit-wise not.*
- virtual `Register operator+ (const iRegister &r) const =0`  
*A standard add operator.*
- virtual `Register & operator++ ()=0`  
*A standard pre-increment operator.*
- virtual `Register & operator++ (int)=0`  
*A standard post-increment operator.*
- virtual `Register operator- (const iRegister &r) const =0`  
*A standard subtraction operator.*
- virtual `Register & operator= (const Register r)=0`  
*A standard assignment operator.*
- virtual `Register & operator= (const iWord &w)=0`  
*A standard assignment operator.*
- 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 `Store (const iRegister &r)=0`  
*Stores a copy of another register.*
- virtual void `Store (const iWord &w)=0`  
*Stores a word of data.*
- 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.*

### 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.

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

#### 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	<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*".

**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*)".

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

Subtracts a word of data from the calling object.

**Parameters**

in	<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.

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

Subtracts a word of data from the calling object.

**Parameters**

in	<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".

**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)".

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

Performs a bit-wise and.

**Parameters**

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

**Postcondition**

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

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

Performs a bit-wise and.

**Parameters**

<code>in</code>	<code>r</code>	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.

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

Performs a bit-wise "or".

**Parameters**

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

**Postcondition**

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

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

Performs a bit-wise or.

**Parameters**

<code>in</code>	<code>r</code>	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`.

**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).

**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.

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

Stores a word of data.

**Parameters**

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

**Postcondition**

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

**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".

**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)"

**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)"

**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.

**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.

**7.8 iWi11 Class Reference**

Defines the internal logic of the Wi-11.

**Public Member Functions**

- 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.*
- virtual bool [LoadObj](#) (const char \*filename)=0  
*Loads the object file and sets up memory as it describes.*

**Private Member Functions**

- virtual Codes::RESULT [\\_Add](#) (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &SR1, const Decoder\_Directory::REGISTER\_ID &SR2)=0  
*Adds two registers and stores the result in a third.*

- virtual Codes::RESULT [\\_Add](#) (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::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\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &SR1, const Decoder\_Directory::REGISTER\_ID &SR2)=0  
*Bit-wise ands two registers and stores the result in a third.*
- virtual Codes::RESULT [\\_And](#) (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::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 [iRegister](#) & [\\_GetRegister](#) (const Decoder\_Directory::REGISTER\_ID &id)=0  
*Retrieves a reference to the register corresponding to "id".*
- virtual Codes::RESULT [\\_JSR](#) (const [iWord](#) &w, bool)=0  
*Initiate a jump to a subroutine (alter the PC).*
- virtual Codes::RESULT [\\_JSRR](#) (const Decoder\_Directory::REGISTER\_ID &baseR, const [iWord](#) &address, bool)=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\_Directory::REGISTER\_ID &DR, const [iWord](#) &address)=0  
*Loads a word in memory into a register.*
- virtual Codes::RESULT [\\_Loadl](#) (const Decoder\_Directory::REGISTER\_ID &DR, const [iWord](#) &address)=0  
*Performs an indirect load.*
- virtual Codes::RESULT [\\_LoadR](#) (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &baseR, const [iWord](#) &address)=0  
*Performs a register-relative load.*

- virtual Codes::RESULT [\\_Not](#) (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::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 [\\_STI](#) (const Decoder\_Directory::REGISTER\_ID &SR1, const [iWord](#) &address)=0

*Performs an indirect store.*

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

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

- virtual Codes::RESULT [\\_STR](#) (const Decoder\_Directory::REGISTER\_ID &SR1, const Decoder\_Directory::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.*

### 7.8.1 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.

## 7.8.2 Member Function Documentation

**7.8.2.1** `virtual iRegister& iWi11::GetRegister ( const Decoder_Directory::REGISTER_ID & id )`  
`[private, pure virtual]`

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

### Parameters

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

### Returns

A reference to the id'd register.

**7.8.2.2** `virtual Codes::RESULT iWi11::Add ( const Decoder_Directory::REGISTER_ID & DR,  
const Decoder_Directory::REGISTER_ID & SR1, const Decoder_Directory::REGISTER_ID  
& SR2 )` `[private, pure virtual]`

Adds two registers and stores the result in a third.

### Parameters

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

**7.8.2.3** `virtual Codes::RESULT iWi11::Add ( const Decoder_Directory::REGISTER_ID &  
DR, const Decoder_Directory::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.

```
7.8.2.4 virtual Codes::RESULT iWi11::And ( const Decoder_Directory::REGISTER_ID & DR,
      const Decoder_Directory::REGISTER_ID & SR1, const Decoder_Directory::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.

```
7.8.2.5 virtual Codes::RESULT iWi11::And ( const Decoder_Directory::REGISTER_ID &
      DR, const Decoder_Directory::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.

**7.8.2.6** `virtual Codes::RESULT iWi11::_Branch ( const iWord & address ) [private, pure 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.

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

Deprecated?

Does nothing.

**7.8.2.8** `virtual Codes::RESULT iWi11::_JSR ( const iWord & w, bool ) [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. However, the CCR will not be altered for this instruction, despite R7 being altered.

**7.8.2.9** `virtual Codes::RESULT iWi11::JSRR ( const Decoder_Directory::REGISTER_ID & baseR, const iWord & address, bool ) [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**

<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.

**7.8.2.10** `virtual Codes::RESULT iWi11::Load ( const Decoder_Directory::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.

**7.8.2.11** `virtual Codes::RESULT iWi11::_LoadI ( const Decoder_Directory::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.

**7.8.2.12** `virtual Codes::RESULT iWi11::_LoadR ( const Decoder_Directory::REGISTER_ID &  
DR, const Decoder_Directory::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.

```
7.8.2.13 virtual Codes::RESULT iWi11::Not ( const Decoder_Directory::REGISTER_ID &
      DR, const Decoder_Directory::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.

```
7.8.2.14 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.

**7.8.2.15** `virtual Codes::RESULT iWi11::_Store ( const Decoder_Directory::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.

**7.8.2.16** `virtual Codes::RESULT iWi11::_STI ( const Decoder_Directory::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](#).

**7.8.2.17** `virtual Codes::RESULT iWi11::_STR ( const Decoder_Directory::REGISTER_ID & SR1, const Decoder_Directory::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.

**7.8.2.18** `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.

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

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

**Parameters**

<code>in</code>	<code><i>filename</i></code>	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.

**7.8.2.20** `virtual void iWi11::DisplayMemory ( ) const [pure virtual]`

Prints the state of memory to standard out.

**Postcondition**

The calling object is not changed.

**7.8.2.21** `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.

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

Executes the instruction pointed to by the PC.

**Parameters**

<code>in</code>	<code><i>verbose</i></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](#).

## 7.9 iWord Class Reference

Defines a "word" of data on the Wi-11 Machine.



## Public Member Functions

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

- virtual void [SetBit](#) (const int i, bool)=0  
*Sets the 'i'th bit of the value.*
- virtual [Word Subtract](#) (const [iWord](#) &w) const =0  
*Subtracts two words.*
- virtual std::string [ToHex](#) () const =0  
*"To Hexadecimal"*
- 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"*

### 7.9.1 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.

### 7.9.2 Member Function Documentation

#### 7.9.2.1 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.

**7.9.2.2** `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.

**7.9.2.3** `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"

**7.9.2.4** `virtual std::string iWord::ToHex ( ) const` `[pure virtual]`

"To Hexadecimal"

**Postcondition**

The value of the word is not changed.

**Returns**

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

**Examples:**

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

### 7.9.2.5 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".

### 7.9.2.6 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".

### 7.9.2.7 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".

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

Adds two words.

**Parameters**

in	w	A word value to be added.
----	---	---------------------------

**Postcondition**

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

**Returns**

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

**Note**

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

**7.9.2.9 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)".

**7.9.2.10 virtual Word iWord::Subtract ( const iWord & w ) const [pure virtual]**

Subtracts two words.

**Parameters**

in	w	A word value to be subtracted.
----	---	--------------------------------

**Postcondition**

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

**Returns**

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

**Note**

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

**7.9.2.11 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)".

**7.9.2.12 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.

**7.9.2.13 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.

**7.9.2.14 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.

**7.9.2.15 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".

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

A standard assignment operator.

**Parameters**

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

**Returns**

A copy of the parameter.

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

**7.9.2.17 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.

**7.9.2.18 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.

**7.9.2.19 virtual bool iWord::operator[] ( const int i ) const [pure virtual]**

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

**Parameters**

in	i	The index of the bit in question.
----	---	-----------------------------------

**Precondition**

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

**Returns**

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

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



Examples:

If the object holds a value of 4 (0...100 in binary): num[2] = 1.  
If it holds a value of 1 (0...001 in binary): num[0] = 1.  
If it holds a negative value (Starting with a 1 in 2's complement): num[15] = 1.

7.9.2.20 virtual void iWord::SetBit ( const int i, bool ) [pure virtual]

Sets the i'th bit of the value.

Parameters

in	i	The index of the bit in question.
----	---	-----------------------------------

Precondition

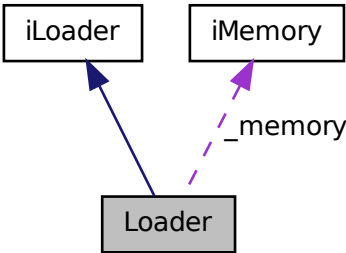
The index must be less than the size of a word, ie. 16.

Works in a similar way to operator[] but sets the bit instead of determining if it is set.

7.10 Loader Class Reference

Implements iLoader.

Collaboration diagram for Loader:



## Public Member Functions

- Codes::RESULT **Load** (const char \*filename, [iWord](#) &PC\_address) const
- [Loader](#) ([iMemory](#) \*mem)

*Set which [Memory](#) object is to be initialized by this object.*

## Private Attributes

- [iMemory](#) \* \_memory

*The reference to [Memory](#).*

### 7.10.1 Detailed Description

Implements [iLoader](#).

### 7.10.2 Constructor & Destructor Documentation

#### 7.10.2.1 [Loader](#)::[Loader](#) ( [iMemory](#) \* *mem* )

Set which [Memory](#) object is to be initialized by this object.

#### Parameters

<i>in</i>	<i>mem</i>	The address where memory is located.
-----------	------------	--------------------------------------

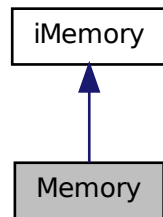
#### Note

Without this there would be nowhere to load the instructions.

## 7.11 Memory Class Reference

Implements [iMemory](#).

Collaboration diagram for Memory:



### Public Member Functions

- `std::vector< std::vector< Word > > GetUsedMemory () const`
- virtual `Word Load (const iWord &) const`
- virtual `Codes::RESULT Reserve (const iWord &initial_address, const iWord &length)`
- virtual `Codes::RESULT Store (const iWord &address, const Word &value)`

### Private Attributes

- `std::vector< Word * > \_bounded\_memory`  
*Provide constant time access to reserved memory.*
- `std::vector< int > \_segment\_lengths`  
*Keep track of the size of reserved memory.*
- `std::vector< int > \_segment\_offsets`  
*Keep track of the initial addresses.*
- `std::map< int, Word > \_unbounded\_memory`  
*Map out-of-bounds values to new Words.*

#### 7.11.1 Detailed Description

Implements [iMemory](#).

## 7.12 ObjectData Struct Reference

A simple encoding of a "record".

### Public Attributes

- `std::vector< std::string > data`  
*The segments of the record.*
- `char type`  
*The type of record: 'H', 'T', or 'E'.*

### 7.12.1 Detailed Description

A simple encoding of a "record".

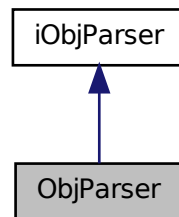
The format of this component is dependent upon the kind of record it is representing.

- Header Record (type = 'H')
  - `data.size() = 3`
    - \* `data[0]` = [Segment Name]
    - \* `data[1]` = [Initial Load Address (as a hex string)]
    - \* `data[2]` = [Segment Length (as a hex string)]
- Text Records (type = 'T')
  - `data.size() = 2`
    - \* `data[0]` = [Address of Data (as a hex string)]
    - \* `data[1]` = [Data (as a hex string)]
- End Records (type = 'E')
  - `data.size() = 1`
    - \* `data[0]` = [Initial PC Address (as a hex string)]

## 7.13 ObjParser Class Reference

Implements [iObjParser](#).

Collaboration diagram for ObjParser:



### Public Member Functions

- [ObjectData GetNext \(\)](#)  
*Reads the next line from the current object file and parses it into an [ObjectData](#) struct for use by the loader.*
- `Codes::RESULT` [Initialize](#) (const char \*name)  
*Closes `_fileStream` if necessary, then opens the file defined by "name".*
- [~ObjParser \(\)](#)  
*Closes a file, if necessary, when an [iObjParser](#) object goes out of scope..*

### Private Attributes

- `std::ifstream` [\\_fileStream](#)  
*Maintains an input stream from the object file specified by the "name" parameter to `Initialize`.*

#### 7.13.1 Detailed Description

Implements [iObjParser](#).

## 7.13.2 Member Function Documentation

### 7.13.2.1 Codes::RESULT ObjParser::Initialize ( const char \* *name* ) [virtual]

Closes `_fileStream` if necessary, then opens the file defined by "name".

#### Parameters

<i>name</i>	The name of the file to be opened, including extension.
-------------	---

#### Returns

Codes::SUCCESS if the file is successfully opened, Codes::FILE\_NOT\_FOUND otherwise.

Implements [iObjParser](#).

### 7.13.2.2 ObjectData ObjParser::GetNext ( ) [virtual]

Reads the next line from the current object file and parses it into an [ObjectData](#) struct for use by the loader.

#### Precondition

Initialize(name) has been called and `_fileStream` is currently open.

#### Postcondition

The get pointer within `_fileStream` has been advanced to the next line.

#### Returns

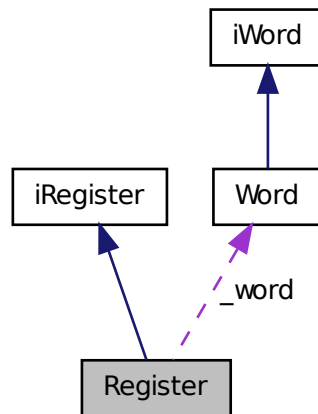
A well-formed [ObjectData](#) struct if a valid line is received, a 'dummy' [ObjectData](#) struct otherwise.

Implements [iObjParser](#).

## 7.14 Register Class Reference

Implements [iRegister](#).

Collaboration diagram for Register:



### Public Member Functions

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

- [Register Subtract](#) (const [iRegister](#) &r) const
- void [Subtract](#) (const [iWord](#) &w)

### Private Attributes

- [Word \\_word](#)

*The word of data held in the register.*

#### 7.14.1 Detailed Description

Implements [iRegister](#).

## 7.15 ResultDecoder Class Reference

Finds the messages associated with a given result code.

### Public Member Functions

- std::string [Find](#) (const Codes::RESULT &result) const

*Looks up a result code.*

- [ResultDecoder](#) ()

*Generates the code-to-message mappings.*

### Private Attributes

- std::map< Codes::RESULT, std::string > [\\_codes](#)

*Maps a result code to, in every case but SUCCESS, an error message.*

#### 7.15.1 Detailed Description

Finds the messages associated with a given result code.



## 7.15.2 Member Function Documentation

### 7.15.2.1 `string ResultDecoder::Find ( const Codes::RESULT & result ) const`

Looks up a result code.

#### Parameters

<code>in</code>	<code><i>result</i></code>	The result code to look up.
-----------------	----------------------------	-----------------------------

#### Returns

The messages associated with "result".

## 7.15.3 Member Data Documentation

### 7.15.3.1 `std::map<Codes::RESULT, std::string> ResultDecoder::_codes` `[private]`

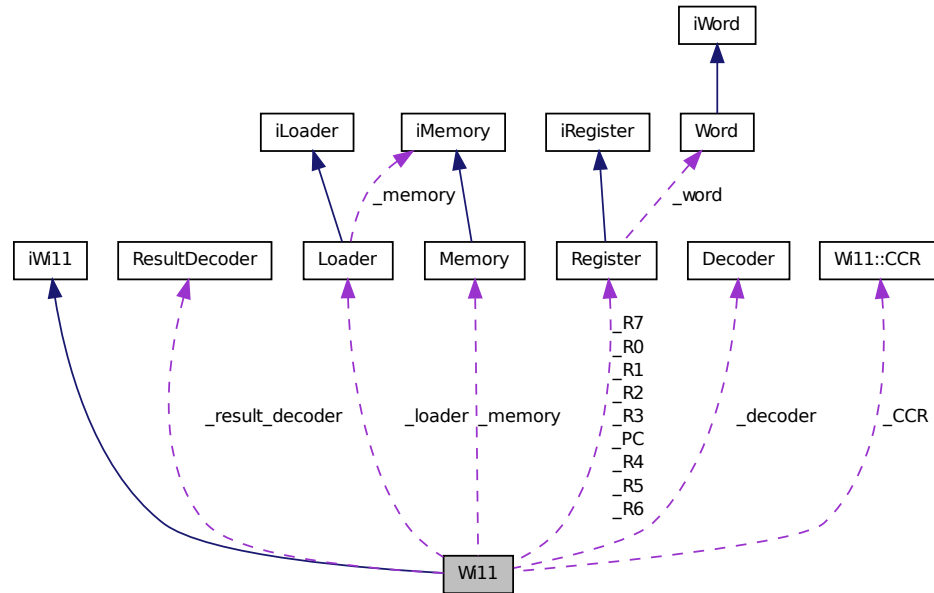
Maps a result code to, in every case but SUCCESS, an error message.

It is static because the result code messages should be available from anywhere.

## 7.16 Wi11 Class Reference

Implements [iWi11](#).

Collaboration diagram for Wi11:



## Classes

- struct **CCR**

*Condition code registers: negative, zero, positive.*

## Public Member Functions

- virtual void **DisplayMemory** () const
- virtual void **DisplayRegisters** () const
- virtual bool **ExecuteNext** (bool verbose=false)
- virtual bool **LoadObj** (const char \*)
- void **poo** () const
- **Wi11** ()

*Creates and organizes the componts of the **Wi11** machine.*

## Private Member Functions

- virtual Codes::RESULT **\_Add** (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &SR1, const Decoder\_Directory::REGISTER\_ID &SR2)
- virtual Codes::RESULT **\_Add** (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &SR1, const **iWord** &immediate)
- virtual Codes::RESULT **\_And** (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &SR1, const **iWord** &immediate)
- virtual Codes::RESULT **\_And** (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &SR1, const Decoder\_Directory::REGISTER\_ID &SR2)
- virtual Codes::RESULT **\_Branch** (const **iWord** &address)
- virtual Codes::RESULT **\_Debug** ()
- **iRegister** & **\_GetRegister** (const Decoder\_Directory::REGISTER\_ID &)
- virtual Codes::RESULT **\_JSR** (const **iWord** &, bool)
- virtual Codes::RESULT **\_JSRR** (const Decoder\_Directory::REGISTER\_ID &baseR, const **iWord** &address, bool link)
- virtual Codes::RESULT **\_Load** (const Decoder\_Directory::REGISTER\_ID &DR, const **iWord** &address)
- virtual Codes::RESULT **\_LoadEA** (const Decoder\_Directory::REGISTER\_ID &DR, const **iWord** &address)
- virtual Codes::RESULT **\_Loadl** (const Decoder\_Directory::REGISTER\_ID &DR, const **iWord** &address)
- virtual Codes::RESULT **\_LoadR** (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &baseR, const **iWord** &address)
- virtual Codes::RESULT **\_Not** (const Decoder\_Directory::REGISTER\_ID &DR, const Decoder\_Directory::REGISTER\_ID &SR)
- std::string **\_RegisterID2String** (const Decoder\_Directory::REGISTER\_ID &) const

*Translates an internal **Register** ID identifier into human readable string format.*

- virtual Codes::RESULT **\_Ret** ()
- virtual Codes::RESULT **\_STI** (const Decoder\_Directory::REGISTER\_ID &SR, const **iWord** &address)
- virtual Codes::RESULT **\_Store** (const Decoder\_Directory::REGISTER\_ID &SR, const **iWord** &address)
- virtual Codes::RESULT **\_STR** (const Decoder\_Directory::REGISTER\_ID &SR, const Decoder\_Directory::REGISTER\_ID &baseR, const **iWord** &address)
- virtual Codes::RESULT **\_Trap** (const **iWord** &code)
- void **\_UpdateCCR** (int)
- Decoder\_Directory::REGISTER\_ID **\_Word2RegisterID** (const **Word** &) const

## Private Attributes

- struct [Wi11::CCR\\_CCR](#)
- [Decoder\\_decoder](#)  
*For decoding instructions fetch from memory.*
- [Loader\\_loader](#)  
*For loading the object file.*
- [Memory\\_memory](#)  
*Acts as the Wi-11's memory.*
- [Register\\_PC](#)
- [Register\\_R0](#)  
*The 8 general purpose registers and PC.*
- [Register\\_R1](#)
- [Register\\_R2](#)
- [Register\\_R3](#)
- [Register\\_R4](#)
- [Register\\_R5](#)
- [Register\\_R6](#)
- [Register\\_R7](#)
- [ResultDecoder\\_result\\_decoder](#)  
*For error messages.*

### 7.16.1 Detailed Description

Implements [iWi11](#).

### 7.16.2 Constructor & Destructor Documentation

#### 7.16.2.1 [Wi11::Wi11](#) ( )

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

## 7.17 Wi11::CCR Struct Reference

Condition code registers: negative, zero, positive.

## Public Attributes

- bool n
- bool p
- bool z

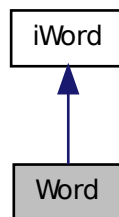
### 7.17.1 Detailed Description

Condition code registers: negative, zero, positive.

## 7.18 Word Class Reference

Implements [iWord](#).

Collaboration diagram for Word:



## Public Member Functions

- [Word Add](#) (const [iWord](#) &w) const
- [Word And](#) (const [iWord](#) &w) const
- void [Copy](#) (const [iWord](#) &w)
- bool [FromHex](#) (const std::string &str)
- bool [FromInt](#) (int value)
- bool [FromStr](#) (const std::string &str)
- [Word Not](#) () const
- [Word operator+](#) (const [iWord](#) &w) const
- [iWord & operator++](#) (int)

- `iWord & operator++ ()`
- `Word operator- (const iWord &w) const`
- `Word & operator= (const Word w)`
- `bool operator[] (const int i) const`
- `Word Or (const iWord &w) const`
- `void SetBit (const int, bool)`
- `Word Subtract (const iWord &w) const`
- `std::string ToHex () const`
- `int ToInt () const`
- `int ToInt2Complement () const`
- `std::string ToStr () const`

### Private Member Functions

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

### Private Attributes

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

## 7.18.1 Detailed Description

Implements `iWord`.

## 7.18.2 Member Function Documentation

### 7.18.2.1 `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[]()`.

### 7.18.3 Member Data Documentation

#### 7.18.3.1 `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.





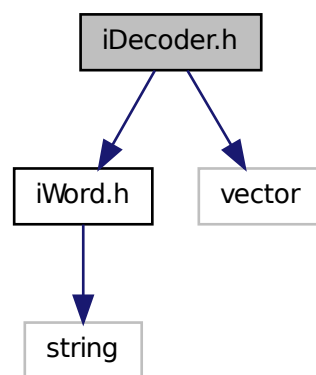
## Chapter 8

# File Documentation

### 8.1 iDecoder.h File Reference

Definition of the Wi-11 instruction decoder.

Include dependency graph for iDecoder.h:



## Classes

- class [iDecoder](#)  
*Defines how Wi-11 instructions are decoded.*
- struct [Instruction](#)  
*Container to simplify interactions with Wi-11 instructions.*

## Namespaces

- namespace [Decoder\\_Directory](#)  
*Declares register id's and instruction types for each register and instruction.*

## Enumerations

- enum INSTRUCTION\_TYPE {  
    ADD, AND, BRx, DBUG,  
    JSR, JSRR, LD, LDI,  
    LDR, LEA, NOT, RET,  
    ST, STI, STR, TRAP,  
    ERROR }  
• enum REGISTER\_ID {  
    R0, R1, R2, R3,  
    R4, R5, R6, R7,  
    PC }

### 8.1.1 Detailed Description

Definition of the Wi-11 instruction decoder.

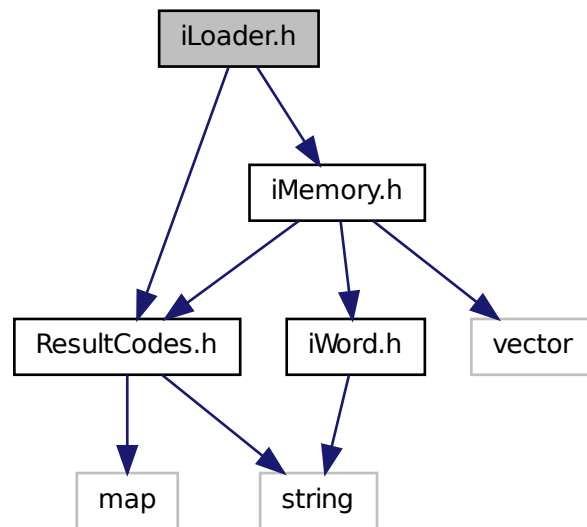
#### Author

Joshua Green  
Andrew Groot

## 8.2 iLoader.h File Reference

Definition of the Wi-11 program loader.

Include dependency graph for iLoader.h:



### Classes

- class [iLoader](#)  
*Defines how the Wi-11 initializes memory.*

### 8.2.1 Detailed Description

Definition of the Wi-11 program loader.

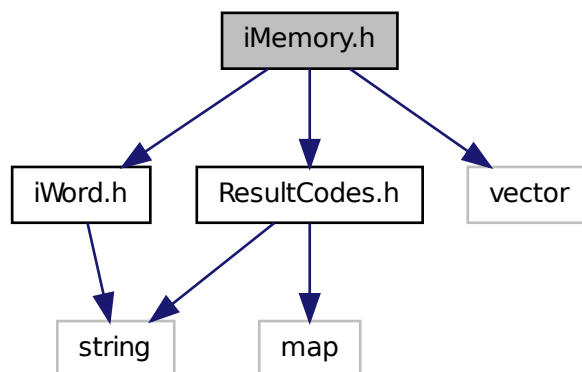
#### Author

Joshua Green  
Andrew Groot

## 8.3 iMemory.h File Reference

Definition of Wi-11 memory.

Include dependency graph for iMemory.h:



### Classes

- class `iMemory`

*Defines the functionality of memory in the Wi-11 machine.*

### 8.3.1 Detailed Description

Definition of Wi-11 memory.

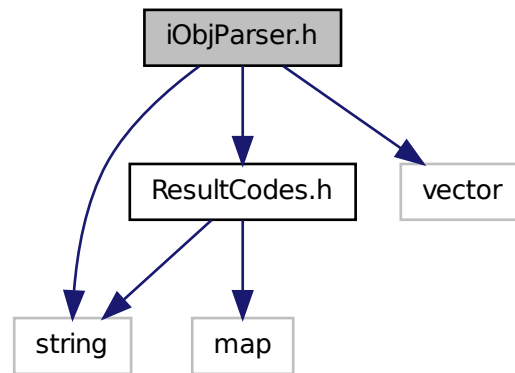
#### Author

Joshua Green  
Andrew Groot

## 8.4 iObjParser.h File Reference

Definition of the Object File Parser.

Include dependency graph for iObjParser.h:



### Classes

- class `iObjParser`  
*Defines how object files are processed.*
- struct `ObjectData`  
*A simple encoding of a "record".*

### 8.4.1 Detailed Description

Definition of the Object File Parser.

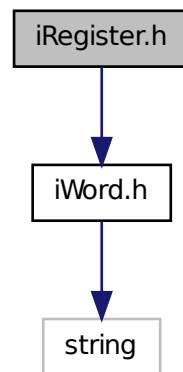
#### Author

Joshua Green  
Andrew Groot

## 8.5 iRegister.h File Reference

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

Include dependency graph for iRegister.h:



### Classes

- class [iRegister](#)

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

### 8.5.1 Detailed Description

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

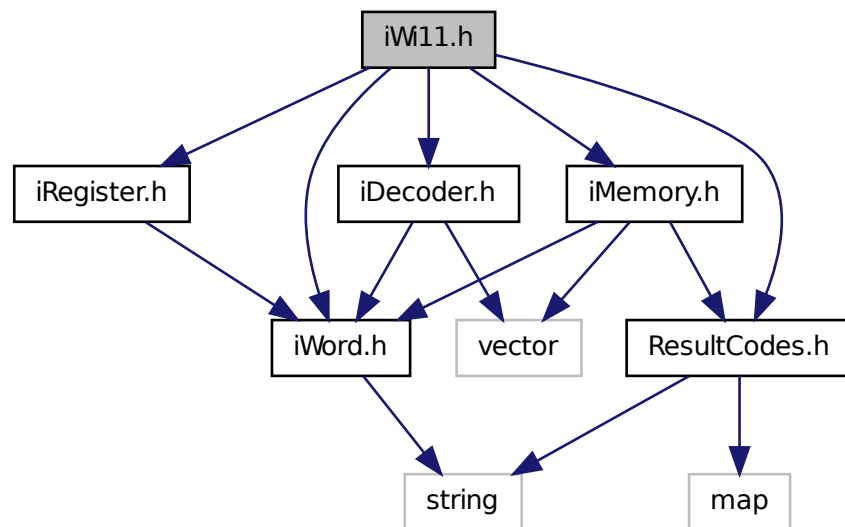
#### Author

Joshua Green  
Andrew Groot

## 8.6 iWi11.h File Reference

Definition of the Wi-11 machine simulator.

Include dependency graph for iWi11.h:



### Classes

- class [iWi11](#)

*Defines the internal logic of the Wi-11.*

### 8.6.1 Detailed Description

Definition of the Wi-11 machine simulator.

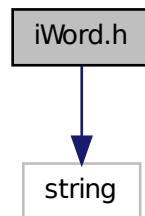
#### Author

Joshua Green  
Andrew Groot

## 8.7 iWord.h File Reference

Definition of a "word" of data.

Include dependency graph for iWord.h:



### Classes

- class [iWord](#)

*Defines a "word" of data on the Wi-11 Machine.*

### 8.7.1 Detailed Description

Definition of a "word" of data.

#### Author

Joshua Green  
Andrew Groot

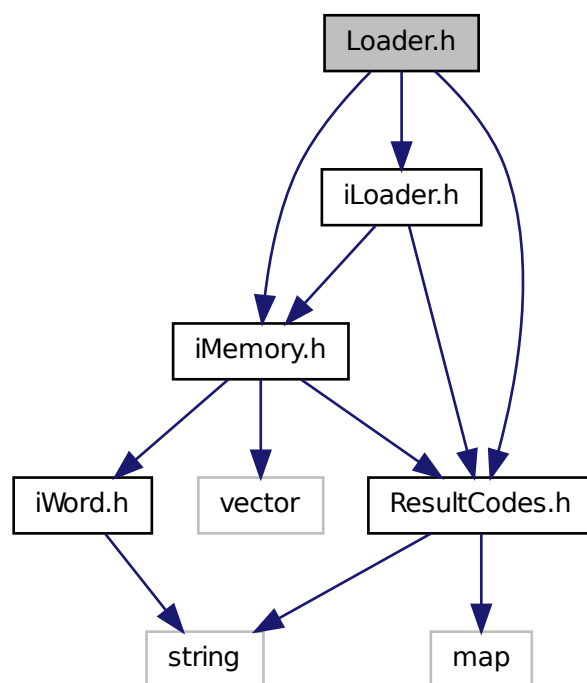
Defines the operations and signatures by which a "word" class should operate. The signatures, while intended to be coded to the interface, are done as to this as C++ allows.

## 8.8 Loader.h File Reference

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



Include dependency graph for Loader.h:



## Classes

- class [Loader](#)  
*Implements [iLoader](#).*

### 8.8.1 Detailed Description

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

#### Author

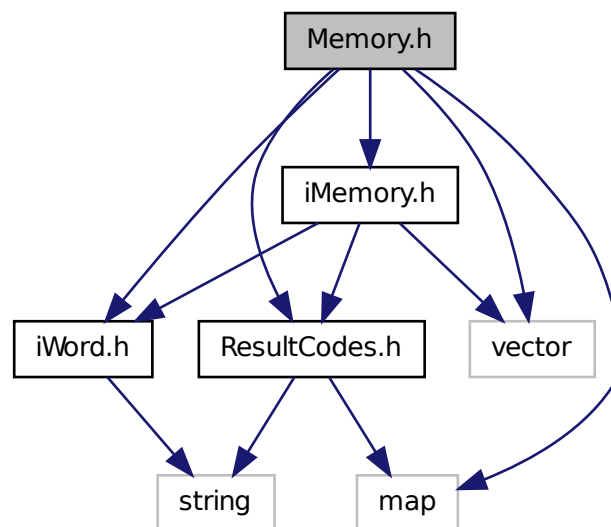
Logan Coulson

Joshua Green  
Andrew Groot

## 8.9 Memory.h File Reference

Definition of private data for the "Memory" class.

Include dependency graph for Memory.h:



### Classes

- class `Memory`  
*Implements `iMemory`.*

### 8.9.1 Detailed Description

Definition of private data for the "Memory" class.

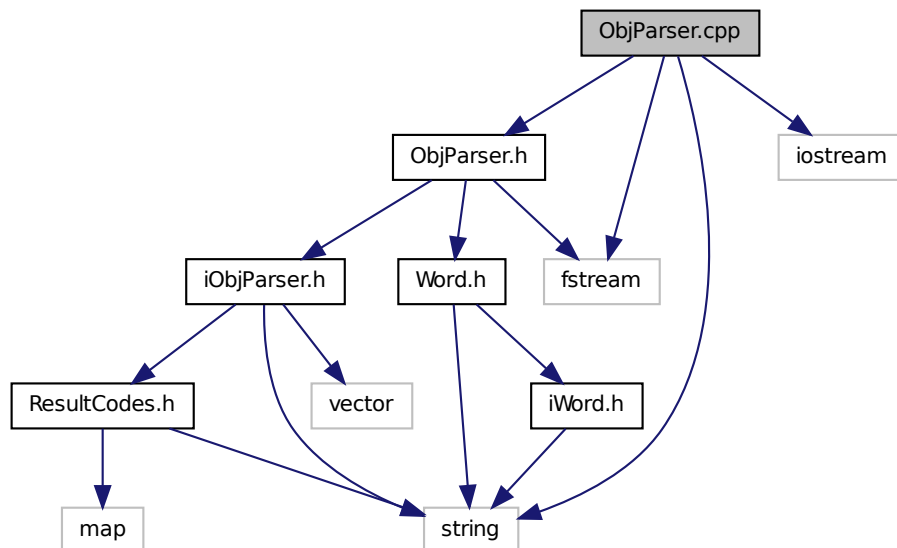
**Author**

Joshua Green  
Andrew Groot

## 8.10 ObjParser.cpp File Reference

Implements the declarations in "ObjParser.h".

Include dependency graph for ObjParser.cpp:



### 8.10.1 Detailed Description

Implements the declarations in "ObjParser.h".

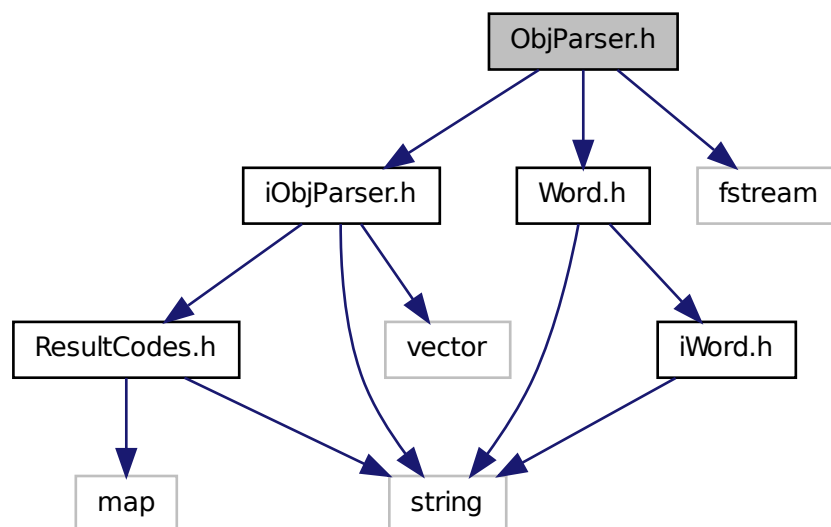
**Author**

Ryan Paulson

## 8.11 ObjParser.h File Reference

Definition of private data for the "ObjParser" class.

Include dependency graph for ObjParser.h:



### Classes

- class [ObjParser](#)  
*Implements [iObjParser](#).*

#### 8.11.1 Detailed Description

Definition of private data for the "ObjParser" class.

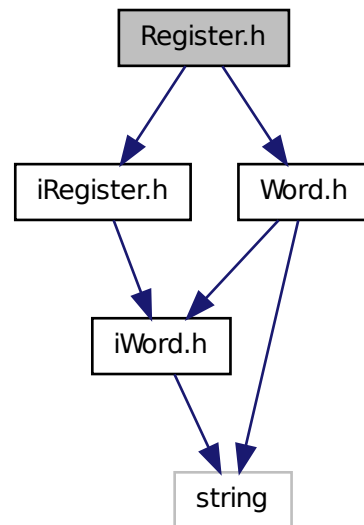
#### Author

Ryan Paulson

## 8.12 Register.h File Reference

Definition of private data for the "Register" class.

Include dependency graph for Register.h:



### Classes

- class [Register](#)  
*Implements [iRegister](#).*

### 8.12.1 Detailed Description

Definition of private data for the "Register" class.

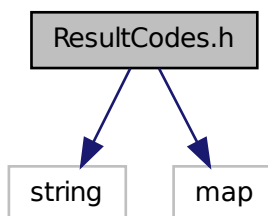
#### Author

Andrew Groot

## 8.13 ResultCodes.h File Reference

Definition of the Wi-11's run-time messages.

Include dependency graph for ResultCodes.h:



### Classes

- class [ResultDecoder](#)

*Finds the messages associated with a given result code.*

### Namespaces

- namespace [Codes](#)

*Values corresponding to the results of Wi-11 function calls.*

### Enumerations

- enum `RESULT` {  
    `ERROR_0`, `SUCCESS`, `HALT`, `UNDEFINED`,  
    `INVALID_OBJECT_FILE`, `INVALID_DATA_ENTRY`, `OUT_OF_BOUNDS`, `NOT_HEX`,  
    `FILE_NOT_FOUND`, `INVALID_TRAP_CODE` }

### 8.13.1 Detailed Description

Definition of the Wi-11's run-time messages.

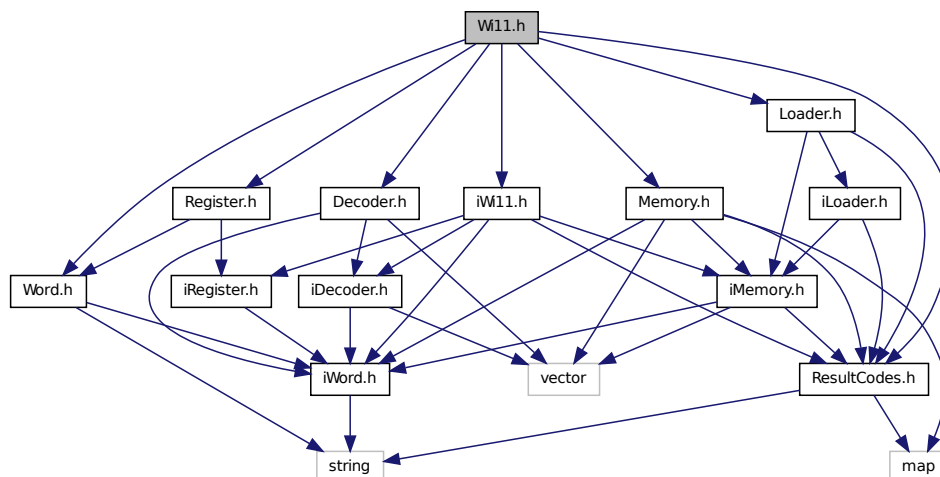
#### Author

Joshua Green  
Andrew Groot

## 8.14 Wi11.h File Reference

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

Include dependency graph for Wi11.h:



### Classes

- class `Wi11`  
Implements `iWi11`.
- struct `Wi11::CCR`  
Condition code registers: negative, zero, positive.

### 8.14.1 Detailed Description

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

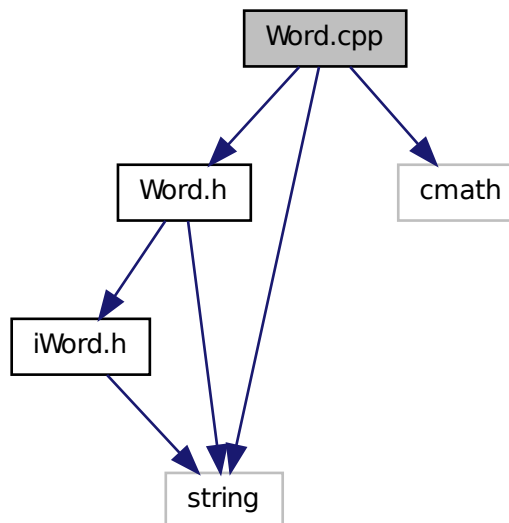
#### Author

Joshua Green  
Andrew Groot

## 8.15 Word.cpp File Reference

Implements the delcarations in "Word.h".

Include dependency graph for Word.cpp:



### 8.15.1 Detailed Description

Implements the delcarations in "Word.h".



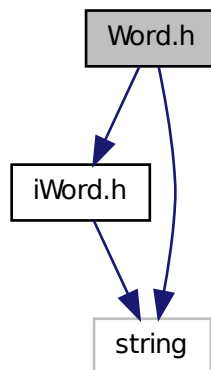
**Author**

Joshua Green  
Andrew Groot

## 8.16 Word.h File Reference

Definition of private data for the "Word" class.

Include dependency graph for Word.h:

**Classes**

- class [Word](#)  
*Implements [iWord](#).*

**Defines**

- #define **WORD\_SIZE** 16

### 8.16.1 Detailed Description

Definition of private data for the "Word" class.

**Author**

Joshua Green

Andrew Groot

# Index

- `_Add`
    - `iWi11`, [31](#)
  - `_And`
    - `iWi11`, [32](#)
  - `_Branch`
    - `iWi11`, [33](#)
  - `_Debug`
    - `iWi11`, [33](#)
  - `_GetRegister`
    - `iWi11`, [31](#)
  - `_HasBit`
    - `Word`, [62](#)
  - `_JSR`
    - `iWi11`, [33](#)
  - `_JSRR`
    - `iWi11`, [34](#)
  - `_Load`
    - `iWi11`, [34](#)
  - `_LoadI`
    - `iWi11`, [35](#)
  - `_LoadR`
    - `iWi11`, [35](#)
  - `_Not`
    - `iWi11`, [36](#)
  - `_Ret`
    - `iWi11`, [36](#)
  - `_STI`
    - `iWi11`, [37](#)
  - `_STR`
    - `iWi11`, [38](#)
  - `_Store`
    - `iWi11`, [37](#)
  - `_Trap`
    - `iWi11`, [38](#)
  - `_codes`
    - `ResultDecoder`, [57](#)
  - `_value`
    - `Word`, [63](#)
- `Add`
  - `iRegister`, [23](#)
  - `iWord`, [45](#)
- `And`
  - `iRegister`, [25](#)
  - `iWord`, [46](#)
- `Codes`, [13](#)
- `Copy`
  - `iWord`, [47](#)
- `data`
  - `Instruction`, [19](#)
- `DecodeInstruction`
  - `iDecoder`, [16](#)
- `Decoder`, [15](#)
- `Decoder_Directory`, [13](#)
- `DisplayMemory`
  - `iWi11`, [39](#)
- `DisplayRegisters`
  - `iWi11`, [40](#)
- `ExecuteNext`
  - `iWi11`, [40](#)
- `Find`
  - `ResultDecoder`, [57](#)
- `FromHex`
  - `iWord`, [44](#)
- `FromInt`
  - `iWord`, [43](#)
- `FromStr`
  - `iWord`, [44](#)
- `GetNext`
  - `iObjParser`, [21](#)

- ObjParser, 54
- GetValue
  - iRegister, 23
- iDecoder, 15
  - DecodeInstruction, 16
- iDecoder.h, 65
- iLoader, 16
  - Load, 16
- iLoader.h, 67
- iMemory, 17
  - Load, 18
  - Reserve, 17
  - Store, 18
- iMemory.h, 68
- Initialize
  - iObjParser, 20
  - ObjParser, 54
- Instruction, 18
  - data, 19
- iObjParser, 20
  - GetNext, 21
  - Initialize, 20
- iObjParser.h, 69
- iRegister, 21
  - Add, 23
  - And, 25
  - GetValue, 23
  - Not, 26
  - operator+, 24
  - operator++, 27, 28
  - operator-, 25
  - operator=, 27
  - Or, 25, 26
  - Store, 26, 27
  - Subtract, 24
- iRegister.h, 70
- iWi11, 28
  - \_Add, 31
  - \_And, 32
  - \_Branch, 33
  - \_Debug, 33
  - \_GetRegister, 31
  - \_JSR, 33
  - \_JSRR, 34
  - \_Load, 34
  - \_LoadI, 35
  - \_LoadR, 35
  - \_Not, 36
  - \_Ret, 36
  - \_STI, 37
  - \_STR, 38
  - \_Store, 37
  - \_Trap, 38
  - DisplayMemory, 39
  - DisplayRegisters, 40
  - ExecuteNext, 40
  - LoadObj, 39
- iWi11.h, 71
- iWord, 40
  - Add, 45
  - And, 46
  - Copy, 47
  - FromHex, 44
  - FromInt, 43
  - FromStr, 44
  - Not, 47
  - operator+, 45
  - operator++, 48
  - operator-, 46
  - operator=, 47
  - Or, 46
  - SetBit, 49
  - Subtract, 45
  - ToHex, 43
  - ToInt, 42
  - ToInt2Complement, 42
  - ToStr, 43
- iWord.h, 72
- Load
  - iLoader, 16
  - iMemory, 18
- Loader, 49
  - Loader, 50
- Loader.h, 72
- LoadObj
  - iWi11, 39
- Memory, 50
- Memory.h, 74
- Not

- iRegister, [26](#)
- iWord, [47](#)
- ObjectData, [52](#)
- ObjParser, [52](#)
  - GetNext, [54](#)
  - Initialize, [54](#)
- ObjParser.cpp, [75](#)
- ObjParser.h, [76](#)
- operator+
  - iRegister, [24](#)
  - iWord, [45](#)
- operator++
  - iRegister, [27](#), [28](#)
  - iWord, [48](#)
- operator-
  - iRegister, [25](#)
  - iWord, [46](#)
- operator=
  - iRegister, [27](#)
  - iWord, [47](#)
- Or
  - iRegister, [25](#), [26](#)
  - iWord, [46](#)
- Register, [54](#)
- Register.h, [77](#)
- Reserve
  - iMemory, [17](#)
- ResultCodes.h, [78](#)
- ResultDecoder, [56](#)
  - \_codes, [57](#)
  - Find, [57](#)
- SetBit
  - iWord, [49](#)
- Store
  - iMemory, [18](#)
  - iRegister, [26](#), [27](#)
- Subtract
  - iRegister, [24](#)
  - iWord, [45](#)
- ToHex
  - iWord, [43](#)
- ToInt
  - iWord, [42](#)
- ToInt2Complement
  - iWord, [42](#)
- ToStr
  - iWord, [43](#)
- Wi11, [57](#)
  - Wi11, [60](#)
- Wi11.h, [79](#)
- Wi11::CCR, [60](#)
- Word, [61](#)
  - \_HasBit, [62](#)
  - \_value, [63](#)
- Word.cpp, [80](#)
- Word.h, [81](#)