OneUp Wi-11 Simulator - Programmer's Guide

Generated by Doxygen 1.7.2

Sun Jan 23 2011 20:38:02

Contents

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 (ususally 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 instuctions.

1.2.2 Text Records

Following the Header Record are serveral 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, recieving 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.

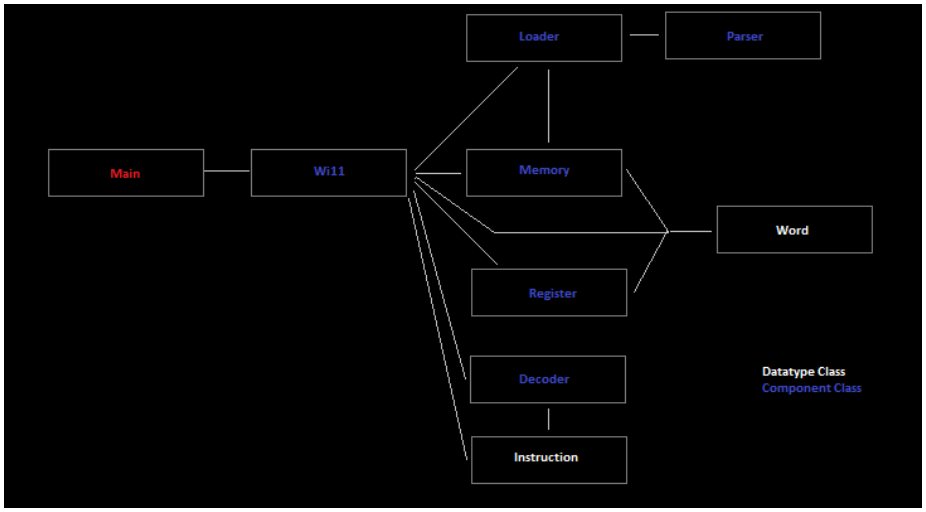


Figure 1.1: This diagram shows the awareness of each component with those operating below it.

1.3.2 Wi-11 Instruction Set

This section describes the format of each operation on the Wi-11. First there are necessary definitions and then the list of instructions. The name of each instruction is followed by the opcode; this includes any base conversions that may be necessary. Then there is a list of the arguments to the command. The opcode is the first four bits of the instruction; the list following the opcode delagates purpose to the following 12 bits.

1.3.2.1 Offsets

Offsets to the PC are used by concatenating them with the PC. Specifically, the first 7 bits of the PC and the 9 bit offset form the new PC value. This essentially separates memory into pages (the first seven bits of the PC corresponding a "page number").

1.3.2.2 Indexes

Indexes are used to specify a distance from a base value. Generally, there is a register holding an address. The index is added to the base address as a positive quantity

(zero-extended) in order to form a new address. Because the index is zero-extended, the new address is always greater than the base address.

1.3.2.3 Intructions

- ADD (two registers), OPCODE: 0001 (1)
  - 3 bits: The destination register
  - 3 bits: First source register
  - 1 bit: A zero
  - 2 bits: Junk - not used.
  - 3 bits: Second source register
- ADD (register and immediate), OPCODE: 0001 (1)
  - 3 bits: The destination register
  - 3 bits: The source register
  - 1 bit: A one
  - 5 bits: An immediate value (2's complement)
- AND (two registers), OPCODE: 0101 (5)
  - 3 bits: The destination register
  - 3 bits: First source register
  - 1 bit: A zero
  - 2 bits: Junk - not used
  - 3 bits: Second source register
- AND (register and immediate), OPCODE: 0101 (5)
  - 3 bits: The destination register
  - 3 bits: The source register
  - 1 bit: A one
  - 5 bits: An immediate value (2's complement)
- BRx, OPCODE: 0000 (0)
  - 1 bit: Corresponds to the CCR's negative bit
  - 1 bit: Corresponds to the CCR's zero bit
  - 1 bit: Corresponds to the CCR's positive bit
  - 9-bits: An offset to the PC
- DBUG, OPCODE: 1000 (8)

  - 12 bits: Junk - not used

- JSR, OPCODE: 0100 (4)
  - 1 bit: The link bit (The PC is stored in R7 if this is set)
  - 2 bits: Junk - not used
  - 9 bits: An offset to the PC
- JSRR, OPCODE: 1100 (12 - C)
  - 1 bit: The link bit (The PC is stored in R7 if this is set)
  - 2 bits: Junk - not used
  - 3 bits: A base register
  - 6 bits: An index to the base register
- LD, OPCODE: 0010 (2)
  - 3 bits: The destination register
  - 9 bits: An offset to the PC
- LDI, OPCODE: 1010 (10 - A)
  - 3 bits: The destination register
  - 9 bits: An offset to the PC
- LDR, OPCODE: 0110 (6)
  - 3 bits: The destination register
  - 3 bits: A base register
  - 6 bits: An index to the base register
- LEA, OPCODE: 1110 (14 - E)
  - 3 bits: The destination register
  - 9 bits: An offset to the PC
- NOT, OPCODE: 1001 (9)
  - 3 bits: The destination register
  - 3 bits: The source register
  - 6 bits: Junk - not used
- RET, OPCODE: 1101 (13 - D)
  - 12 bits: Junk - not used
- ST, OPCODE: 0011 (3)

  - 3 bits: The source register
  - 9 bits: An offset to the PC
- STI, OPCODE: 1011 (11 - B)
  - 3 bits: The source register
  - 9 bits: An offset to the PC
- STR, OPCODE: 0111 (7)
  - 3 bits: The source register
  - 3 bits: A base register
  - 6 bits: An index to the base register
- TRAP, OPCODE: 1111 (15 - F)
  - 4 bits: Junk - not used
  - 8 bits: A trap vector

1.3.2.4 Traps

Traps execute a system call. The details of these so-called "trap vectors" are below.

- 0x21 - OUT
  - Print the ASCII character in the last 8 bits of R0.
- 0x22 - PUTS
  - Print the string starting at the address in R0 and ending at a null character.
- 0x23 - IN
  - Prompt for and read an ASCII character. Put the result in R0.
- 0x25 - HALT
  - End execution.
- 0x31 - OUTN
  - Print the value in R0 as a decimal integer.
- 0x33 - INN
  - Prompt for and read a decimal number. Put the result in R0.
- 0x43 - RND
  - Generate a random integer and store it in R0.

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Class Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

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_HEADER_ENTRY, **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 fucntion.
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

in	*inst*	The instruction to be translated.

Returns

An Instruction object as specificied 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

in	*filename*	The name of the object file to be read.
out	*PC_address*	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

  Peforms 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 nlogn 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	*initial_address*	The smallest address for the instruction memory.
in	*length*	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 nlogn 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 unecessary 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 unecessary 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

in	*filename*	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 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 iWord &w)=0

  A standard assignment operator.

- virtual Register Or (const iRegister &r) const =0

  Performs a bit-wise or.

- virtual void Or (const iWord &w)=0

  Performs a bit-wise "or".

- virtual void Store (const iWord &w)=0

  Performs a bit-wise not.

- virtual void Store (const iRegister &r)=0

  Stores a copy of another register.

- virtual Register Subtract (const iRegister &r) const =0

  Subtracts a word of data from the calling object.

- virtual void Subtract (const iWord &w)=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 inteface are meant to mimic the functionality of the Wi-11 machine, allowing for simplified execution of the instructions therein. This interace 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

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.

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".

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	*w*	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	*r*	The value to be subtracted.

Postcondition

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

Returns

A new Register object holding the value of the calling object minus the value in "r".

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

in	*w*	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

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.

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

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

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 void iRegister::Store (const iWord & *w*) [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.

Stores a word of data.

Parameters

in	*w*	The value to be store.

Postcondition

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

7.7.2.14 virtual void iRegister::Store (const iRegister & *r*) [pure virtual]

Stores a copy of another register.

Parameters

in	*r*	The register to be copied.

Postcondition

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

7.7.2.15 virtual Register& iRegister::operator= (const iWord & *w*) [pure virtual]

A standard assignment operator.

Note

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

7.7.2.16 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.

7.7.2.17 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 _LoadI (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

  Perfroms 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

in	*id*	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

out	*DR*	The destination register.
in	*SR1*	The first source register.
in	*SR2*	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	*DR*	The destination register.
in	*SR1*	The source register.
in	*immediate*	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	*DR*	The destination register.
in	*SR1*	The first source register.
in	*SR2*	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	*DR*	The destination register.
in	*SR1*	The source register.
in	*immediate*	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	*address*	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

in	*w*	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, depite 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

in	*address*	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, depite 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	*DR*	The destination register.
in	*address*	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	*DR*	The destination register.
in	*address*	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	*DR*	The destination register.
in	*baseR*	A register whose value works as a base address.
in	*address*	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	*DR*	The destination register.
in	*SR*	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	*SR1*	The source register (holds the data to be stored).
in	*address*	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	*SR1*	The source register (holds the data to be stored).
in	*address*	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]

Perfroms a register-relative store.

Parameters

in	*SR1*	The source register (holds the data to be stored).
in	*baseR*	A register whose value acts as a base address.
in	*address*	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	*code*	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.
Stardard 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

in	*filename*	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 fucntion 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

`in`	*verbose*	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 inteface are meant to mimic the functionality of the Wi-11 machine, allowing for simplified execution of the instructions therein. As the size of a "word" depends on the architecture, classes implementing this interface should define the word length to be 16 bits in length.

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

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

Postcondition

"value" is not changed.

Returns

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

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

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

7.9.2.6 virtual bool iWord::FromStr (const std::string & *str*) [pure virtual]

"From String"

Parameters

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

Postcondition

"str" is not changed.

Returns

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

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

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

7.9.2.7 virtual bool iWord::FromHex (const std::string & *str*) [pure virtual]

"From Hexadecimal"

Parameters

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

Postcondition

"str" is not changed.

Returns

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

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

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

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

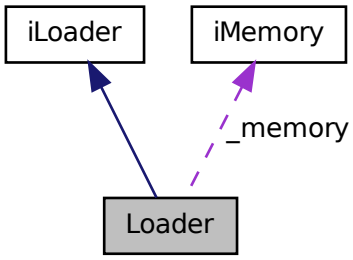
The index must be less than the size of a word, ie. 16.

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

in	*mem*	The address where memory is located.

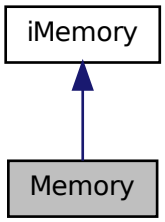
Note

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.

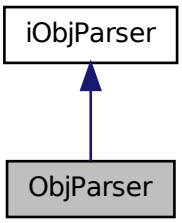
- Header Record (type = 'H')
  - data.size() = 3
    - data[0] = [Segment Name]
    - data[1] = [Initial Load Address (as a hex string)]
    - data[2] = [Segment Length (as a hex string)]

- 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

name	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

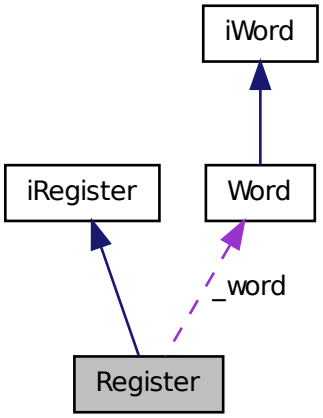
A well-formed ObjectData struct if a valid line is received, a 'dummy' ObjectData struct otherwise.

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 **operator+** (const iRegister &r) const
- Register & **operator++** ()
- Register & **operator++** (int)
- Register **operator-** (const iRegister &r) const
- Register & **operator=** (const iWord &w)
- void **Or** (const iWord &w)
- Register **Or** (const iRegister &r) const
- **Register** (const iWord &w)

- void **Store** (const iRegister &r)
- void **Store** (const iWord &w)
- 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

in	*result*	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]

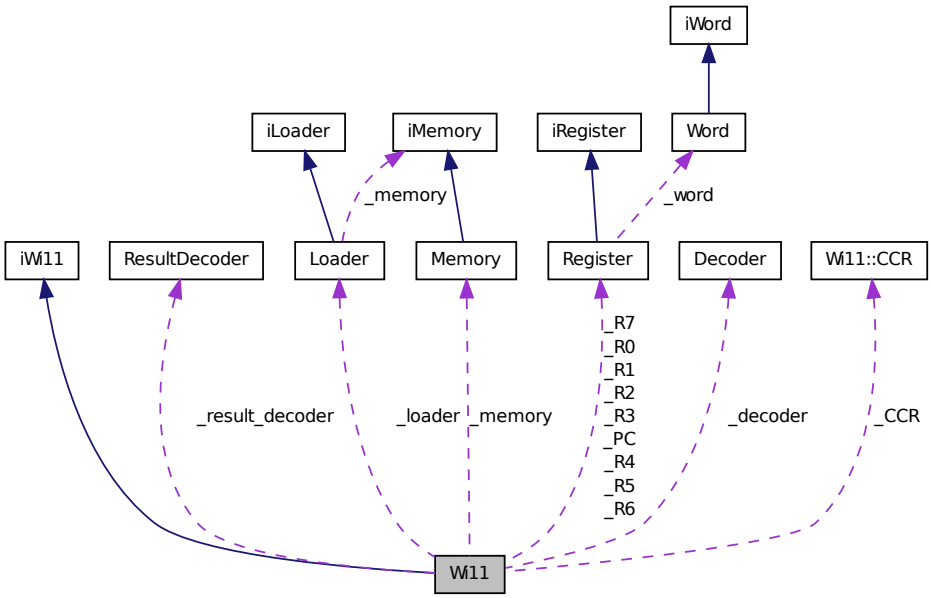
Maps a result code to, in every case but SUCCESS, an error message.

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

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 **_LoadI** (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**

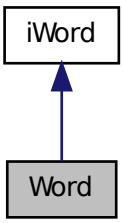
7.17.1 Detailed Description

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

i	The index of the digit desired from the binary representation of _word.

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.

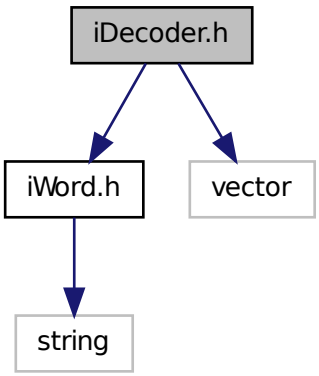
Chapter 8

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.

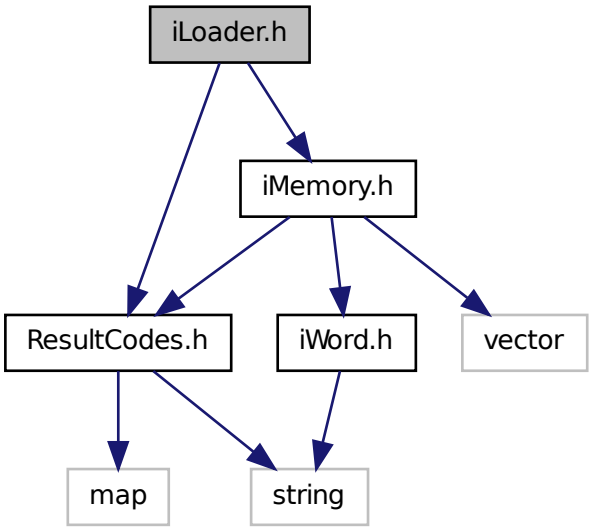
Author

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.

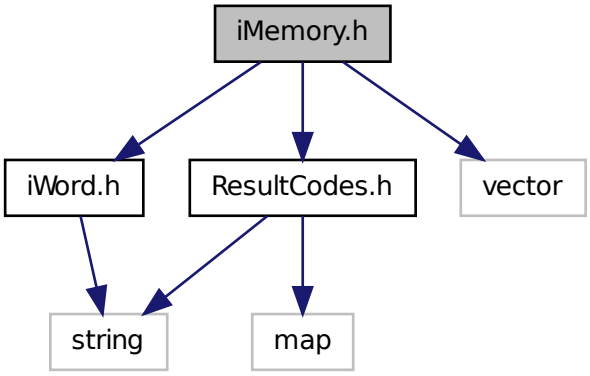
Author

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.

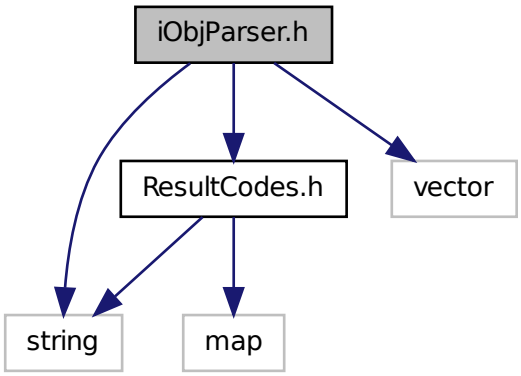
Author

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.

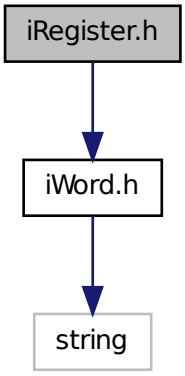
Author

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.

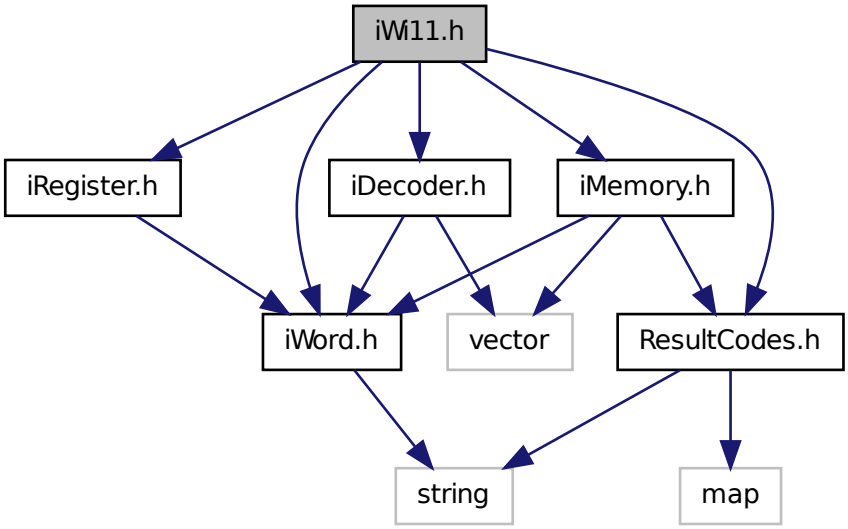
Author

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.

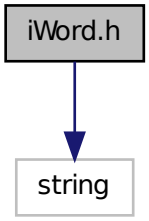
Author

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

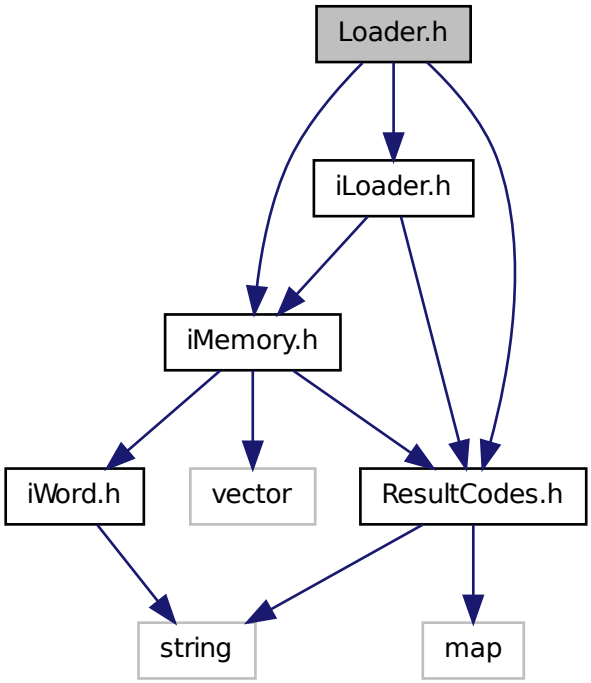
Joshua Green
Andrew Groot

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

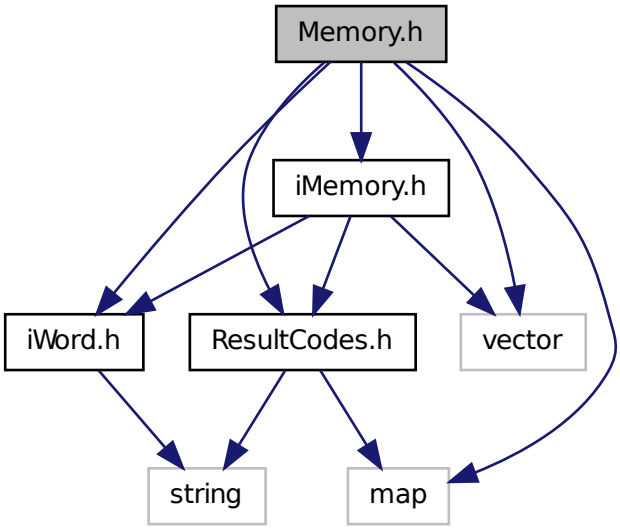
Logan Coulson

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.

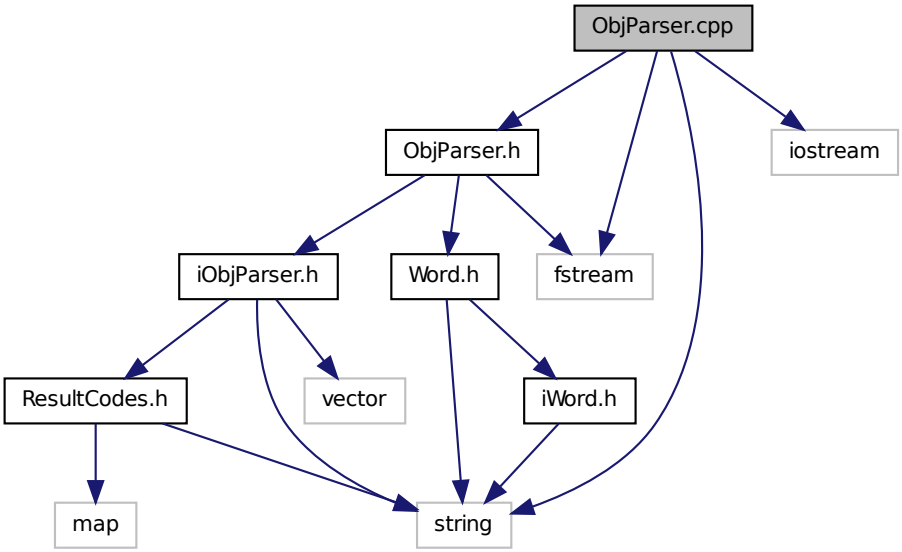
Author

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".

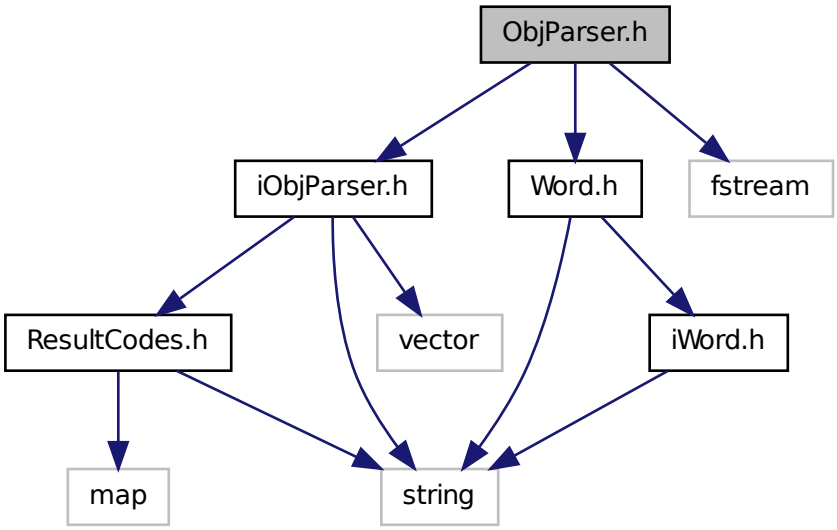
Author

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.

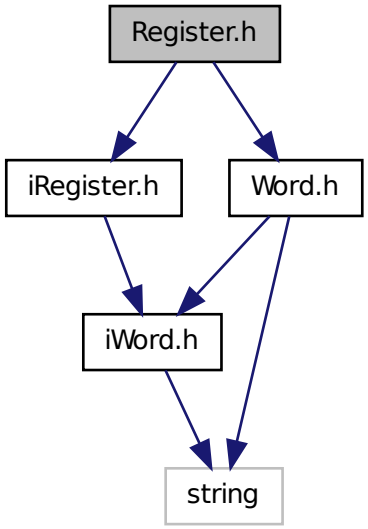
Author

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.

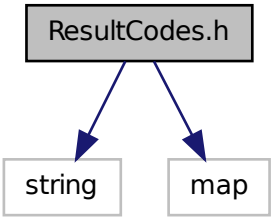
Author

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_HEADER_ENTRY, **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.

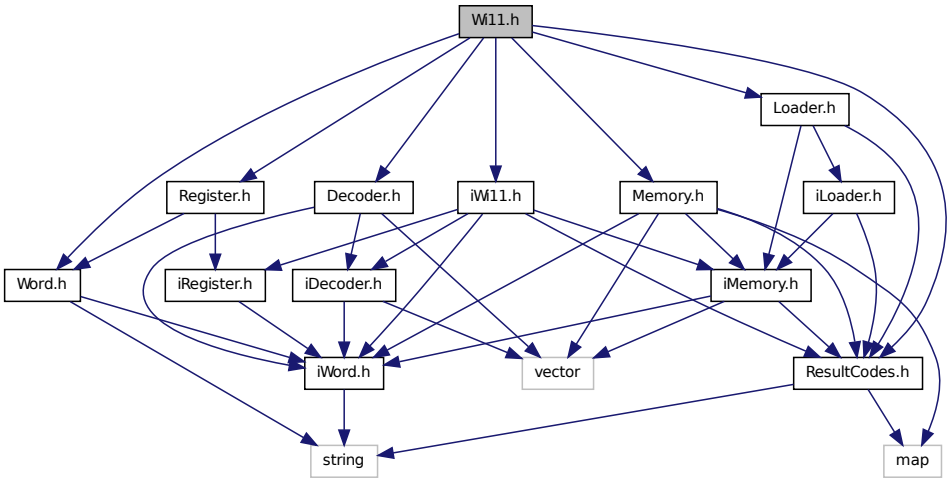
Author

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.

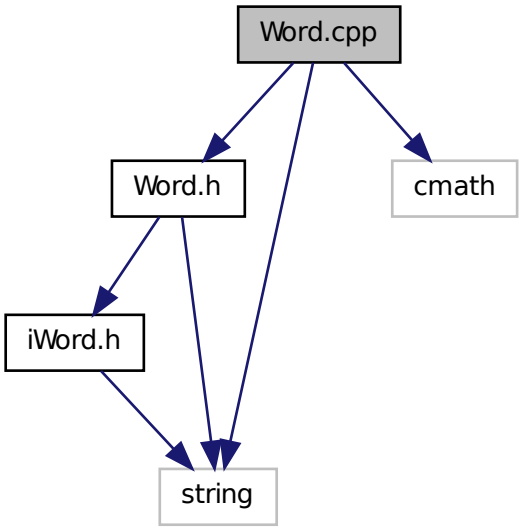
Author

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".

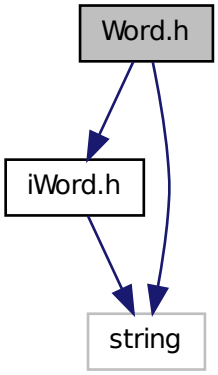
Author

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