OneUp Wi-11 Simulator: Programmer's Guide

Generated by Doxygen 1.7.3

Thu Mar 10 2011 00:01:19

Contents

1 Introduction **1**

1.1 Introduction 1

1.2 Assembler Input 2

1.2.1 Format 2

1.2.1.1 Labels 2

1.2.1.2 Whitespace 2

1.2.1.3 Operations 2

1.2.1.4 Operands 3

1.2.1.5 Comments 3

1.2.1.6 Sample Input 3

1.2.2 Instructions 4

1.2.2.1 .ORIG 4

1.2.2.2 .MAIN 5

1.2.2.3 .END 5

1.2.2.4 .EQU 5

1.2.2.5 .FILL 5

1.2.2.6 .STRZ 5

1.2.2.7 .BLKW 6

1.2.2.8 .EXT 6

1.2.2.9 .ENT 6

1.2.2.10 ADD 6

1.2.2.11 AND 6

1.2.2.12 BRNZP 6

1.2.2.13 DBUG 7

1.2.2.14 JSR 7

1.2.2.15 JMP 7

1.2.2.16 JSRR 7

1.2.2.17 JMPR 7

1.2.2.18 LD 7

1.2.2.19 LDI 7

1.2.2.20 LDR 8

1.2.2.21 NOT 8

1.2.2.22 RET 8

1.2.2.23 ST 8

1.2.2.24 STI 8

1.2.2.25 STR 8
1.2.2.26 TRAP 9
1.3 Output 9
1.3.1 Object File 9
1.3.2 Object File 9
1.3.2.1 Example 10
1.3.3 Errors 10
1.4 Interaction 10
1.4.1 Main 10
1.4.1.1 Commandline Flags 11
1.4.2 Assembler 11
1.4.3 Extractor 11
1.4.4 FileArray 12
1.4.5 itos 12
1.4.6 Line 12
1.4.7 Linker 12
1.4.8 Object Parser 12
1.4.9 Table 12
1.4.10 Printer 12
1.4.11 Word 13
1.5 Coding Conventions 13
1.6 Dependency Diagrams 13
1.7 Object Files 14
1.7.1 The Header Record 14
1.7.2 Text Records 14
1.7.3 .END 15
1.8 Interaction 15
1.8.1 Components 15
1.8.1.1 Loading 16
1.8.1.2 Executing 16
1.8.2 Instructions 16
1.8.2.1 Offsets 17
1.8.2.2 Indexes 17
1.8.2.3 Intructions 17
1.8.2.4 Traps 19

2 Namespace Index **21**
2.1 Namespace List 21

3 Class Index **23**
3.1 Class Hierarchy 23

4 Class Index **25**
4.1 Class List 25

5 File Index **27**

5.1 File List 27

6 Namespace Documentation 29

6.1 Codes Namespace Reference 29
6.1.1 Detailed Description 30
6.2 Decoder_Directory Namespace Reference 30
6.2.1 Detailed Description 31

7 Class Documentation 33

7.1 Codes::RESULT Struct Reference 33
7.1.1 Detailed Description 33
7.2 Decoder Class Reference 33
7.2.1 Detailed Description 34
7.3 Extractor Class Reference 34
7.3.1 Detailed Description 35
7.3.2 Member Function Documentation 35
7.3.2.1 _LineNumber 35
7.4 FileArray Class Reference 36
7.4.1 Detailed Description 37
7.5 iDecoder Class Reference 37
7.5.1 Detailed Description 37
7.5.2 Member Function Documentation 37
7.5.2.1 DecodeInstruction 37
7.6 iExtractor Class Reference 37
7.6.1 Detailed Description 38
7.6.2 Member Function Documentation 38
7.6.2.1 Open 38
7.6.2.2 GetSymbols 38
7.6.2.3 GetLength 39
7.7 iFileArray Class Reference 39
7.7.1 Detailed Description 39
7.7.2 Member Function Documentation 40
7.7.2.1 Add 40
7.7.2.2 Name 40
7.7.2.3 operator[] 40
7.7.2.4 Size 41
7.7.2.5 Clear 41
7.8 iLine Class Reference 41
7.8.1 Detailed Description 41
7.8.2 Member Function Documentation 42
7.8.2.1 ReadLine 42
7.8.2.2 Label 42
7.8.2.3 Literal 42
7.8.2.4 Instruction 42
7.8.2.5 operator[] 43
7.8.2.6 Size 43

7.8.2.7 ToString 43
7.8.2.8 HasLabel 43
7.8.2.9 IsPseudoOp 43
7.8.2.10 HasLiteral 44
7.8.2.11 IsComment 44
7.9 iLoader Class Reference 44
7.9.1 Detailed Description 44
7.9.2 Member Function Documentation 44
7.9.2.1 Load 44
7.10 iMemory Class Reference 45
7.10.1 Detailed Description 45
7.10.2 Member Function Documentation 46
7.10.2.1 Reserve 46
7.10.2.2 Load 46
7.10.2.3 Store 46
7.10.2.4 GetUsedMemory 47
7.11 Instruction Struct Reference 47
7.11.1 Detailed Description 47
7.11.2 Member Data Documentation 47
7.11.2.1 data 47
7.12 iObjParser Class Reference 48
7.12.1 Detailed Description 49
7.12.2 Member Function Documentation 49
7.12.2.1 Initialize 49
7.12.2.2 GetNext 49
7.13 iPrinter Class Reference 50
7.13.1 Detailed Description 50
7.13.2 Member Function Documentation 50
7.13.2.1 Open 50
7.13.2.2 Print 51
7.14 iRegister Class Reference 51
7.14.1 Detailed Description 52
7.14.2 Member Function Documentation 52
7.14.2.1 GetValue 52
7.14.2.2 Add 53
7.14.2.3 Add 53
7.14.2.4 operator+ 53
7.14.2.5 Subtract 54
7.14.2.6 Subtract 54
7.14.2.7 operator- 54
7.14.2.8 And 54
7.14.2.9 And 55
7.14.2.10 Or 55
7.14.2.11 Or 55
7.14.2.12 Not 56
7.14.2.13 Store 56

7.14.2.14 Store 56
7.14.2.15 operator= 57
7.14.2.16 operator++ 57
7.14.2.17 operator++ 57
7.15 iSymbolTable Class Reference 57
7.15.1 Detailed Description 58
7.15.2 Member Function Documentation 58
7.15.2.1 InsertLabel 58
7.15.2.2 InsertLiteral 59
7.15.2.3 AddExternal 59
7.15.2.4 IsSymbol 59
7.15.2.5 IsExternal 59
7.15.2.6 GetLabelAddr 59
7.15.2.7 GetLiteralAddr 60
7.15.2.8 IsRelocatable 60
7.15.2.9 GetLiterals 60
7.16 iWi11 Class Reference 61
7.16.1 Detailed Description 63
7.16.2 Member Function Documentation 63
7.16.2.1 _GetRegister 63
7.16.2.2 _Add 64
7.16.2.3 _Add 64
7.16.2.4 _And 65
7.16.2.5 _And 65
7.16.2.6 _Branch 66
7.16.2.7 _Debug 66
7.16.2.8 _JSR 66
7.16.2.9 _JSRR 67
7.16.2.10 _Load 67
7.16.2.11 _LoadI 68
7.16.2.12 _LoadR 68
7.16.2.13 _Not 69
7.16.2.14 _Ret 69
7.16.2.15 _Store 70
7.16.2.16 _STI 70
7.16.2.17 _STR 70
7.16.2.18 _Trap 71
7.16.2.19 LoadObj 72
7.16.2.20 DisplayMemory 72
7.16.2.21 DisplayRegisters 72
7.16.2.22 ExecuteNext 73
7.17 iWord Class Reference 73
7.17.1 Detailed Description 75
7.17.2 Member Function Documentation 75
7.17.2.1 ToInt 75
7.17.2.2 ToInt2Complement 75

7.17.2.3 ToStr 76
7.17.2.4 ToHex 76
7.17.2.5 FromInt 76
7.17.2.6 FromStr 77
7.17.2.7 FromHex 77
7.17.2.8 Add 78
7.17.2.9 operator+ 78
7.17.2.10 Subtract 78
7.17.2.11 operator- 79
7.17.2.12 And 79
7.17.2.13 Or 79
7.17.2.14 Not 80
7.17.2.15 Copy 80
7.17.2.16 operator= 80
7.17.2.17 operator++ 80
7.17.2.18 operator++ 81
7.17.2.19 operator[] 81
7.17.2.20 SetBit 81
7.18 Line Class Reference 82
7.18.1 Detailed Description 84
7.18.2 Member Function Documentation 84
7.18.2.1 _IsWS 84
7.18.2.2 _GetNext 84
7.18.2.3 _CheckArgs 84
7.19 Loader Class Reference 85
7.19.1 Detailed Description 86
7.19.2 Constructor & Destructor Documentation 86
7.19.2.1 Loader 86
7.20 Memory Class Reference 86
7.20.1 Detailed Description 87
7.21 ObjectData Struct Reference 87
7.21.1 Detailed Description 88
7.22 ObjParser Class Reference 88
7.22.1 Detailed Description 89
7.22.2 Member Function Documentation 89
7.22.2.1 Initialize 89
7.22.2.2 GetNext 90
7.23 Printer Class Reference 90
7.23.1 Detailed Description 92
7.23.2 Member Function Documentation 92
7.23.2.1 _SetBits 92
7.23.2.2 _IsReg 93
7.23.2.3 _RegNum 93
7.23.2.4 _SetBits 93
7.23.2.5 _ParseWord 94
7.23.2.6 _Check9 94

7.23.2.7 _Check6 94
7.23.2.8 _Check5 95
7.23.2.9 _PreError 95
7.23.2.10 _LineListing 95
7.23.2.11 _InFileData 96
7.24 Register Class Reference 96
7.24.1 Detailed Description 97
7.25 ResultDecoder Class Reference 97
7.25.1 Detailed Description 98
7.25.2 Member Function Documentation 98
7.25.2.1 Find 98
7.25.3 Member Data Documentation 98
7.25.3.1 _codes 98
7.26 SymbolTable Class Reference 98
7.26.1 Detailed Description 100
7.27 Wi11 Class Reference 100
7.27.1 Detailed Description 103
7.27.2 Constructor & Destructor Documentation 103
7.27.2.1 Wi11 103
7.28 Wi11::CCR Struct Reference 103
7.28.1 Detailed Description 103
7.29 Word Class Reference 103
7.29.1 Detailed Description 105
7.29.2 Constructor & Destructor Documentation 105
7.29.2.1 Word 105
7.29.3 Member Function Documentation 106
7.29.3.1 _HasBit 106
7.29.3.2 ToHexAbbr 106
7.29.3.3 FromHexAbbr 106
7.29.4 Member Data Documentation 106
7.29.4.1 _value 106

8 File Documentation **109**
8.1 Decoder.h File Reference 109
8.1.1 Detailed Description 110
8.2 Extractor.h File Reference 111
8.2.1 Detailed Description 112
8.3 FileArray.h File Reference 112
8.3.1 Detailed Description 113
8.4 iDecoder.h File Reference 113
8.4.1 Detailed Description 114
8.5 iExtractor.h File Reference 114
8.5.1 Detailed Description 115
8.6 iFileArray.h File Reference 116
8.6.1 Detailed Description 117
8.7 iLine.h File Reference 117

8.7.1 Detailed Description 118
8.8 iLoader.h File Reference 118
8.8.1 Detailed Description 119
8.9 iMemory.h File Reference 119
8.9.1 Detailed Description 119
8.10 iObjParser.h File Reference 120
8.10.1 Detailed Description 120
8.11 iPrinter.h File Reference 121
8.11.1 Detailed Description 122
8.12 iRegister.h File Reference 122
8.12.1 Detailed Description 123
8.13 iSymbolTable.h File Reference 123
8.13.1 Detailed Description 124
8.14 itos.h File Reference 124
8.14.1 Detailed Description 124
8.15 iWi11.h File Reference 125
8.15.1 Detailed Description 126
8.16 iWord.h File Reference 126
8.16.1 Detailed Description 126
8.17 Line.h File Reference 127
8.17.1 Detailed Description 128
8.18 Loader.h File Reference 128
8.18.1 Detailed Description 129
8.19 Memory.h File Reference 129
8.19.1 Detailed Description 130
8.20 ObjParser.cpp File Reference 131
8.20.1 Detailed Description 131
8.21 ObjParser.h File Reference 131
8.21.1 Detailed Description 132
8.22 Printer.h File Reference 133
8.22.1 Detailed Description 133
8.23 Register.h File Reference 134
8.23.1 Detailed Description 134
8.24 ResultCodes.h File Reference 135
8.24.1 Detailed Description 136
8.25 SymbolTable.h File Reference 136
8.25.1 Detailed Description 137
8.26 Wi11.h File Reference 138
8.26.1 Detailed Description 138
8.27 Word.h File Reference 139
8.27.1 Detailed Description 139

Chapter 1

Introduction

Primary Authors: Joshua Green

Logan Coulson

Andrew Groot

Edited by: Andrew Canale

1.1 Introduction

The "Wi-11 Machine" is a simulated 16-bit computer architecture. It has 8 general purpose registers, 3 condition code registers (CCRs), and a program counter (PC). The Wi-11 Simulator emulates the execution of the Wi-11 processor and memory components. However, programming for the Wi-11 Simulator can be difficult as it only accepts encoded object files as input. To better enable the user to more easily produce executable code, this package includes an assembler that translates assembly language into object files usable by the Wi-11 Simulator. Additionally, a linker has been added to this package to further enable the user. The linker provides a lateral level of functionality, combining multiple source files into one cohesive object file. Before alterations are made to the programming of this machine simulator package, a basic understanding of the coded environment is required. In particular, it is necessary to understand the formats of the assembly source files as well as the produced object file formats. It is also wise to be aware how the components interact with one another, and be aware of the programming conventions used.

1.2 Assembler Input

1.2.1 Format

The input file provided to the assembler has a limitation on the maximum size of the symbol table, the .o file, and the literal table. When the user runs the program, he/she may add a -s# to the end of their command; this overrides the default value of 1000 symbols. The maximum number of literals allowed is one half the maximum number of symbols specified by the user. Additionally, the number of records (the effective memory footprint (excluding literals)) within the object file is limited to twice the number of symbols specified by the user. Therefore, with the default value, the maximum number of source records is is two thousand, one thousand symbols, and five hundred literals. Each line of the input file must conform to the following structural organization:

1.2.1.1 Labels

With the exception of .ORIG and EQU, labels are optional per line. Labels may only be defined from the beginning of a line until the first whitespace. Labels must not begin with an uppercase 'R', nor begin with a lowercase 'x', as these indicate a register or a hexadecimal number respectively. Labels are case-sensitive, and may contain only alphanumeric ASCII characters. The first character in a label must be alphabetic. If a label is defined, it can be used as a reference in place of any Operand in Operation/Pseudo-Operation calls, except within operands in the same line on which the label is defined. Additionally, if a label is being used to replace a register operand, it must be a number zero through seven. Labels start on the leftmost position of the text file, and continue until a whitespace is encountered. When a symbol is used as the last argument for ADD or AND instructions, it is always interpreted as an imm5 operand rather than a source register.

1.2.1.2 Whitespace

Whitespace is any number of spaces and tabs in a row. Outside of comments, whitespace must exist before an operation (but after a label, if present), between an operation and its operands, and after the last operand to the end of the line or the start of a comment.

1.2.1.3 Operations

The operation field must be preceded on the same line by whitespace, or a label followed by whitespace. Operation and pseudo-operation names must be uppercase letters. Pseudo-operations (for use only by the compiler) may take the place of an operation, and are specified by beginning with a period. A list of all operations and pseudo-operations can be found in the ‘Instructions’ section of this document. There must be whitespace between the operation field, and its operands.

1.2.1.4 Operands

Operands continue to whitespace/end of line, or a semicolon. The types and numbers of operands are restricted by the specific operation or pseudo-operation invoked. To see these limitations, refer to section 1.2.2. A comma must be present between any two operands. No whitespace is permitted between or within operands. This assembler supports the following operands:

Registers

- Registers are specified by an uppercase ‘R’, followed by the number of the specific register. For example, register one is written as ‘R1’. The registers available to the user range from R0 – R7.

Constants

- Constants must be either decimal or hexadecimal. Decimal constants used as an operand must be immediately preceded by a pound sign ‘::’. Hexadecimal constants are preceded by a lowercase ‘x’. (i.e. Decimal 10 is written as #10. Hexadecimal 2C is written as x2C). An imm5 operand must be in the range #-16..#15, or x0..x1F. An address operand must be in the range #0..#65535, or x0..xFFFF. Note that only the least significant 9 bits of this value are used in the machine code encoding. An index6 operand must be in the range #0...#63, or x0...x3F. A trapvect8 operand must be in the range #0..#255, or x0..xFF.

1.2.1.5 Comments

Comments start with a semi-colon ‘;’, and may only come after code present on the same line, if any code is present on the line.There must be whitespace between the operand and any comments. Alternatively, a comment may be present on a line by itself. If a comment spans multiple lines, a semicolon must be present at the beginning of every line.

1.2.1.6 Sample Input

```
; Should execute properly
; Example Program
Lab2EG   .ORIG   x30B0
count    .FILL   #4
Begin    LD      ACC,count        ;R1 <- 4
         LEA     R0,msg
```

```
loop      TRAP    x22             ;print "hi! "
          ADD     ACC,ACC,#-1     ;R1--
          BRP     loop
          JMP     Next
msg       .STRZ   "hi! "
Next      AND     R0,R0,x0        ;R0 <- 0
          NOT     R0,R0           ;R0 <- xFFFF
          ST      R0,Array        ;M[Array] <- xFFFF
          LEA     R5,Array
          LD      R6,=#100        ;R6 <= #100
          STR     R0,R5,#1        ;M[Array+1] <= xFFFF
          TRAP    x25
ACC       .EQU    #1
; ----- Scratch Space -----
Array     .BLKW   #3
          .FILL   x10
          .END    Begin
```

1.2.2 Instructions

This assembler supports many different functions and a healthy understanding of the different functions is advisable. DR (Destination Register) is the location where the final value is stored. SR (Source Register) is the unchanged register containing the value used by the operation. imm5 (Immediate) is 5 bits, and is sign extended to 16 bits when used. pgoffset9 (Page Offset Nine) is used to form the last 7 bits for memory access. The first 7 bits originate from the PC (Program Counter), the remaining 9 bits are concatenated by the pgoffset9. index6 (Index Six) is used as a six bit number that is added to a register(BaseR) to determine an offset. In addition to the operations provided by the simulator, there are several pseudo ops that the assembler provides. A pseudo op is recognizable by the period '.' at the beginning of the command. Pseudo ops are not always directly translated to the register transfer language, but may often be directives to the linker or assembler.

1.2.2.1 .ORIG

.ORIG x0-xFFFF

Excluding .MAIN, .ORIG must be the first non-comment record in the source program. The optional operand indicates the absolute address the program is to be stored in. If the operand is absent, the program may be "Relocatable", that is may be stored anywhere (determined at load time). If there is no operand, the entire program must fit in one page of memory (512 16-bit words). It is also required to have a 6 character label, which is used to generate the header record in the .o file. Additionally, the provided label is automatically declared as if it were declared as an Entry Point (via .ENT).

1.2.2.2 .MAIN

.MAIN .MAIN has no arguments and must be placed before the .ORIG statement. This pseudo op declares the current source file as the only starting point of execution and causes all other linked files' .END values to be ignored. A file with .MAIN does not affect its ability to be assembled stand-alone. If multiple linked files contain the .MAIN pseudo op, then the linker will report this condition as an error; remove all but one instance of .MAIN from all linked object files before linking. If the linker is given multiple source files and is unable to find an instance of .MAIN then the first source file's .END record is used as the start of execution.

1.2.2.3 .END

.END x0-xFFFF

This operation indicates the end of the input program. The operand is optional, and it indicates where the program will start executing. If the operand is not present, it means that execution begins at the first address in the segment, and the assembler assumes an address of x0. There must be a .END command in every program.

1.2.2.4 .EQU

.EQU

.EQU equates the label to the operand, creating a constant. This constant can be any previously defined symbol or constant. .EQU requires a label and an operand to be present.

1.2.2.5 .FILL

.FILL #-32768-32768 or x0-xFFFF

.FILL defines a one word quantity, whose contents is the value of the operand. This operand can be in hexadecimal, decimal, or a label. If a label is used, its value must be within acceptable ranges (i.e. #-32768..xFFFF). This value is stored by the assembler in the word of memory that the .FILL occupies. Additionally, the assembler location counter is moved forward one word.

1.2.2.6 .STRZ

.STRZ "a string with escaped \"'s"

.STRZ defines a null-terminated block of words to hold a string of the characters in the operand field (in this case, the next 6 memory locations would hold the ASCII for "words\0"). ASCII representations are used to store these characters in memory.

1.2.2.7 .BLKW

.BLKW x1-xFFFF

.BLKW creates a block of storage. It occupies a number words of memory as indicated by the operand. The block of storage is not initialized.

1.2.2.8 .EXT

.EXT

.EXT declares a single, or a list (separated by commas), of symbols. Symbols declared by .EXT must be defined via .ENT, or via .ORIG, by a separate object file; using .EXT renders the current object unable to be assembled stand-alone and must be linked with the file containing the symbol definitions via the Linker.

1.2.2.9 .ENT

.ENT

.ENT declares a single symbol name as for use outside the scope of the current object file. However, .ENT symbols may also be used within the current object file. Declaring a symbol via .ENT does not affect its ability to be assembled stand-alone.

1.2.2.10 ADD

ADD DR SR1 SR2 or ADD DR SR1 imm5

ADD takes the contents of SR1 and SR2 or imm5, adds them, and places the result in DR.

1.2.2.11 AND

AND DR SR1 SR2 or AND DR SR1 imm5

AND performs a logical ‘and’ between SR1 and either SR2 or imm5, and places the result in DR.

1.2.2.12 BRNZP

BRNZP pgoffset9

BRNZP checks to see if the CCR matches the NZP component of the branch command. If it does, it sets the PC to be a concatenation of bits [15:9] of the PC with bits [8:0] (pgoffset9) of the instruction. Any combination of N’s, Z’s, and P’s are acceptable.

1.2.2.13 DBUG

DBUG

DBUG prints out the contents of all of the machine registers to the console.

1.2.2.14 JSR

JSR pgoffset9

JSR sets the PC to be a concatenation of bits [15:9] of the PC with bits [8:0] (pgoffset9) of the instruction. JSR also stores the current PC into R7, so that the RET instruction can restore the PC and cause execution to jump back to the instruction following the JSR.

1.2.2.15 JMP

JMP pgoffset9

JMP behaves the same as JSR, except that it does not store the current program counter into R7.

1.2.2.16 JSRR

JSRR BaseR index6

JSRR sets the PC equal to the contents of memory at location BaseR+index6. The original PC is stored in R7.

1.2.2.17 JMPR

JMPR BaseR index6

JMPR behaves the same as JSRR, except that it does not store the PC in R7.

1.2.2.18 LD

LD DR pgoffset9

LD loads DR with the contents of memory at the concatenation of [15:9] of the PC with bits [8:0] (pgoffset9) of the instruction.

1.2.2.19 LDI

LDI DR pgoffset9

LD loads DR with the contents of the memory address in the contents of memory at the address indicated by concatenation of [15:9] of the PC with bits [8:0] (pgoffset9) of the instruction.

1.2.2.20 LDR

LDR DR BaseR pgoffset9

DR is loaded with the operand at the address that is formed by adding the zero-extended six bit offset (index6) to the specified base register (BaseR). Thus, the index is always interpreted as a positive quantity (in the range #0-#63).

1.2.2.21 NOT

NOT DR SR

NOT inverts the bits of the SR and stores the results in DR.

1.2.2.22 RET

RET

RET copies the contents of R7 to the PC.

1.2.2.23 ST

ST SR pgoffset9

ST stores the contents of SR in memory at the concatenation of [15:9] of the PC with bits [8:0] (pgoffset9) of the instruction.

1.2.2.24 STI

STI SR pgoffset9

STI stores SR in the memory address in the contents of memory at the address indicated by concatenation of [15:9] of the PC with bits [8:0] (pgoffset9) of the instruction.

1.2.2.25 STR

STR SR BaseR index6

SR is saved at the address that is formed by adding the zero-extended six bit offset (index6) to the specified base register (BaseR). Thus, the index is always interpreted as a positive quantity (in the range 0-63).

1.2.2.26 TRAP

TRAP trapvect8

Depending on the trapvect8, trap does one of the following:

- x21 out Write the character in R0[7:0] to the console.
- x22 puts Write the null-terminated string pointed to by R0 to the console.
- x23 in Print a prompt on the screen and read a single character from the keyboard. The character is copied to the screen and its ASCII code is copied to R0. The high 8 bits of R0 are cleared.
- x25 halt Halt execution and print a message to the console.
- x31 outn Write the value of R0 to the console as a decimal integer.
- x33 inn Print a prompt on the screen and read a decimal number from the keyboard. The number is echoed to the screen and stored in R0. One may place specific requirements on the size, formatting, etc. of the input.
- x43 rnd Store a random number in R0.

1.3 Output

The assembler has two distinct outputs. The first outputs an object file usable by a simulator of the Wi-11 machine, and the second is a human readable listing for the user. The assembler will also display error codes on the console if necessary.

1.3.1 Object File

The object file consists of a header record, text records, and an end record as described in the Programmers guide for the OneUp Wi-11 simulator. These records are generated by the assembler from the program taken in as input.

1.3.2 Object File

The second output by the assembler is a more human readable output. It contains the source program, as well as its assembly. The current format for this data is (Address in hexadecimal) contents hexadecimal contents binary (line number) Label instruction operands. This allows for a easy to read output that has all of the necessary information in it.

1.3.2.1 Example

```
                              (   2) Lab2EG     .ORIG   x30B0
(30B0) 0004  0000000000000100 (   3) count      .FILL   #4
(30B1) 22B0  0010001010110000 (   4) Begin      LD      ACC,count        ;R1 <- 4
(30B2) E0B7  1110000010110111 (   5)            LEA     R0,msg
(30B3) F022  1111000000100010 (   6) loop       TRAP    x22              ;print "hi!
(30B4) 127F  0001001001111111 (   7)            ADD     ACC,ACC,#-1      ;R1--
(30B5) 02B3  0000001010110011 (   8)            BRP     loop
(30B6) 40BC  0100000010111100 (   9)            JMP     Next
(30B7) 0068  0000000001101000 (  10) msg                .STRZ   "hi! "
(30B8) 0069  0000000001101001 (  10)
(30B9) 0021  0000000000100001 (  10)
(30BA) 0020  0000000000100000 (  10)
(30BB) 0000  0000000000000000 (  10)
(30BC) 5020  0101000000100000 (  11) Next       AND     R0,R0,x0       ;R0 <- 0
(30BD) 9000  1001000000000000 (  12)            NOT     R0,R0          ;R0 <- xFFFF
(30BE) 30C3  0011000011000011 (  13)            ST      R0,Array       ;M[Array] <- x
(30BF) EAC3  1110101011000011 (  14)            LEA     R5,Array
(30C0) 2CC7  0010110011000111 (  15)            LD      R6,=#100       ;R6 <= #100
(30C1) 7141  0111000101000001 (  16)            STR     R0,R5,#1       ;M[Array+1] <=
(30C2) F025  1111000000100101 (  17)            TRAP    x25
                              (  18) ACC                .EQU    #1
(30C3)                        (  20) Array      .BLKW   #3
(30C6) 0010  0000000000010000 (  21)            .FILL   x10
(30C7) 0064  0000000001100100 ( lit) <100>
                              (  22)            .END    Begin
```

1.3.3 Errors

The Assembler can also output errors as included in the ResultCodes.h file. Additional errors can be added if needed, but adding superfluous errors should generally be avoided. Any error message is always returned, and the caller function either evaluates that result, or reports an error state.

1.4 Interaction

The interaction portion of this document describes how each function works, and what other functions that function calls.

1.4.1 Main

The main function parses the command line arguments and ensures their validity. Additionally, the main function handles integration of the three core components of the simulator package: assembler, linker, and simulator. It is possible to break execution after each phase as well as begin execution at later phases.

1.4.1.1 Commandline Flags

- Assembler, Linker, and Simulator Shared Flags:

  -t: Predefine the trap code labels -- out, puts, in, halt, outn, inn, & rnd. Optional.

  -s#: Increase the symbol limit -- defaults to 1000. Optional.

  -v: Verbose: Print a description of what is being done during assembly and linking. Optional.

- Assembler-Specific Flags:

  -a: Assemble "infile" into "outfile", do not link or execute.

- Linker-Specific Flags:

  -o: Assemble and link multiple files, creating a single object file named "outfile"

  -n: Specifies that the input files are already assembled.

- Simulator-Specific Flags:

  -ox: Assemble, link, and execute; create "outfile" as in the '-o' option. Either 'ox' or 'x' flags required.

  -x: Execute a pre-linked object file. Skip assembly and linking steps. Either 'ox' or 'x' flags required.

  -d: Print debug information during execution. Optional.

  -n: Specifies that the input files are already assembled. Optional.

1.4.2 Assembler

The Assembler uses the extractor class to generate the symbol tables. This assembler uses a 2-pass method. The symbol table is generated during the first pass. The second pass uses the extractor and line class. After that, it calls the result function to generate the users listing. It then uses the printer class to print the resulting .o file.

1.4.3 Extractor

The extractor uses the SymbolTable class to create a symbol table for the labels and literals as it parses through the input program. It also uses the line class to generate the rest of the .o file as it goes through.

1.4.4 FileArray

FileArray keeps an array of ObjectParser instances with a simple ordering property to aid the linking process. The ObjectParser designated with the .MAIN attribute is first in the array to enable the linker to use the proper start of execution address.

1.4.5 itos

itos compliments design by providing easy access to a function that converts an integer to an std::string.

1.4.6 Line

The line class is responsible for converting each line into the corresponding text record in the .o file. It uses the symbol table and word to do this.

1.4.7 Linker

The Linker combines multiple source files into one cohesive file. Linking does not modify any of the input files. The output file's .EXT and .ENT records are evaluated and defined and the produced file is ready for simulation. The produced file is does not contain any .EXT or .ENT records and therefore is impractical to attempt to re-link the produced file. The linker can support multiple relocatable source files but is unable to link a non-relocatable file to another file.

1.4.8 Object Parser

The ObjParser component handles all file input used to interface with the object files. The ObjParser component is capable of reading on object file line entry at a time and returns an ObjectData struct token. The ObjectData.type value is gaurenteed to be a null character if an invalid line entry was obtained.

1.4.9 Table

The symbol table keeps track of all of the symbols, and their values, by creating a map that maps each symbol to its value using the word class.

1.4.10 Printer

The printer is responsible for printing the output generated by the main. It uses both word and line to do this.

1.4.11 Word

Word is used to keep track of memory values, and to convert between hexadecimal, binary, and decimal.

1.5 Coding Conventions

The following is a list of coding conventions to follow while maintaining this project.

- Use comments to ensure that what you are doing is understandable.
- Specifications are stored as low in the dependency tree as possible.
- All interface .h files start with an i.
- Do not use namespaces in header files, as they will apply to oter components as well, which could cause problems.
- The first letter in each word in a method name is capitalized. All other letters in the name are lowercase
- All constants names are all-caps.
- The header files define the classes and functions, the .cpp files execute them.
- Private functions begin with a '_' character.
- Always use braces while using if, while, and for commands, even if they are not needed for what you were using it for.

1.6 Dependency Diagrams

In the pages to come, there are several dependancy diagrams that display the interactions of files and components. This section serves as a key for use in understanding their conventions.

- A **white** box indicates a class. A **marker** in the lower right corner of the box indicates that the class has base classes that are hidden. If the box has a **dashed** border this indicates virtual inheritance.
- A **solid** arrow indicates public inheritance.
- A **dashed** arrow indicates protected inheritance.
- A **dotted** arrow indicates private inheritance.

1.7 Object Files

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

- One Header Record
- Zero or more Text Records
- One End Record

1.7.1 The Header Record

The Header Record is a single line that prepares the system for 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.

1.7.2 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.

1.7.3 .END

The End Record is, as the name would suggest, the last line of the file. 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.

1.8 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.

1.8.1 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 to 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.

1.8.1.1 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.

1.8.1.2 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.

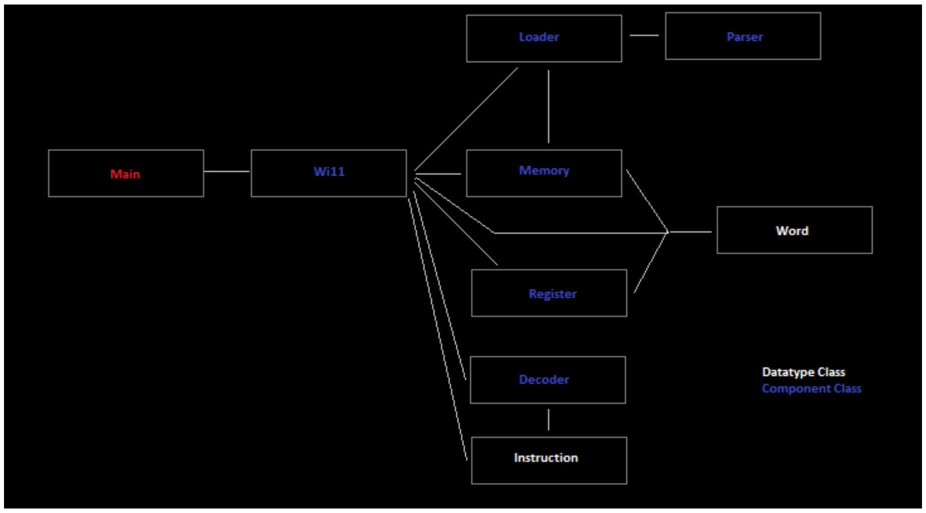


Figure 1.1: This diagram shows the awareness of each component with those operating below it.

1.8.2 Instructions

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 delegates purpose to the following 12 bits.

1.8.2.1 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").

1.8.2.2 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.

1.8.2.3 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

1.8.2.4 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:

Codes (Values corresponding to the results of Wi-11 function calls) 29
Decoder_Directory (Declares register id's and instruction types for each register and instruction) 30

Chapter 3

Class Index

3.1 Class Hierarchy

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

Codes::RESULT 33
Decoder 33
iDecoder 37
iExtractor 37
    Extractor 34
iFileArray 39
    FileArray 36
iLine 41
    Line 82
iLoader 44
    Loader 85
iMemory 45
    Memory 86
Instruction 47
iObjParser 48
    ObjParser 88
iPrinter 50
    Printer 90
iRegister 51
    Register 96
iSymbolTable 57
    SymbolTable 98

iWi11 61
    Wi11 100
iWord 73
    Word 103
ObjectData 87
ResultDecoder 97
Wi11::CCR 103

Chapter 4

Class Index

4.1 Class List

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

Codes::RESULT (Holds error-reporting information) 33
Decoder (Implements iDecoder) 33
Extractor (Implements the iExtractor interface) 34
FileArray (Implements iFileArray) 36
iDecoder (Defines how Wi-11 instructions are decoded) 37
iExtractor (Extracts symbols from a source file to be used on a second read) . 37
iFileArray (Keeps an array of files in with a simple ordering property to aid the linking process) 39
iLine (Stores information about a Line of Wi-11 assembly code) 41
iLoader (Defines how the Wi-11 initializes memory) 44
iMemory (Defines the functionality of memory in the Wi-11 machine) 45
Instruction (Container to simplify interactions with Wi-11 instructions) . . . 47
iObjParser (Defines how object files are processed) 48
iPrinter (Writes an object file and prints a listing to standard out) 50
iRegister (Defines a "register" in the Wi-11 machine) 51
iSymbolTable (Stores symbols and literals extracted from a source file) . . . 57
iWi11 (Defines the internal logic of the Wi-11) 61
iWord (Defines a "word" of data on the Wi-11 Machine) 73
Line (Implements the iLine interface) 82
Loader (Implements iLoader) 85
Memory (Implements iMemory) 86
ObjectData (A simple encoding of a "record") 87
ObjParser (Implements iObjParser) 88
Printer (Implements the iPrinter interface) 90
Register (Implements iRegister) 96

ResultDecoder (Finds the messages associated with a given result code) . . . 97
SymbolTable (Implements the iSymbolTable interface) 98
Wi11 (Implements iWi11) 100
Wi11::CCR (Condition code registers: negative, zero, positive) 103
Word (Implements iWord) 103

Chapter 5

File Index

5.1 File List

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

Decoder.h (Definition of the private data for the "Decoder" class. (none)) . . 109
Extractor.h (Definition of the private data for the "Extractor" class) 111
FileArray.h (Definition of the private data for the "FileArray" class) 112
iDecoder.h (Definition of the Wi-11 instruction decoder) 113
iExtractor.h (Definition of the Extractor for the Wi-11 assembler) 114
iFileArray.h (Definition of a file-handling system for the Wi-11 linker) 116
iLine.h (Definition of a "Line" of Wi-11 assembly code) 117
iLoader.h (Definition of the Wi-11 program loader) 118
iMemory.h (Definition of Wi-11 memory) 119
iObjParser.h (Definition of the Object File Parser) 120
iPrinter.h (Definition of the output of the Wi-11 assembler) 121
iRegister.h (Definition of a "register" in the Wi-11 machine) 122
iSymbolTable.h (Definition of the symbol table for the Wi-11 assembler) . . 123
itos.h (Two utility functions: int to std::string (decimal and hex)) 124
iWi11.h (Definition of the Wi-11 machine simulator) 125
iWord.h (Definition of a "word" of data) 126
Line.h (Definition of the private data for the "Line" class) 127
Loader.h (Definition of the private data for the "Loader" class) 128
Memory.h (Definition of private data for the "Memory" class) 129
ObjParser.cpp (Implements the declarations in "../h/ObjParser.h") 131
ObjParser.h (Definition of private data for the "ObjParser" class) 131
Printer.h (Definition of the private data for the "Printer" class) 133
Register.h (Definition of private data for the "Register" class) 134
ResultCodes.h (Definition of the Wi-11 assemblers error messages) 135
SymbolTable.h (Definition of the private data for the "SymbolTable" class) . 136

Wi11.h (Definition of the private data for the "Wi11" class) 138
Word.h (Definition of private data for the "Word" class) 139

Chapter 6

Namespace Documentation

6.1 Codes Namespace Reference

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

Classes

- struct RESULT

  Holds error-reporting information.

Enumerations

- enum **ERROR** {

  ERROR_0, **SUCCESS**, **HALT**, **UNDEFINED**,

  INVALID_HEADER_ENTRY, **INVALID_DATA_ENTRY**, **OUT_OF_BOUNDS**, **NOT_HEX**,

  INVALID_TRAP_CODE, **INVALID_START_PC**, **REQUESTED_MEMORY_-TOO_LARGE**, **BAD_MALLOC**,

  RELOCATE_ENTRY_IN_ABSOLUTE, **RELOCATION_OVERFLOW**, **RELOCATION_-OUTSIDE_BOUNDS**, **UNRESOLVED_EXTERNAL**,

  LINK_ABS, **MULTI_MAIN**, **INV_LBL**, **LBL_WO_INST**,

  INV_INST, **STRZ_NOT_STR**, **END_OF_STR**, **STR_JUNK**,

  ARG_SIZE, **EMPTY_ARG**, **INV_REG**, **INV_CONST**,

  INV_ARG, **INV_HEX**, **INV_DEC**, **INV_BR**,

NON_LD_LIT, **ORIG**, **ORIG2**, **ORIG_LBL**,
ORIG_HEX, **REQ_LABEL**, **LBL_NOT_FOUND**, **REDEF_LBL**,
MAX_S_SIZE, **MAX_L_SIZE**, **MAX_LENGTH**, **ABS_REL**,
INV_IMM, **INV_IDX**, **PG_ERR**, **NO_END**,
END_OB, **UNEXP_EOF**, **REL_PG_SIZE**, **MEM_FIT**,
MAIN_LBL, **END_LBL**, **INV_COMMENT**, **MAIN**,
EXT_REDEF, **RELATIVE**, **FILE_NOT_FOUND**, **FILE_NOT_OPENED** }

6.1.1 Detailed Description

Values corresponding to the results of Wi-11 function calls. Contains the error and success codes for the Assembler, Linker and Simulator.

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 **REGISTER_ID** {
  R0, **R1**, **R2**, **R3**,
  R4, **R5**, **R6**, **R7**,
  PC }
- enum **INSTRUCTION_TYPE** {
  ADD, **AND**, **BRx**, **DBUG**,
  JSR, **JSRR**, **LD**, **LDI**,
  LDR, **LEA**, **NOT**, **RET**,
  ST, **STI**, **STR**, **TRAP**,
  ERROR }

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 Codes::RESULT Struct Reference

Holds error-reporting information.

Public Member Functions

- **RESULT** (ERROR err=ERROR_0, std::string inf="")
- bool **operator==** (const ERROR) const
- bool **operator!=** (const ERROR) const

Public Attributes

- std::string **info**
- ERROR **msg**

7.1.1 Detailed Description

Holds error-reporting information.

7.2 Decoder Class Reference

Implements iDecoder.

Public Member Functions

- Instruction **DecodeInstruction** (const iWord &) const

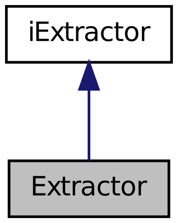
7.2.1 Detailed Description

Implements iDecoder.

7.3 Extractor Class Reference

Implements the iExtractor interface.

Collaboration diagram for Extractor:



Public Member Functions

- Extractor (int size=SYMBOL_TABLE_MAX_SIZE)

  Gets the default max number of symbols, defaults on 1000.

- ∼Extractor ()

  Closes the input file, if necessary.

- bool **Open** (std::string filename)
- Codes::RESULT **GetSymbols** (SymbolTable &symbols)
- Word **GetLength** () const

Private Member Functions

- std::string _LineNumber (int pos)

  Creates a string "Line n", with n = pos.

Private Attributes

- std::ifstream _fileStream

  The input file.

- int _length

  Keep length after SymbolTable is filled.

- int _max_size

  The maximum size of the symbol table.

7.3.1 Detailed Description

Implements the iExtractor interface. This implementation is very nearly trivial. In this file the default maximum symbol table size is declared and store the run-time value of the max will be stored in the Extractor object. The private variable keeping track of the object file length is an int to facilitate the reporting of a file whose length has exceeded the upper boundary of memory.

7.3.2 Member Function Documentation

7.3.2.1 string Extractor::_LineNumber (int *pos*) `[private]`

Creates a string "Line n", with n = pos.

Parameters

`in`	*pos*	The line number.

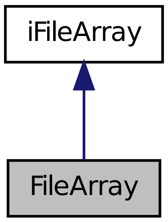
Returns

A string as described above.

7.4 FileArray Class Reference

Implements iFileArray.

Collaboration diagram for FileArray:



Public Member Functions

- FileArray ()

  Sets the _hasMain variable to false.

- Codes::RESULT **Add** (std::string name)
- void **Reset** ()
- std::string **Name** (int index) const
- ObjParser & **operator[ ]** (int index)
- int **Size** () const
- void **Clear** ()

Private Attributes

- std::vector< ObjParser ∗ > _files

  Keeps all of the files open for the two pass algorithm.

- std::vector< std::string > _names

  Keeps track of the names of the files.

- bool _hasMain

  True iff a main file has already been found.

7.4.1 Detailed Description

Implements iFileArray.

7.5 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.

7.5.1 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.

7.5.2 Member Function Documentation

7.5.2.1 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 specified in its documentation.

7.6 iExtractor Class Reference

Extracts symbols from a source file to be used on a second read.

Public Member Functions

- virtual bool Open (std::string filename)=0

  Opens the input file to be read.
- virtual Codes::RESULT GetSymbols (SymbolTable &symbols)=0

  Extracts the symbols and literals from the source file.
- virtual Word GetLength () const =0

7.6.1 Detailed Description

Extracts symbols from a source file to be used on a second read. This class provides a simple interface to fill a SymbolTable object and retrieve the size in memory this program would require.

7.6.2 Member Function Documentation

7.6.2.1 virtual bool iExtractor::Open (std::string *filename*) `[pure virtual]`

Opens the input file to be read.

Parameters

`in`	*filename*	The source file.

Returns

True iff the file was successfully opened.

7.6.2.2 virtual Codes::RESULT iExtractor::GetSymbols (SymbolTable & *symbols*) `[pure virtual]`

Extracts the symbols and literals from the source file.

Parameters

in:out]	symbols The object into which the symbols and literals will be stored.

Returns

SUCCESS iff the file syntax is sound, otherwise an appropriate error code.

This opeeration acts as the first of the two passes on the source file. Everything that can be checked on this first pass will, all other possible errors will be left until the second pass.

7.6.2.3 virtual Word iExtractor::GetLength ( ) const `[pure virtual]`

Precondition

GetSymbols() has already been successfully run.

Returns

The size of the space in memory needed to hold the source file.

7.7 iFileArray Class Reference

Keeps an array of files in with a simple ordering property to aid the linking process.

Public Member Functions

- virtual Codes::RESULT Add (std::string name)=0

  Adds a file to the array.

- virtual void Reset ()=0

  Sets all of the files in the array back to the beginning of input.

- virtual std::string Name (int index) const =0

  Returns the name of a file in the array.

- virtual ObjParser & operator[ ] (int index)=0

  Access a particular file.

- virtual int Size () const =0
- virtual void Clear ()=0

7.7.1 Detailed Description

Keeps an array of files in with a simple ordering property to aid the linking process. Using this, the linker doesn't have to worry about checking for multiple main files, nor does it have to bother opening the files or separating the lines into their parts because of the ObjectData objects returned by ObjParser's GetNext() method.

7.7.2 Member Function Documentation

7.7.2.1 virtual Codes::RESULT iFileArray::Add (std::string *name*) `[pure virtual]`

Adds a file to the array.

Parameters

`in`	*name*	The name of the file to add.

Returns

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

Normally, the file is opened and added to the back of the array. If the file has a "main" entry, it is put at the beginning of the array. If there is already another main file at the front, an error is returned.

7.7.2.2 virtual std::string iFileArray::Name (int *index*) const `[pure virtual]`

Returns the name of a file in the array.

Parameters

`in`	*index*	The index of the desired file.

Returns

The name of the file.

7.7.2.3 virtual ObjParser& iFileArray::operator[] (int *index*) `[pure virtual]`

Access a particular file.

Parameters

`in`	*index*	The index of the desired file.

Returns

A reference to the ObjParser object that opened the file.

7.7.2.4 virtual int iFileArray::Size ( ) const [pure virtual]

Returns

The number of files in the array.

7.7.2.5 virtual void iFileArray::Clear ( ) [pure virtual]

Closes and removes all the file in the array.

7.8 iLine Class Reference

Stores information about a Line of Wi-11 assembly code.

Public Member Functions

- virtual Codes::RESULT ReadLine (std::string line)=0

  Parses and tokenizes a string.

- virtual std::string Label () const =0
- virtual int Literal () const =0
- virtual std::string Instruction () const =0
- virtual std::string operator[ ] (int index) const =0

  Provides access to the arguments of the instruction.

- virtual int Size () const =0
- virtual std::string ToString () const =0
- virtual bool HasLabel () const =0
- virtual bool IsPseudoOp () const =0
- virtual bool HasLiteral () const =0
- virtual bool IsComment () const =0

7.8.1 Detailed Description

Stores information about a Line of Wi-11 assembly code. This class defines an interface for pulling relevent data from a line of assembly code without having to handle whitespace or test for various syntactic properties.

7.8.2 Member Function Documentation

7.8.2.1 virtual Codes::RESULT iLine::ReadLine (std::string *line*) `[pure virtual]`

Parses and tokenizes a string.

Parameters

`in`	*line*	The line to be parsed.

Returns

SUCCESS iff the line is valid wi-11 assembly code, otherwise an appropriate error code.

7.8.2.2 virtual std::string iLine::Label ( ) const `[pure virtual]`

Precondition

HasLabel() returns true.

Returns

The label found at the beginning of the line.

7.8.2.3 virtual int iLine::Literal ( ) const `[pure virtual]`

Precondition

HasLiteral() returns true.

Returns

The literal found in the line.

7.8.2.4 virtual std::string iLine::Instruction ( ) const `[pure virtual]`

Returns

The instruction found in the line.

7.8.2.5 virtual std::string iLine::operator[] (int *index*) const `[pure virtual]`

Provides access to the arguments of the instruction.

Parameters

`in`	*index*	The index of the argument desired, starting at 0 for the first one.

Returns

Argument number "index".

One Important Case

- .STRZ: String arugments are returned without the surrounding quotes. Only escaped quotes will appear in this string.

7.8.2.6 virtual int iLine::Size ( ) const `[pure virtual]`

Returns

The number of arguments to the instruction.

7.8.2.7 virtual std::string iLine::ToString ( ) const `[pure virtual]`

Returns

The line of code as found in the source file.

7.8.2.8 virtual bool iLine::HasLabel ( ) const `[pure virtual]`

Returns

True iff a label was found in the line.

7.8.2.9 virtual bool iLine::IsPseudoOp ( ) const `[pure virtual]`

Returns

True iff the instruction is a pseudo-op (.ORIG, .END, .EQU, etc.).

7.8.2.10 virtual bool iLine::HasLiteral ( ) const `[pure virtual]`

Returns

True iff a literal was found in the line.

7.8.2.11 virtual bool iLine::IsComment ( ) const `[pure virtual]`

Returns

True iff he line only contained a comment or whitespace.

7.9 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).

Private Member Functions

- virtual Codes::RESULT **_GetLoadAddress** (Word &produced_addr, const Word &segment_length) const =0

7.9.1 Detailed Description

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

7.9.2 Member Function Documentation

7.9.2.1 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).

7.10 iMemory Class Reference

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

Public Member Functions

- virtual Codes::RESULT Reserve (const iWord &initial_address, const iWord &length)=0

  Reserves an initial section of memory for instructions.

- virtual Word Load (const iWord &w) const =0

  Performs a load.

- virtual Codes::RESULT Store (const iWord &address, const Word &value)=0

  Peforms a store.

- virtual std::vector< std::vector< Word > > GetUsedMemory () const =0

  Details memory's current state.

7.10.1 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.

7.10.2 Member Function Documentation

7.10.2.1 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.

7.10.2.2 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.

7.10.2.3 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.

7.10.2.4 virtual std::vector< std::vector<Word> > iMemory::GetUsedMemory ( ) const [pure virtual]

Details memory's current state.

Returns

The values in memory that have been written to or reserved.

7.11 Instruction Struct Reference

Container to simplify interactions with Wi-11 instructions.

Public Attributes

- Decoder_Directory::INSTRUCTION_TYPE type

  The type of instruction.

- std::vector< Word > data

  The arguments to the operation (including unnecessary bits).

7.11.1 Detailed Description

Container to simplify interactions with Wi-11 instructions.

7.11.2 Member Data Documentation

7.11.2.1 std::vector<Word> Instruction::data

The arguments 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 Instructions 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).

7.12 iObjParser Class Reference

Defines how object files are processed.

Public Member Functions

- virtual Codes::RESULT Initialize (const char ∗filename)=0

  Attempts to open the object file.

- virtual ObjectData GetNext ()=0

  Pre-processes the next line of the object file.

7.12.1 Detailed Description

Defines how object files are processed.

7.12.2 Member Function Documentation

7.12.2.1 virtual Codes::RESULT iObjParser::Initialize (const char ∗ *filename*) `[pure virtual]`

Attempts to open the 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.

7.12.2.2 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.

7.13 iPrinter Class Reference

Writes an object file and prints a listing to standard out.

Public Member Functions

- virtual Codes::RESULT Open (std::string infile, std::string outname)=0

  Opens the input and output files for reading and writing.
- virtual Codes::RESULT Print (SymbolTable &symbols, Word &file_length)=0

  Reads from the source file, writes an object file, and prints a listing.

7.13.1 Detailed Description

Writes an object file and prints a listing to standard out. This class defines a very simple interface by which the second pass of the two-pass algorithm is completed and the user is presented with useful information about the encoding process as well as an object file if the code is entirely valid.

7.13.2 Member Function Documentation

7.13.2.1 virtual Codes::RESULT iPrinter::Open (std::string *infile*, std::string *outname*) `[pure virtual]`

Opens the input and output files for reading and writing.

Parameters

`in`	*infile*	The name of the input file to be opened.
`in`	*outfile*	The name of the output file to be opened.

Returns

7.13.2.2 virtual Codes::RESULT iPrinter::Print (SymbolTable & *symbols*, Word & *file_length*) `[pure virtual]`

Reads from the source file, writes an object file, and prints a listing.

Parameters

`in`	*symbols*	A SymbolTable produced from a previous read of the input file.
`in`	*file_length*	The size the program should occupy in memory.

Returns

SUCCESS iff the object file could be created; otherwise an appropriate error message.

7.14 iRegister Class Reference

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

Public Member Functions

- virtual Word GetValue () const =0

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

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

  Adds a word of data to the calling object.

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

  Adds a word of data to the calling object.

- virtual Register operator+ (const iRegister &r) const =0

  A standard add operator.

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

  Subtracts a word of data from the calling object.

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

  Subtracts a word of data from the calling object.

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

  A standard subtraction operator.

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

Performs a bit-wise and.

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

  Performs a bit-wise and.

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

  Performs a bit-wise "or".

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

  Performs a bit-wise or.

- virtual void Not ()=0

  Performs a bit-wise not.

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

  A standard assignment operator.

- virtual Register & operator++ ()=0

  A standard pre-increment operator.

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

  A standard post-increment operator.

7.14.1 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 interace class will serve as a base from which the general purpose registers and program counter of the Wi-11 can be defined.

7.14.2 Member Function Documentation

7.14.2.1 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.

7.14.2.2 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.

7.14.2.3 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".

7.14.2.4 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)".

7.14.2.5 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.

7.14.2.6 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".

7.14.2.7 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)".

7.14.2.8 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.

7.14.2.9 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.

7.14.2.10 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.

7.14.2.11 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.

7.14.2.12 virtual void iRegister::Not ( ) `[pure virtual]`

Performs a bit-wise not.

Postcondition

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

7.14.2.13 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".

7.14.2.14 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".

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

A standard assignment operator.

Note

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

7.14.2.16 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.

7.14.2.17 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.

7.15 iSymbolTable Class Reference

Stores symbols and literals extracted from a source file.

Public Member Functions

- virtual void InsertLabel (std::string label, Word addr, bool relocate=false)=0
  Add a label to the table.

- virtual void InsertLiteral (int value, Word addr)=0

  Add a literal to the table.

- virtual void AddExternal (std::string label)=0

  Store an external label name.

- virtual bool IsSymbol (std::string symbol) const =0
- virtual bool IsExternal (std::string label) const =0
- virtual Word GetLabelAddr (std::string symbol) const =0

  Look up the address for a symbol.

- virtual Word GetLiteralAddr (int value) const =0

  Look up the address for a literal.

- virtual bool IsRelocatable (std::string label) const =0
- virtual const std::map< int, Word > ∗ GetLiterals () const =0

  Aids the Printer in outputing literals.

7.15.1 Detailed Description

Stores symbols and literals extracted from a source file. This class defines an interface for storing various mappings vital to the two-pass algorithm used in this assembler.

7.15.2 Member Function Documentation

7.15.2.1 virtual void iSymbolTable::InsertLabel (std::string *label,* Word *addr,* bool *relocate =* `false`) `[pure virtual]`

Add a label to the table.

Parameters

`in`	*label*	The label to the stored.
`in`	*addr*	The address or value to associate it with.
`in`	*relocate*	Whether or not the program is relocatable.

Precondition

Contains(label) returns false.

7.15.2.2 virtual void iSymbolTable::InsertLiteral (int *value*, Word *addr*) `[pure virtual]`

Add a literal to the table.

Parameters

`in`	*value*	The value to be stored.
`in`	*addr*	The address whether the literal will be stored.

7.15.2.3 virtual void iSymbolTable::AddExternal (std::string *label*) `[pure virtual]`

Store an external label name.

Parameters

`in`	*label*	The label to be stored.

7.15.2.4 virtual bool iSymbolTable::IsSymbol (std::string *symbol*) const `[pure virtual]`

Parameters

`in`	*symbol*	The symbol to look for.

Returns

True iff "symbol" is in the table.

7.15.2.5 virtual bool iSymbolTable::IsExternal (std::string *label*) const `[pure virtual]`

Parameters

`in`	*label*	The label to look for.

Returns

True iff "label" is an external label.

7.15.2.6 virtual Word iSymbolTable::GetLabelAddr (std::string *symbol*) const `[pure virtual]`

Look up the address for a symbol.

Parameters

`in`	*symbol*	The symbol to look up.

Precondition

Contains(symbol) returns true.

Returns

The address of the symbol.

7.15.2.7 virtual Word iSymbolTable::GetLiteralAddr (int *value*) const `[pure virtual]`

Look up the address for a literal.

Parameters

`in`	*value*	The literal value to look up.

Returns

The address of the literal.

A literal should always be present as it will be found where it was declared. Any errors must be assembler related, not user related.

7.15.2.8 virtual bool iSymbolTable::IsRelocatable (std::string *label*) const `[pure virtual]`

Parameters

`in`	*label*	The label to look up.

Precondition

Contains(label) returns true.

Returns

True iff the symbol is relocatable.

7.15.2.9 virtual const std::map<int, Word>* iSymbolTable::GetLiterals ( ) const `[pure virtual]`

Aids the Printer in outputing literals.

Returns

An iterator from the literal values to their addresses.

7.16 iWi11 Class Reference

Defines the internal logic of the Wi-11.

Public Member Functions

- virtual bool LoadObj (const char ∗filename)=0

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

- virtual void DisplayMemory () const =0

  Prints the state of memory to standard out.

- virtual void DisplayRegisters () const =0

  Prints the state of every register to standard out.

- virtual bool ExecuteNext (bool verbose=false)=0

  Executes the instruction pointed to by the PC.

Private Member Functions

- virtual iRegister & _GetRegister (const Decoder_Directory::REGISTER_ID &id)=0

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

- 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 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 _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 _STI (const Decoder_Directory::REGISTER_ID &SR1, const iWord &address)=0

  Performs an indirect store.

- 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.

7.16.1 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.

7.16.2 Member Function Documentation

7.16.2.1 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.

7.16.2.2 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.

7.16.2.3 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.

7.16.2.4 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.

7.16.2.5 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.

7.16.2.6 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.

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

Deprecated?

Does nothing.

7.16.2.8 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.

7.16.2.9 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.

7.16.2.10 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.

7.16.2.11 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.

7.16.2.12 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.

7.16.2.13 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.

7.16.2.14 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.

7.16.2.15 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.

7.16.2.16 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.

7.16.2.17 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	*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.

7.16.2.18 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.
Standard out is the console.

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

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

Prints the state of memory to standard out.

Postcondition

The calling object is not changed.

7.16.2.21 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.

7.16.2.22 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.

7.17 iWord Class Reference

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

Public Member Functions

- virtual int ToInt () const =0

  "To non-negative Integer"

- virtual int ToInt2Complement () const =0

  "To Integer as 2's Complement"

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

  "To String"

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

  "To Hexadecimal"

- virtual bool FromInt (int value)=0

  "From Integer"

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

  "From String"

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

  "From Hexadecimal"

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

  Adds two words.

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

  A standard addition operator.

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

  Subtracts two words.

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

  A standard subtraction operator.

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

  "And"s the bits of two words.

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

  "Or"s the bits of two words.

- virtual Word Not () const =0

  "Not"s the bits of a word.

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

  Copies a word.

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

  A standard assignment operator.

- virtual iWord & operator++ ()=0

  A standard pre-increment operator.

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

  A standard post-increment operator.

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

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

- virtual void SetBit (const int i, bool)=0

  Sets the 'i'th bit of the value.

7.17.1 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.

7.17.2 Member Function Documentation

7.17.2.1 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.

7.17.2.2 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.

7.17.2.3 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"

7.17.2.4 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"

7.17.2.5 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".

7.17.2.6 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".

7.17.2.7 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".

7.17.2.8 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.

7.17.2.9 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)".

7.17.2.10 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.

7.17.2.11 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)".

7.17.2.12 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.

7.17.2.13 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.

7.17.2.14 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.

7.17.2.15 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".

7.17.2.16 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.

7.17.2.17 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.

7.17.2.18 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.

7.17.2.19 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.

7.17.2.20 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

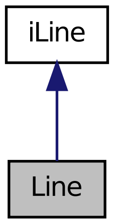
The index must be less than the size of a word, ie. 16.

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

7.18 Line Class Reference

Implements the iLine interface.

Collaboration diagram for Line:



Public Member Functions

- Codes::RESULT **ReadLine** (std::string line)
- std::string **Label** () const
- std::string **Instruction** () const
- std::string **operator[ ]** (int index) const
- int **Size** () const
- int **Literal** () const
- std::string **ToString** () const
- bool **HasLabel** () const
- bool **IsPseudoOp** () const
- bool **HasLiteral** () const
- bool **IsComment** () const

Private Member Functions

- bool _IsWS (char ch) const

  Tests a character for whitespace.

- std::string _GetNext (std::string &str) const

  Get the next whitespace-free sub-string.

- Codes::RESULT _CheckArgs ()

  Tests the arguments extracted from the line for validity.

Private Attributes

- std::string _label

  Label, if any. Empty string otherwise.

- std::string _inst

  Instruction, if any. Empty string otherwise.

- std::vector< std::string > _args

  Holds each argument to the instruction, if any.

- std::string _code

  Holds a copy of the line of code.

- int _literal

  Holds a literal value, if any.

- bool _hasLabel

  True iff line contained a label.

- bool _hasLiteral

  True iff line contained a literal.

- bool _comment

  True iff line was a comment.

7.18.1 Detailed Description

Implements the iLine interface. This class store redundant amounts of information separated in different ways to allow the client to recreate and reformat the original text with ease.

7.18.2 Member Function Documentation

7.18.2.1 bool Line::_IsWS (char *ch*) const [private]

Tests a character for whitespace.

Parameters

in	*ch*	The character to be tested.

Returns

True iff ch is a space or a tab.

7.18.2.2 string Line::_GetNext (std::string & *str*) const [private]

Get the next whitespace-free sub-string.

Parameters

in	*str*	The string from which to obtain the substring.

Returns

The first whitespace-free substring of str.

If str contains only whitespace, the empty string is returned.

7.18.2.3 RESULT Line::_CheckArgs ( ) [private]

Tests the arguments extracted from the line for validity.

Returns

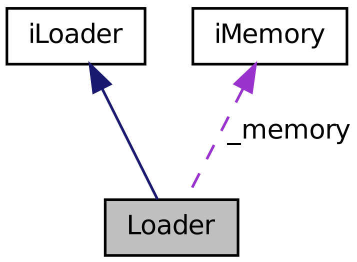
SUCCESS if the arugments are valid; an appropriate error otherwise.

Here, valid arguments are as defined in the Operands sections.

7.19 Loader Class Reference

Implements iLoader.

Collaboration diagram for Loader:



Public Member Functions

- Loader (iMemory ∗mem)

    Set which Memory object is to be initialized by this object.

- Codes::RESULT **Load** (const char ∗filename, iWord &PC_address) const

Private Member Functions

- Codes::RESULT **_GetLoadAddress** (Word &produced_addr, const Word &segment_-length) const

Private Attributes

- iMemory ∗ _memory

    The reference to Memory.

7.19.1 Detailed Description

Implements iLoader.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 Loader::Loader (**iMemory** ∗ *mem*)

Set which Memory object is to be initialized by this object.

Parameters

`in`	*mem*	The address where memory is located.

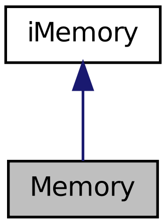
Note

Without this there would be nowhere to load the instructions.

7.20 Memory Class Reference

Implements iMemory.

Collaboration diagram for Memory:



Public Member Functions

- virtual ∼Memory ()

  Deletes any dynamically allocated memory.

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

Private Attributes

- std::vector< Word ∗ > _bounded_memory

  Provide constant time access to reserved memory.

- std::vector< int > _segment_offsets

  Keep track of the initial addresses.

- std::vector< int > _segment_lengths

  Keep track of the size of reserved memory.

- std::map< int, Word > _unbounded_memory

  Map out-of-bounds values to new Words.

7.20.1 Detailed Description

Implements iMemory.

7.21 ObjectData Struct Reference

A simple encoding of a "record".

Public Attributes

- char type

  The type of record: 'H', 'T', or 'E'.

- std::vector< std::string > data

  The segments of the record.

7.21.1 Detailed Description

A simple encoding of a "record".

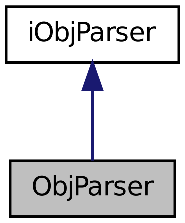
> The format of this component is dependent upon the kind of record it is representing.

- 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)]

7.22 ObjParser Class Reference

Implements iObjParser.

Collaboration diagram for ObjParser:



Public Member Functions

- ∼ObjParser ()

  Closes a file, if necessary, when an iObjParser object goes out of scope.

- Codes::RESULT Initialize (const char ∗name)

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

- ObjectData GetNext ()

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

Private Attributes

- std::ifstream _fileStream

  Maintains an input stream from the object file specified by the "name" parameter to Initialize.

7.22.1 Detailed Description

Implements iObjParser.

7.22.2 Member Function Documentation

7.22.2.1 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.

7.22.2.2 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

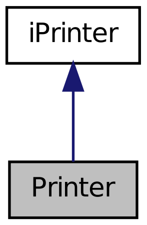
A well-formed ObjectData struct if a valid line is received, a 'dummy' ObjectData struct otherwise.

Implements iObjParser.

7.23 Printer Class Reference

Implements the iPrinter interface.

Collaboration diagram for Printer:



Public Member Functions

- Printer (bool verbose=true)

Sets the mode of execution in terms of console output.

- ~Printer ()

  Closes the input and output files, if necessary.

- Codes::RESULT **Open** (std::string infile, std::string outfile)
- Codes::RESULT **Print** (SymbolTable &symbols, Word &file_length)

Private Member Functions

- void _SetBits (Word &w, int value, int index)

  Set 0 or more bits of a word.

- Codes::RESULT _IsReg (std::string reg)

  Tests a string as a valid register.

- int _RegNum (std::string reg)

  Get the index of register from a string.

- void _SetBits (std::string reg, Word &initial_mem, int &bit_offset)

  Sets the bits of initial_mem for registers.

- Word _ParseWord (const std::string &op, const SymbolTable &symbols)
- bool _Check9 (Word value, Word PC)

  Checks a Pgoffset9 value.

- bool _Check6 (Word value)

  Checks a 6-bit index.

- bool _Check5 (Word value)

  Checks a 5-bit immediate value.

- void _PreError (const std::string &line)

  Prints to the console to make error messages more readable.

- void _LineListing (const Word &current_address, const Word &value, const Line &current_line, const int &pos)

  Prints the listing entry for a single non-pseudo-op instruction.

- std::string _InFileData (const int line_number, const Line &current_line)

  Prints to the console to make listings more readable.

Private Attributes

- std::ifstream _inStream

  The input file.

- std::ofstream _outStream

  The output file.

- bool _verbose

  The Printer object will print to the console iff this is True.

7.23.1 Detailed Description

Implements the iPrinter interface. This implementation shows the complexity of a Printer component. Much of the redundancy inherent in the concept of this class is accounted for through the use of its many private functions. However, the method chosen for returning error messages makes it very difficult to truly optimize this code. Therefore, even much of the code specific to each type of operation is just a variant of another component. The handling of different instructions is done through a while loop containing a large if-elseif-elseif... statement. The components are separated by their argument pattern in an attempt to compactify the code as much as possible while still maintaining its correctness and readability.

7.23.2 Member Function Documentation

7.23.2.1 void Printer::_SetBits (Word & *w*, int *value*, int *index*) [private]

Set 0 or more bits of a word.

Parameters

	in:out]	w The word whose bits are to be set.
in	*value*	An integer corresponding to a bit mask.
in	*index*	The starting position of bit-setting.

Precondition

"index" is less than the size of the word.
"value" can be contained in |size of word| - "index" bits.

Note

This is essentially a localized bit-wise or.

Examples

Word=8, value=1, index=0 => Word=9.
Word=1F, value=14, index=11 => Word=E01F.

7.23.2.2 RESULT Printer::_IsReg (std::string *reg*) [private]

Tests a string as a valid register.

Parameters

in	*reg*	The string to be tested.

Returns

SUCCESS if reg is in the form RX where x is a number from 0 to 7. Otherwise, an appropriate error is returned.

7.23.2.3 int Printer::_RegNum (std::string *reg*) [private]

Get the index of register from a string.

Parameters

in	*reg*	The register string.

Precondition

reg is a valid register.

Returns

The index of register alluded to by reg

7.23.2.4 void Printer::_SetBits (std::string *reg*, Word & *initial_mem*, int & *bit_offset*) [private]

Sets the bits of initial_mem for registers.

Parameters

in	*reg*	The register string.
in	*initial_mem*	The Word object to be changed.
in	*bit_offset*	The location of the first bit to be set (leftmost being 15). Passed by reference for cascade effect.

Precondition

bit_offset > 2

Postcondition

initial_mem has bits [bit_offset:bit_offset-2] appropriately set

7.23.2.5 Word Printer::_ParseWord (const std::string & *op*, const SymbolTable & *symbols*) [private]

Get a 16 value from a string.

Parameters

in	*op*	The string to be parsed.
in	*symbols*	A table of symbols in case it is necessary.

Returns

The value contained in op, whether it is a constant or a symbol. If the symbol is not defined that will be handled outside of this function.

7.23.2.6 bool Printer::_Check9 (Word *value*, Word *PC*) [private]

Checks a Pgoffset9 value.

Parameters

in	*value*	The value to be checked.
in	*PC*	The value of the PC for value to be checked against.

Returns

True iff the highest order 7 bits of value and PC are the same.

7.23.2.7 bool Printer::_Check6 (Word *value*) [private]

Checks a 6-bit index.

Parameters

in	*value*	The value to be checked.

Returns

True iff value can be expressed in 6 bits.

7.23.2.8 bool Printer::_Check5 (Word *value*) [private]

Checks a 5-bit immediate value.

Parameters

in	*value*	The value to be checked.

Returns

True iff value can be expressed in 5 bits.

7.23.2.9 void Printer::_PreError (const std::string & *line*) [private]

Prints to the console to make error messages more readable.

Parameters

in	*line*	The line in which the error was found.

Shows the user the line with the error and formats the output to make error messages more readable.

7.23.2.10 void Printer::_LineListing (const Word & *current_address,* const Word & *value,* const Line & *current_line,* const int & *pos*) [private]

Prints the listing entry for a single non-pseudo-op instruction.

Parameters

in	*current_-address*	The address of the instruction.
in	*value*	The value to be store at current_address.
in	*current_line*	The line that has been translated.
in	*pos*	The current line number in the input file.

7.23.2.11 string Printer::_InFileData (const int *line_number*, const Line & *current_line*) [private]

Prints to the console to make listings more readable.

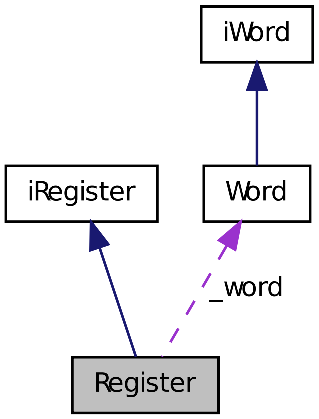
Parameters

in	*line_number*	The current line in the file.
in	*current_line*	The line that has been translated.

7.24 Register Class Reference

Implements iRegister.

Collaboration diagram for Register:



Public Member Functions

- Register ()

  Sets a new Register's value to 0.

- **Register** (const iWord &w)

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

Private Attributes

- Word _word

  The word of data held in the register.

7.24.1 Detailed Description

Implements iRegister.

7.25 ResultDecoder Class Reference

Finds the messages associated with a given result code.

Public Member Functions

- ResultDecoder ()

  Generates the code-to-message mappings.

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

  Looks up a result code.

Private Attributes

- std::map< Codes::ERROR, std::string > _codes

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

7.25.1 Detailed Description

Finds the messages associated with a given result code.

7.25.2 Member Function Documentation

7.25.2.1 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".

7.25.3 Member Data Documentation

7.25.3.1 std::map<Codes::ERROR, std::string> ResultDecoder::_codes [private]

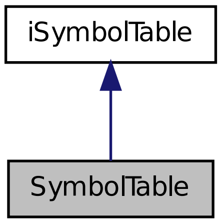
Maps a result code to, in every case but SUCCESS, an error message.

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

7.26 SymbolTable Class Reference

Implements the iSymbolTable interface.

Collaboration diagram for SymbolTable:



Public Member Functions

- void **InsertLabel** (std::string label, Word addr, bool relocate=false)
- void **InsertLiteral** (int value, Word addr)
- void **AddExternal** (std::string label)
- bool **IsSymbol** (std::string symbol) const
- bool **IsExternal** (std::string label) const
- Word **GetLabelAddr** (std::string symbol) const
- Word **GetLiteralAddr** (int value) const
- bool **IsRelocatable** (std::string label) const
- int **LabelCount** () const
- int **LiteralCount** () const
- const std::map< int, Word > ∗ **GetLiterals** () const

Private Attributes

- std::map< std::string, Word > _symbols

  Store symbol to address mappings.

- std::map< int, Word > _literals

  Store literal value to address mappings.

- std::map< std::string, bool > _relocatable

  Store relocation information by string.

- std::map< std::string, bool > _externals

    Store exteranls -- map merely for ease of use.

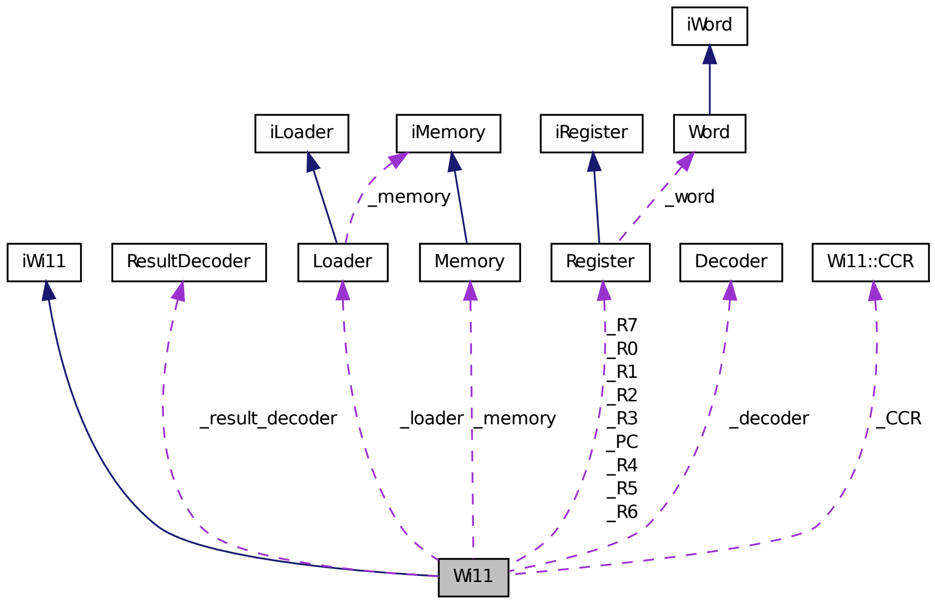
7.26.1 Detailed Description

Implements the iSymbolTable interface. This implementation keeps mappings of the different symbols and literal values to their address, as well as a mapping of symbol to its "relocatable" status. The purpose of this class is primarily to isolate the use of some of the more fragile code and provide the client with a tool rather than a list of components to pass into every function.

7.27 Wi11 Class Reference

Implements iWi11.

Collaboration diagram for Wi11:



Classes

- struct CCR

  Condition code registers: negative, zero, positive.

Public Member Functions

- Wi11 ()

  Creates and organizes the components of the Wi11 machine.

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

Private Member Functions

- std::string _RegisterID2String (const Decoder_Directory::REGISTER_ID &) const

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

- Decoder_Directory::REGISTER_ID **_Word2RegisterID** (const Word &) const

- iRegister & **_GetRegister** (const Decoder_Directory::REGISTER_ID &)
- void **_UpdateCCR** (int)
- 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 Decoder_Directory::REGISTER_-ID &SR2)
- virtual Codes::RESULT **_And** (const Decoder_Directory::REGISTER_ID &DR, const Decoder_Directory::REGISTER_ID &SR1, const iWord &immediate)
- virtual Codes::RESULT **_Branch** (const iWord &address)
- virtual Codes::RESULT **_Debug** ()
- 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 **_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 **_LoadEA** (const Decoder_Directory::REGISTER_ID &DR, const iWord &address)
- virtual Codes::RESULT **_Not** (const Decoder_Directory::REGISTER_ID &DR, const Decoder_Directory::REGISTER_ID &SR)
- virtual Codes::RESULT **_Ret** ()
- virtual Codes::RESULT **_Store** (const Decoder_Directory::REGISTER_ID &SR, const iWord &address)
- virtual Codes::RESULT **_STI** (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)

Private Attributes

- Memory _memory

  Acts as the Wi-11's memory.

- Register _R0

  The 8 general purpose registers and PC.

- Register **_R1**
- Register **_R2**
- Register **_R3**
- Register **_R4**
- Register **_R5**
- Register **_R6**
- Register **_R7**
- Register **_PC**
- struct Wi11::CCR **_CCR**
- Loader _loader

  For loading the object file.

- Decoder _decoder

  For decoding instructions fetch from memory.

- ResultDecoder _result_decoder

  For error messages.

7.27.1 Detailed Description

Implements iWi11.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 Wi11::Wi11 ( )

Creates and organizes the components of the Wi11 machine.

Initializes the general purpose registers, CCR, and memory.

7.28 Wi11::CCR Struct Reference

Condition code registers: negative, zero, positive.

Public Attributes

- bool **n**
- bool **z**
- bool **p**

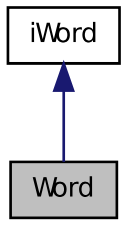
7.28.1 Detailed Description

Condition code registers: negative, zero, positive.

7.29 Word Class Reference

Implements iWord.

Collaboration diagram for Word:



Public Member Functions

- Word ()

  Sets a new Word's value to 0.

- Word (int i)

  Sets a new Word's value to "i".

- int **ToInt** () const
- int **ToInt2Complement** () const
- std::string **ToStr** () const
- std::string **ToHex** () const
- std::string ToHexAbbr () const

  Same as ToHex() but with a different format.

- bool **FromInt** (int value)
- bool **FromStr** (const std::string &str)
- bool **FromHex** (const std::string &str)
- bool FromHexAbbr (const std::string &str)

  Same as FromHex() but with a different format.

- Word **Add** (const iWord &w) const
- Word **operator+** (const iWord &w) const
- Word **Subtract** (const iWord &w) const
- Word **operator-** (const iWord &w) const
- Word **And** (const iWord &w) const

- Word **Or** (const iWord &w) const
- Word **Not** () const
- void **Copy** (const iWord &w)
- Word & **operator=** (const Word &w)
- iWord & **operator++** ()
- iWord & **operator++** (int)
- bool **operator[ ]** (const int i) const
- void **SetBit** (const int, bool)

Static Public Attributes

- static const int **MAX_SIZE** = 0xFFFF

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.

7.29.1 Detailed Description

Implements iWord.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 Word::Word (int *i*)

Sets a new Word's value to "i".

Parameters

in	*i*	The value for the new Word to hold.

Precondition

"i" must fit within 16 bits.

7.29.3 Member Function Documentation

7.29.3.1 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[ ]().

7.29.3.2 string Word::ToHexAbbr ( ) const

Same as ToHex() but with a different format.

The format in question is the same as ToHex() but without leading the "0x" or leading zeros.

Examples

A value of 4 would give "4".
A value of 17 would give "11".
A vallue of -2 would give "FFFE".

7.29.3.3 bool Word::FromHexAbbr (const std::string & *str*)

Same as FromHex() but with a different format.

Precondition

The hex string cannot exceed the values a Word can hold.

The format this function uses is the same as ToHexAbbr().

7.29.4 Member Data Documentation

7.29.4.1 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.

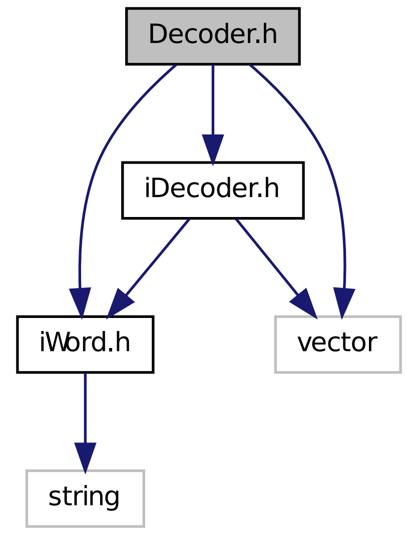
Chapter 8

File Documentation

8.1 Decoder.h File Reference

Definition of the private data for the "Decoder" class. (none)

Include dependency graph for Decoder.h:



Classes

- class Decoder

    Implements iDecoder.

8.1.1 Detailed Description

Definition of the private data for the "Decoder" class. (none)

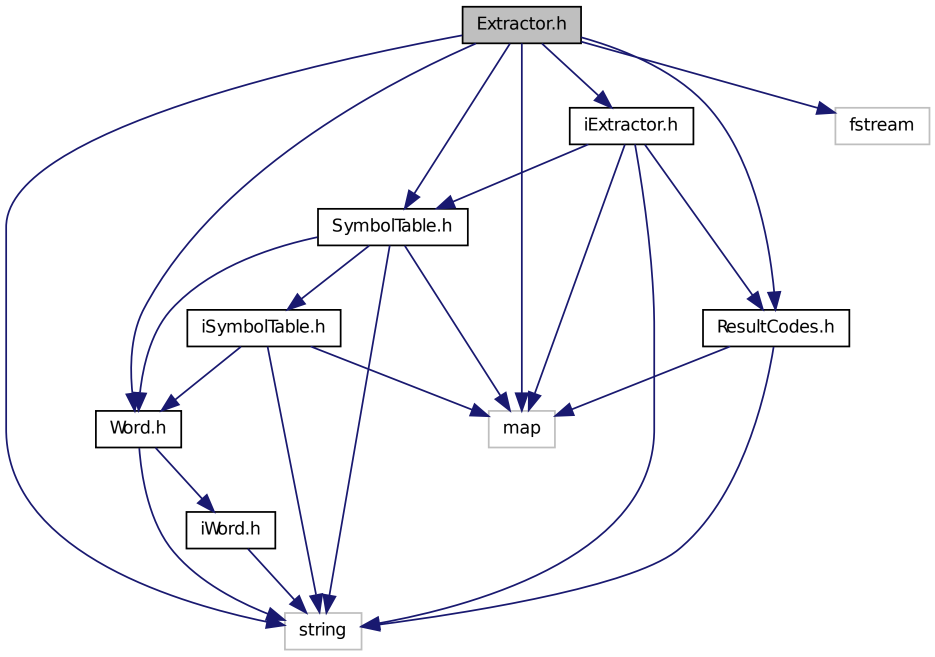
Author

Andrew Canale
Andrew Groot

8.2 Extractor.h File Reference

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

Include dependency graph for Extractor.h:



Classes

- class Extractor

  Implements the iExtractor interface.

Defines

- #define **SYMBOL_TABLE_MAX_SIZE** 1000

8.2.1 Detailed Description

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

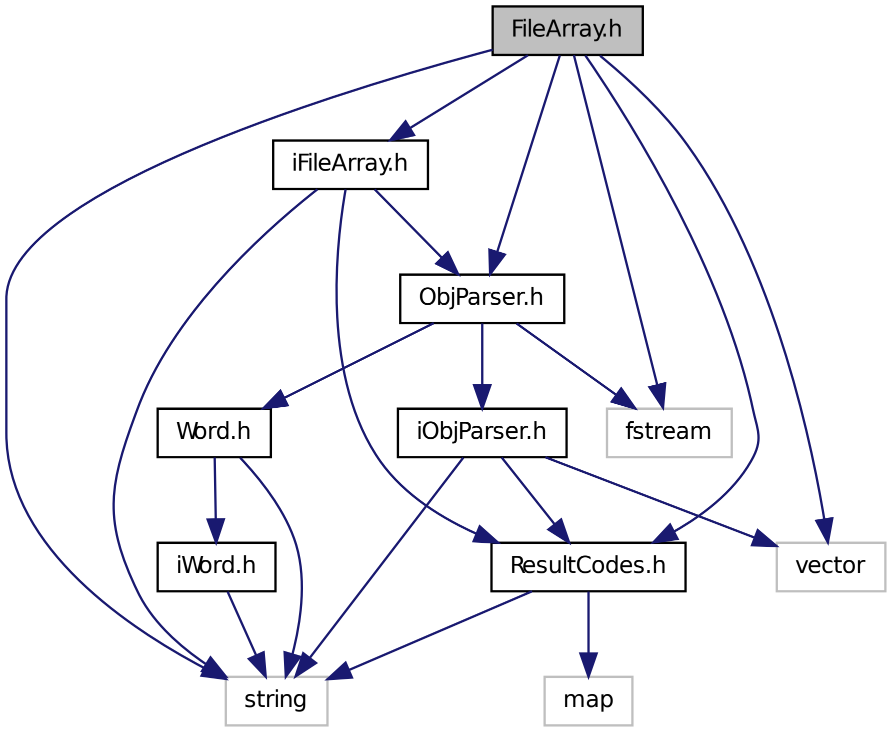
Author

Andrew Groot

8.3 FileArray.h File Reference

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

Include dependency graph for FileArray.h:



Classes

- class FileArray

    Implements iFileArray.

8.3.1 Detailed Description

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

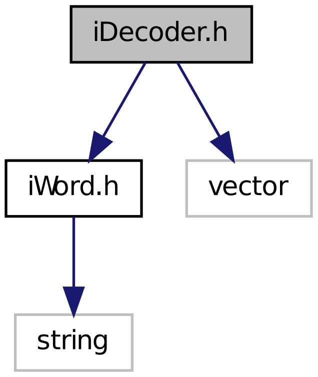
Author

Andrew Groot

8.4 iDecoder.h File Reference

Definition of the Wi-11 instruction decoder.

Include dependency graph for iDecoder.h:



Classes

- struct Instruction

Container to simplify interactions with Wi-11 instructions.

- class iDecoder

  Defines how Wi-11 instructions are decoded.

Namespaces

- namespace Decoder_Directory

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

Enumerations

- enum **REGISTER_ID** {
  R0, **R1**, **R2**, **R3**,
  R4, **R5**, **R6**, **R7**,
  PC }
- enum **INSTRUCTION_TYPE** {
  ADD, **AND**, **BRx**, **DBUG**,
  JSR, **JSRR**, **LD**, **LDI**,
  LDR, **LEA**, **NOT**, **RET**,
  ST, **STI**, **STR**, **TRAP**,
  ERROR }

8.4.1 Detailed Description

Definition of the Wi-11 instruction decoder.

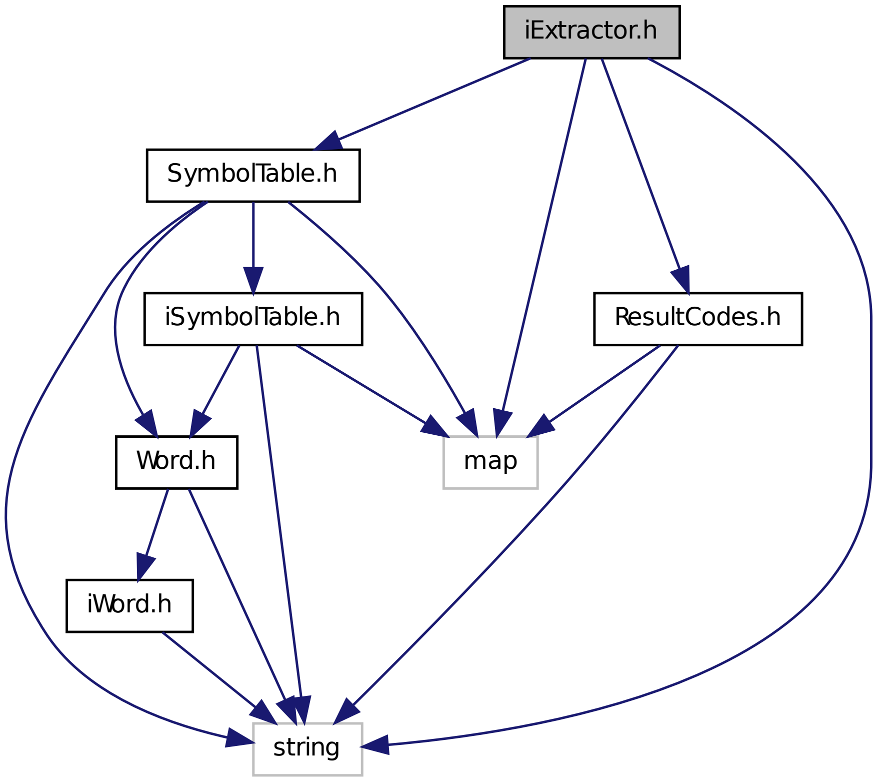
Author

Joshua Green
Andrew Groot

8.5 iExtractor.h File Reference

Definition of the Extractor for the Wi-11 assembler.

Include dependency graph for iExtractor.h:



Classes

- class iExtractor

  Extracts symbols from a source file to be used on a second read.

8.5.1 Detailed Description

Definition of the Extractor for the Wi-11 assembler.

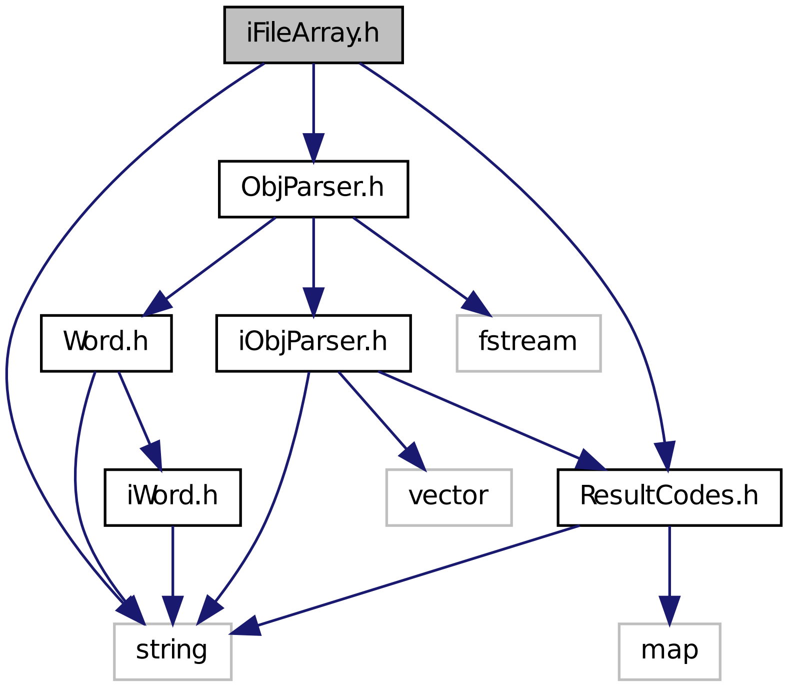
Author

Andrew Groot

8.6 iFileArray.h File Reference

Definition of a file-handling system for the Wi-11 linker.

Include dependency graph for iFileArray.h:



Classes

- class iFileArray

  Keeps an array of files in with a simple ordering property to aid the linking process.

8.6.1 Detailed Description

Definition of a file-handling system for the Wi-11 linker.

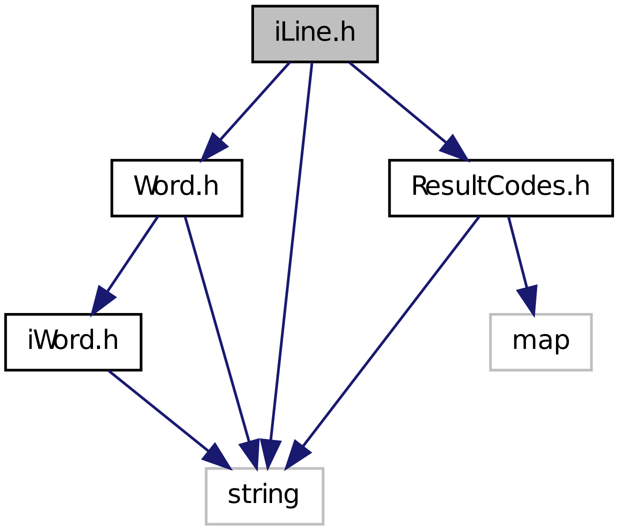
Author

Andrew Groot

8.7 iLine.h File Reference

Definition of a "Line" of Wi-11 assembly code.

Include dependency graph for iLine.h:



Classes

- class iLine

    Stores information about a Line of Wi-11 assembly code.

8.7.1 Detailed Description

Definition of a "Line" of Wi-11 assembly code.

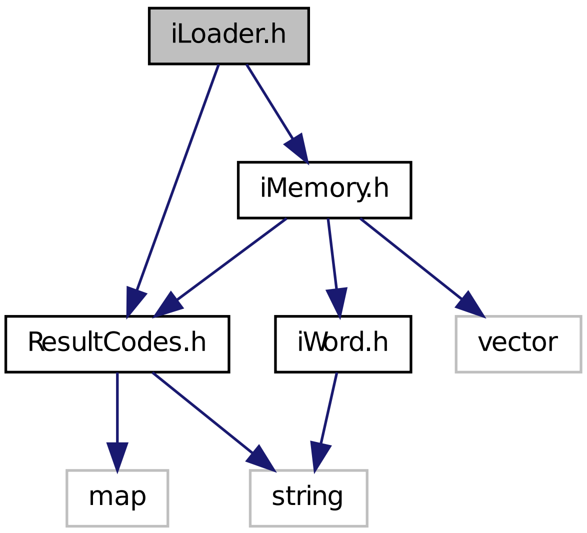
Author

Andrew Groot

8.8 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.

8.8.1 Detailed Description

Definition of the Wi-11 program loader.

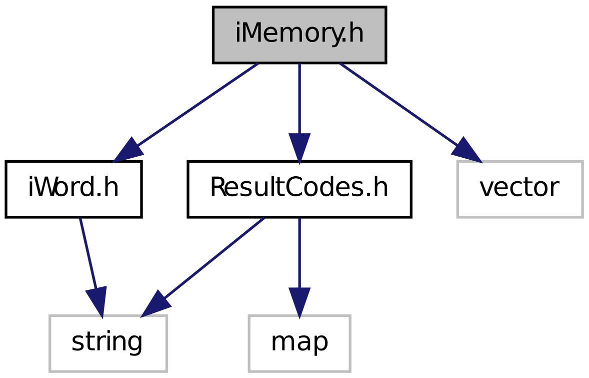
Author

Joshua Green
Andrew Groot

8.9 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.

8.9.1 Detailed Description

Definition of Wi-11 memory.

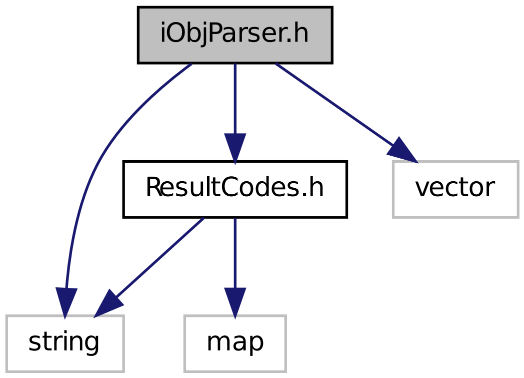
Author

Joshua Green
Andrew Groot

8.10 iObjParser.h File Reference

Definition of the Object File Parser.

Include dependency graph for iObjParser.h:



Classes

- struct ObjectData

  A simple encoding of a "record".

- class iObjParser

  Defines how object files are processed.

8.10.1 Detailed Description

Definition of the Object File Parser.

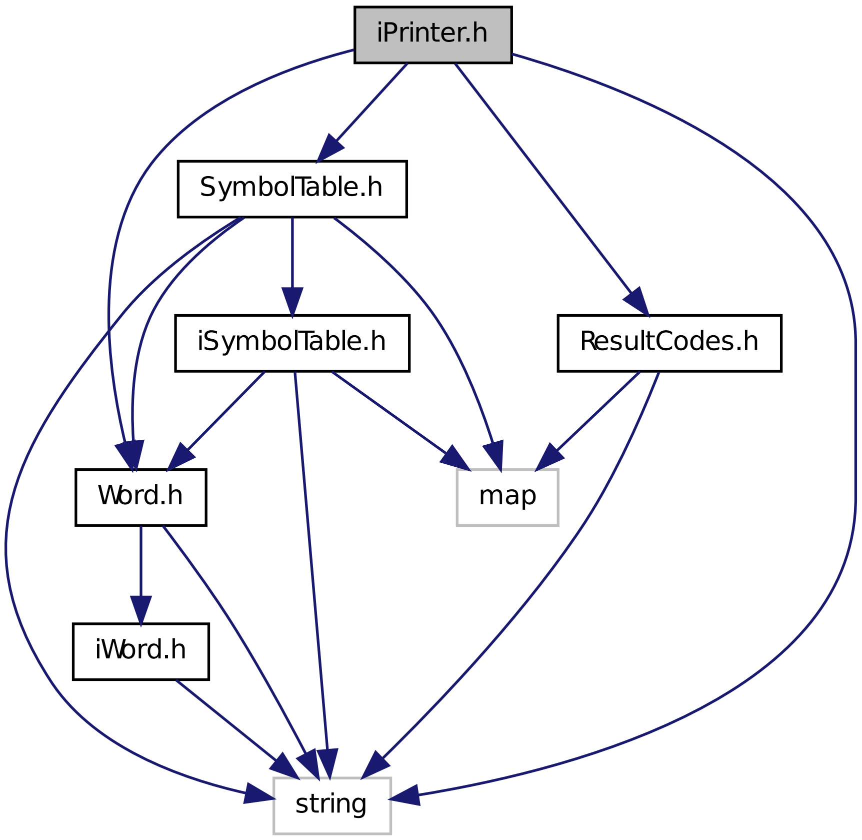
Author

Joshua Green
Andrew Groot

8.11 iPrinter.h File Reference

Definition of the output of the Wi-11 assembler.

Include dependency graph for iPrinter.h:



Classes

- class iPrinter

  Writes an object file and prints a listing to standard out.

8.11.1 Detailed Description

Definition of the output of the Wi-11 assembler.

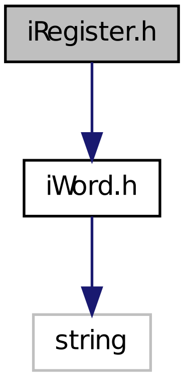
Author

Andrew Groot

8.12 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.

8.12.1 Detailed Description

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

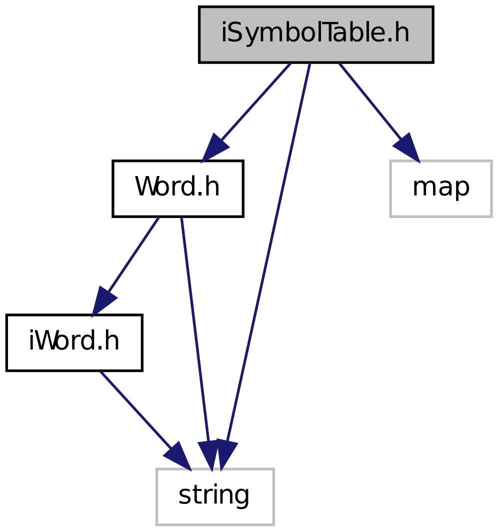
Author

Joshua Green
Andrew Groot

8.13 iSymbolTable.h File Reference

Definition of the symbol table for the Wi-11 assembler.

Include dependency graph for iSymbolTable.h:



Classes

- class iSymbolTable

  Stores symbols and literals extracted from a source file.

8.13.1 Detailed Description

Definition of the symbol table for the Wi-11 assembler.

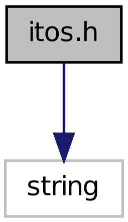
Author

Andrew Groot

8.14 itos.h File Reference

Two utility functions: int to std::string (decimal and hex).

Include dependency graph for itos.h:



Functions

- std::string **itos** (long int number)
- std::string **itoshex** (long int number)

8.14.1 Detailed Description

Two utility functions: int to std::string (decimal and hex).

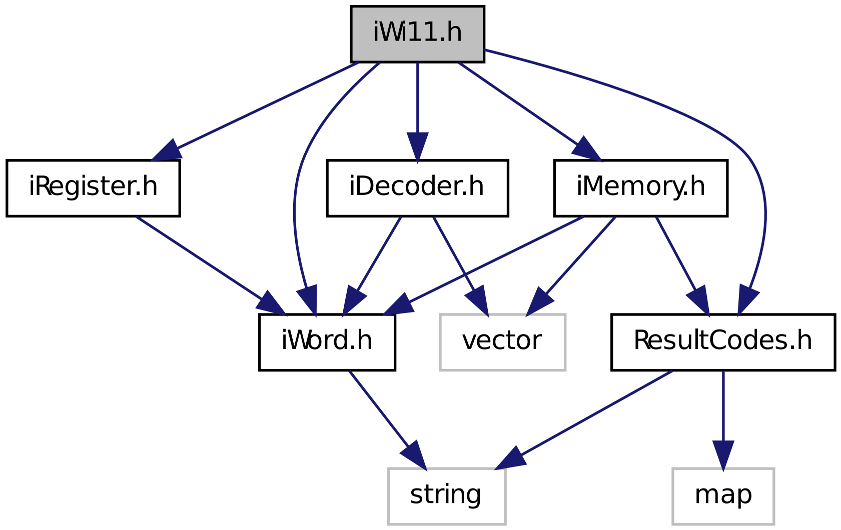
Author

Joshua Green
Andrew Groot

8.15 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.

8.15.1 Detailed Description

Definition of the Wi-11 machine simulator.

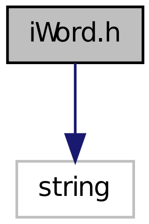
Author

Joshua Green
Andrew Groot

8.16 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.

8.16.1 Detailed Description

Definition of a "word" of data.

Author

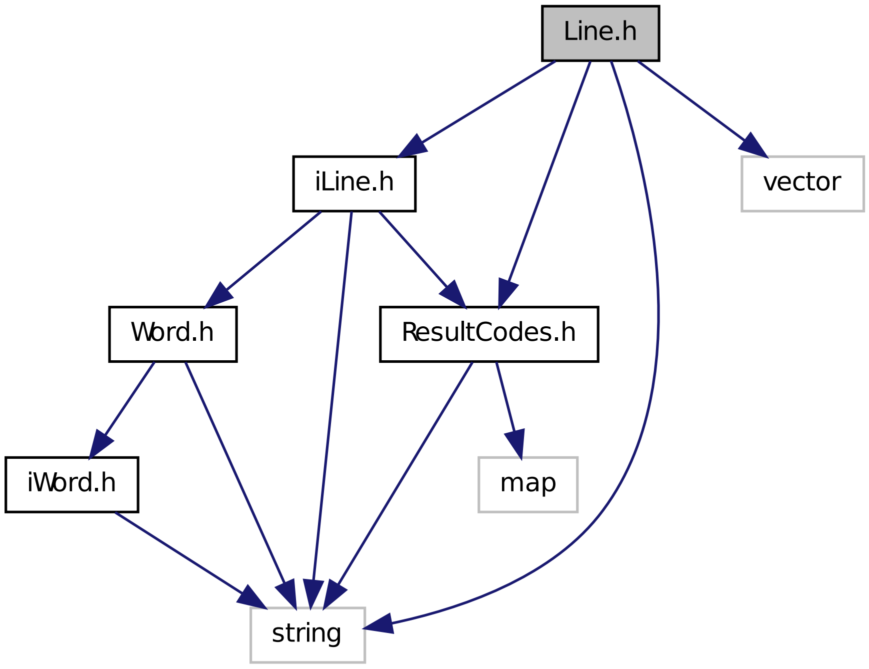
Joshua Green
Andrew Groot

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.

8.17 Line.h File Reference

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

Include dependency graph for Line.h:



Classes

- class Line

    Implements the iLine interface.

8.17.1 Detailed Description

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

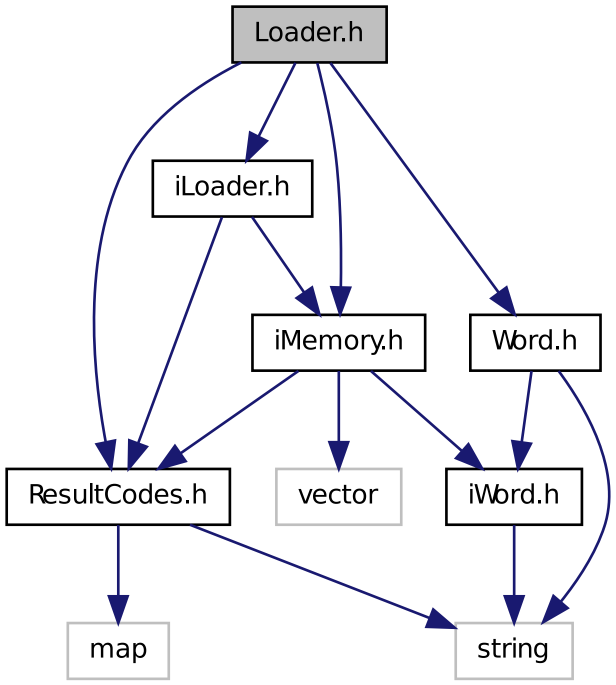
Author

Andrew Groot

8.18 Loader.h File Reference

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

Include dependency graph for Loader.h:



Classes

- class Loader

    Implements iLoader.

8.18.1 Detailed Description

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

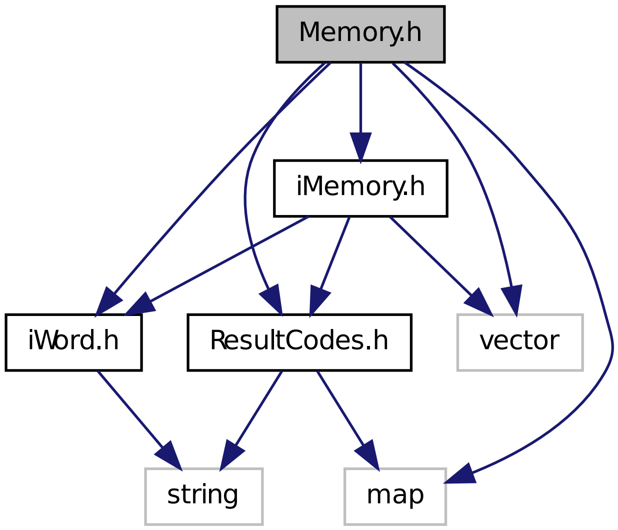
Author

Logan Coulson
Joshua Green
Andrew Groot

8.19 Memory.h File Reference

Definition of private data for the "Memory" class.

Include dependency graph for Memory.h:



Classes

- class Memory

  Implements iMemory.

8.19.1 Detailed Description

Definition of private data for the "Memory" class.

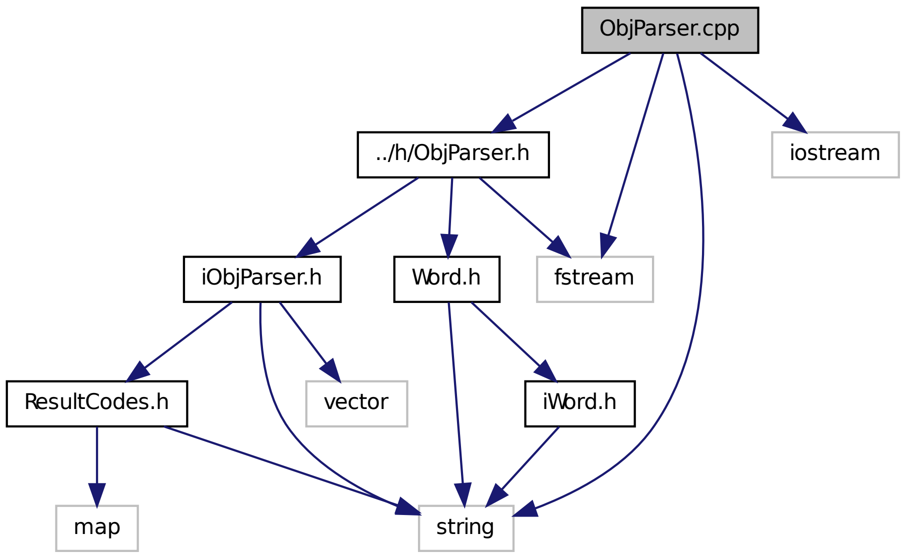
Author

Joshua Green
Andrew Groot

8.20 ObjParser.cpp File Reference

Implements the declarations in "../h/ObjParser.h".

Include dependency graph for ObjParser.cpp:



8.20.1 Detailed Description

Implements the declarations in "../h/ObjParser.h".

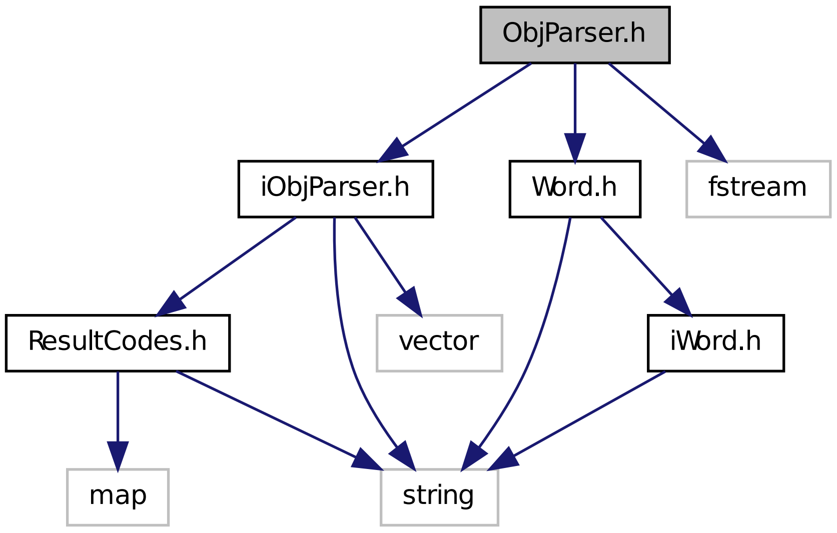
Author

Ryan Paulson
Joshua Green
Andrew Groot

8.21 ObjParser.h File Reference

Definition of private data for the "ObjParser" class.

Include dependency graph for ObjParser.h:



Classes

- class ObjParser

    Implements iObjParser.

Defines

- #define **RELOCATE_FLAG** "RELOCATE"

8.21.1 Detailed Description

Definition of private data for the "ObjParser" class.

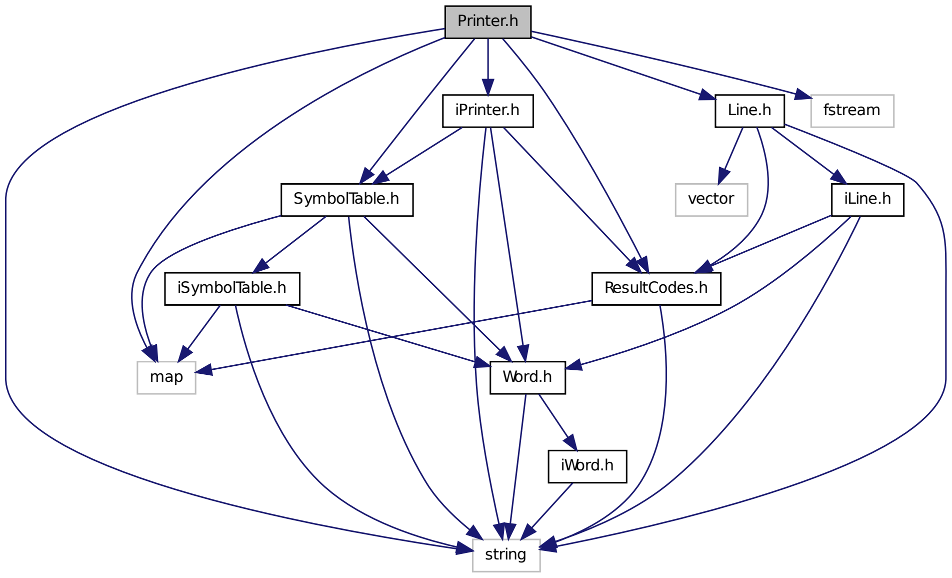
Author

Ryan Paulson

8.22 Printer.h File Reference

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

Include dependency graph for Printer.h:



Classes

- class Printer

    Implements the iPrinter interface.

8.22.1 Detailed Description

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

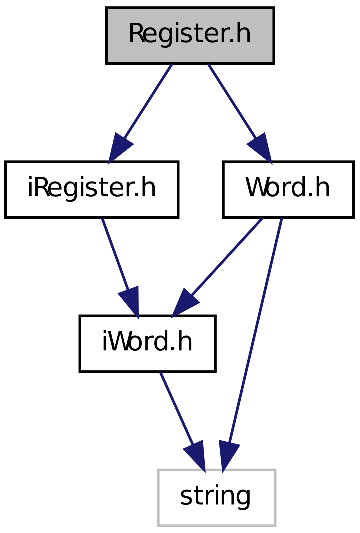
Author

Andrew Groot

8.23 Register.h File Reference

Definition of private data for the "Register" class.

Include dependency graph for Register.h:



Classes

- class Register

  Implements iRegister.

8.23.1 Detailed Description

Definition of private data for the "Register" class.

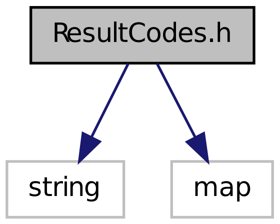
Author

Andrew Groot

8.24 ResultCodes.h File Reference

Definition of the Wi-11 assemblers error messages.

Include dependency graph for ResultCodes.h:



Classes

- struct Codes::RESULT

  Holds error-reporting information.

- 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 **ERROR** {

  ERROR_0, **SUCCESS**, **HALT**, **UNDEFINED**,

  INVALID_HEADER_ENTRY, **INVALID_DATA_ENTRY**, **OUT_OF_BOUNDS**, **NOT_HEX**,

INVALID_TRAP_CODE, **INVALID_START_PC**, **REQUESTED_MEMORY_-TOO_LARGE**, **BAD_MALLOC**,

RELOCATE_ENTRY_IN_ABSOLUTE, **RELOCATION_OVERFLOW**, **RELOCATION_-OUTSIDE_BOUNDS**, **UNRESOLVED_EXTERNAL**,

LINK_ABS, **MULTI_MAIN**, **INV_LBL**, **LBL_WO_INST**,

INV_INST, **STRZ_NOT_STR**, **END_OF_STR**, **STR_JUNK**,

ARG_SIZE, **EMPTY_ARG**, **INV_REG**, **INV_CONST**,

INV_ARG, **INV_HEX**, **INV_DEC**, **INV_BR**,

NON_LD_LIT, **ORIG**, **ORIG2**, **ORIG_LBL**,

ORIG_HEX, **REQ_LABEL**, **LBL_NOT_FOUND**, **REDEF_LBL**,

MAX_S_SIZE, **MAX_L_SIZE**, **MAX_LENGTH**, **ABS_REL**,

INV_IMM, **INV_IDX**, **PG_ERR**, **NO_END**,

END_OB, **UNEXP_EOF**, **REL_PG_SIZE**, **MEM_FIT**,

MAIN_LBL, **END_LBL**, **INV_COMMENT**, **MAIN**,

EXT_REDEF, **RELATIVE**, **FILE_NOT_FOUND**, **FILE_NOT_OPENED** }

8.24.1 Detailed Description

Definition of the Wi-11 assemblers error messages.

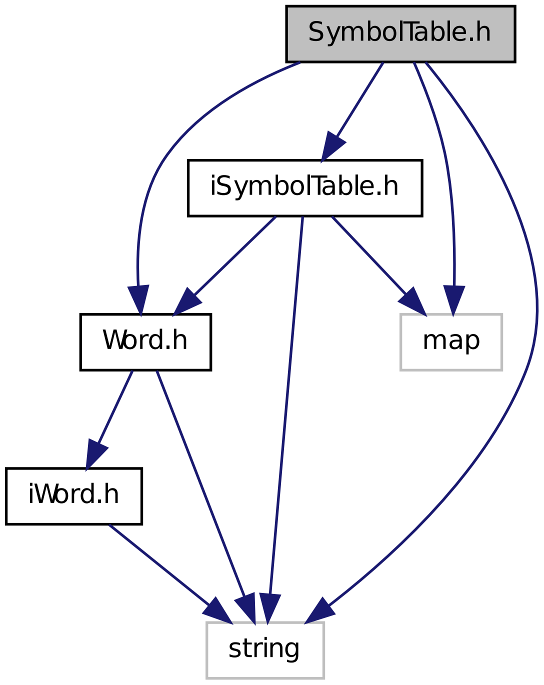
Author

Joshua Green
Andrew Groot

8.25 SymbolTable.h File Reference

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

Include dependency graph for SymbolTable.h:



Classes

- class SymbolTable

  Implements the iSymbolTable interface.

8.25.1 Detailed Description

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

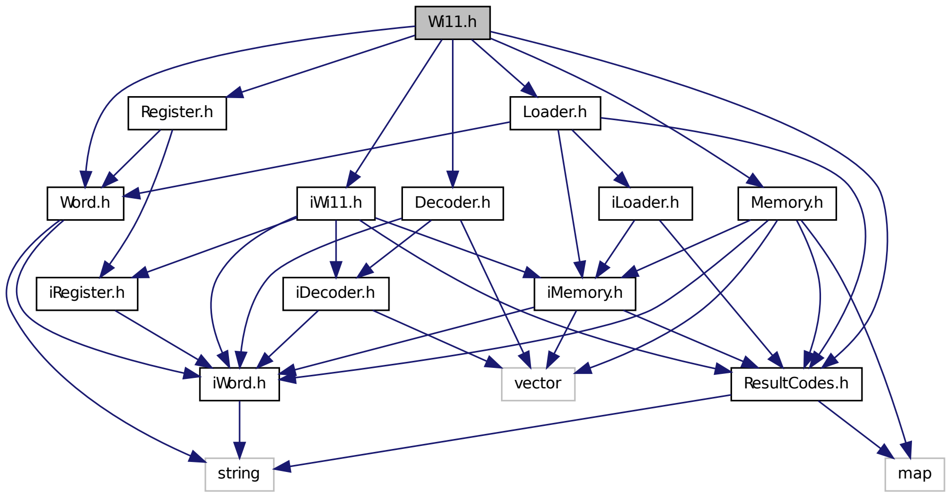
Author

Andrew Groot

8.26 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.

8.26.1 Detailed Description

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

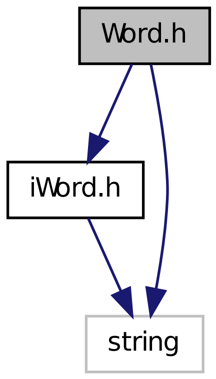
Author

Joshua Green
Andrew Groot

8.27 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

8.27.1 Detailed Description

Definition of private data for the "Word" class.

Author

Joshua Green
Andrew Groot

Index

_Add
  iWi11, 63, 64
_And
  iWi11, 64, 65
_Branch
  iWi11, 65
_Check5
  Printer, 95
_Check6
  Printer, 94
_Check9
  Printer, 94
_CheckArgs
  Line, 84
_Debug
  iWi11, 66
_GetNext
  Line, 84
_GetRegister
  iWi11, 63
_HasBit
  Word, 106
_InFileData
  Printer, 95
_IsReg
  Printer, 93
_IsWS
  Line, 84
_JSR
  iWi11, 66
_JSRR
  iWi11, 66
_LineListing
  Printer, 95
_LineNumber
  Extractor, 35
_Load
  iWi11, 67
_LoadI
  iWi11, 67
_LoadR
  iWi11, 68
_Not
  iWi11, 69
_ParseWord
  Printer, 94
_PreError
  Printer, 95
_RegNum
  Printer, 93
_Ret
  iWi11, 69
_STI
  iWi11, 70
_STR
  iWi11, 70
_SetBits
  Printer, 92, 93
_Store
  iWi11, 69
_Trap
  iWi11, 71
_codes
  ResultDecoder, 98
_value
  Word, 106

Add
  iFileArray, 40
  iRegister, 53
  iWord, 77
AddExternal
  iSymbolTable, 59
And

iRegister, 54, 55
iWord, 79

Clear
iFileArray, 41
Codes, 29
Codes::RESULT, 33
Copy
iWord, 80

data
Instruction, 47
DecodeInstruction
iDecoder, 37
Decoder, 33
Decoder.h, 109
Decoder_Directory, 30
DisplayMemory
iWi11, 72
DisplayRegisters
iWi11, 72

ExecuteNext
iWi11, 73
Extractor, 34
_LineNumber, 35
Extractor.h, 111

FileArray, 36
FileArray.h, 112
Find
ResultDecoder, 98
FromHex
iWord, 77
FromHexAbbr
Word, 106
FromInt
iWord, 76
FromStr
iWord, 77

GetLabelAddr
iSymbolTable, 59
GetLength
iExtractor, 39
GetLiteralAddr
iSymbolTable, 60
GetLiterals
iSymbolTable, 60
GetNext
iObjParser, 49
ObjParser, 89
GetSymbols
iExtractor, 38
GetUsedMemory
iMemory, 47
GetValue
iRegister, 52

HasLabel
iLine, 43
HasLiteral
iLine, 43

iDecoder, 37
DecodeInstruction, 37
iDecoder.h, 113
iExtractor, 37
GetLength, 39
GetSymbols, 38
Open, 38
iExtractor.h, 114
iFileArray, 39
Add, 40
Clear, 41
Name, 40
Size, 40
iFileArray.h, 116
iLine, 41
HasLabel, 43
HasLiteral, 43
Instruction, 42
IsComment, 44
IsPseudoOp, 43
Label, 42
Literal, 42
ReadLine, 42
Size, 43
ToString, 43
iLine.h, 117
iLoader, 44
Load, 44
iLoader.h, 118

iMemory, 45
    GetUsedMemory, 47
    Load, 46
    Reserve, 46
    Store, 46
iMemory.h, 119
Initialize
    iObjParser, 49
    ObjParser, 89
InsertLabel
    iSymbolTable, 58
InsertLiteral
    iSymbolTable, 58
Instruction, 47
    data, 47
    iLine, 42
iObjParser, 48
    GetNext, 49
    Initialize, 49
iObjParser.h, 120
iPrinter, 50
    Open, 50
    Print, 50
iPrinter.h, 121
iRegister, 51
    Add, 53
    And, 54, 55
    GetValue, 52
    Not, 56
    operator+, 53
    operator++, 57
    operator-, 54
    operator=, 57
    Or, 55
    Store, 56
    Subtract, 53, 54
iRegister.h, 122
IsComment
    iLine, 44
IsExternal
    iSymbolTable, 59
IsPseudoOp
    iLine, 43
IsRelocatable
    iSymbolTable, 60
IsSymbol
    iSymbolTable, 59
iSymbolTable, 57
    AddExternal, 59
    GetLabelAddr, 59
    GetLiteralAddr, 60
    GetLiterals, 60
    InsertLabel, 58
    InsertLiteral, 58
    IsExternal, 59
    IsRelocatable, 60
    IsSymbol, 59
iSymbolTable.h, 123
itos.h, 124
iWi11, 61
    _Add, 63, 64
    _And, 64, 65
    _Branch, 65
    _Debug, 66
    _GetRegister, 63
    _JSR, 66
    _JSRR, 66
    _Load, 67
    _LoadI, 67
    _LoadR, 68
    _Not, 69
    _Ret, 69
    _STI, 70
    _STR, 70
    _Store, 69
    _Trap, 71
    DisplayMemory, 72
    DisplayRegisters, 72
    ExecuteNext, 73
    LoadObj, 72
iWi11.h, 125
iWord, 73
    Add, 77
    And, 79
    Copy, 80
    FromHex, 77
    FromInt, 76
    FromStr, 77
    Not, 79
    operator+, 78
    operator++, 80, 81
    operator-, 79

operator=, 80
Or, 79
SetBit, 81
Subtract, 78
ToHex, 76
ToInt, 75
ToInt2Complement, 75
ToStr, 75
iWord.h, 126

Label
iLine, 42
Line, 82
_CheckArgs, 84
_GetNext, 84
_IsWS, 84
Line.h, 127
Literal
iLine, 42
Load
iLoader, 44
iMemory, 46
Loader, 85
Loader, 86
Loader.h, 128
LoadObj
iWi11, 72

Memory, 86
Memory.h, 129

Name
iFileArray, 40
Not
iRegister, 56
iWord, 79

ObjectData, 87
ObjParser, 88
GetNext, 89
Initialize, 89
ObjParser.cpp, 131
ObjParser.h, 131
Open
iExtractor, 38
iPrinter, 50
operator+
iRegister, 53
iWord, 78
operator++
iRegister, 57
iWord, 80, 81
operator-
iRegister, 54
iWord, 79
operator=
iRegister, 57
iWord, 80
Or
iRegister, 55
iWord, 79

Print
iPrinter, 50
Printer, 90
_Check5, 95
_Check6, 94
_Check9, 94
_InFileData, 95
_IsReg, 93
_LineListing, 95
_ParseWord, 94
_PreError, 95
_RegNum, 93
_SetBits, 92, 93
Printer.h, 133

ReadLine
iLine, 42
Register, 96
Register.h, 134
Reserve
iMemory, 46
ResultCodes.h, 135
ResultDecoder, 97
_codes, 98
Find, 98

SetBit
iWord, 81
Size
iFileArray, 40

iLine, 43
Store
    iMemory, 46
    iRegister, 56
Subtract
    iRegister, 53, 54
    iWord, 78
SymbolTable, 98
SymbolTable.h, 136

ToHex
    iWord, 76
ToHexAbbr
    Word, 106
ToInt
    iWord, 75
ToInt2Complement
    iWord, 75
ToStr
    iWord, 75
ToString
    iLine, 43

Wi11, 100
    Wi11, 103
Wi11.h, 138
Wi11::CCR, 103
Word, 103
    _HasBit, 106
    _value, 106
    FromHexAbbr, 106
    ToHexAbbr, 106
    Word, 105
Word.h, 139