Programmer’s Guide

Lab 2 Group BEERZ

**Brad Kline**

**Elliot Schumacher, Evan Todd, Ryan Powers, and Zach Smith**

**2/23/2011**

Contents

[Introduction 3](#_Toc285995847)

[Assembler Overview 3](#_Toc285995848)

[Directory Structure 3](#_Toc285995849)

[Design Conventions 3](#_Toc285995850)

[Module Inter-Relationships 3](#_Toc285995851)

[Data Structures 3](#_Toc285995852)

[Component Descriptions 4](#_Toc285995853)

[Client-Side 4](#_Toc285995854)

[Implementer-Side 5](#_Toc285995855)

[Main 5](#_Toc285995856)

[Assembler 5](#_Toc285995857)

[ByteOperations 5](#_Toc285995858)

[Instruction 5](#_Toc285995859)

[InstructionDefinition 5](#_Toc285995860)

[LiteralTable 5](#_Toc285995861)

[Operand 5](#_Toc285995862)

[OperandDefinition 5](#_Toc285995863)

[OperandType 5](#_Toc285995864)

[Program 5](#_Toc285995865)

[Symbol 5](#_Toc285995866)

[SymbolTable 5](#_Toc285995867)

# Introduction

This document contains a summary of the implementation of the “Wi-11 Machine” assembler. This guide will cover all of the components that make up and are used in the execution of the assembler.

# Assembler Overview

## Directory Structure

## Design Conventions

## Module Inter-Relationships

# Data Structures

There are several important data structures that are used by the assembler in the creation of the object file and the listing file. The following is a listing of the data structure, the file it appears in, the usage of the data structure, its implementation, and its invariant.

* Object: “Instruction” defined by Instruction.java
  + Represents an individual “Wi-11 Machine” instruction
  + Implemented as an **InstructionDefinition,** an array of **Operand**’s and two strings defining the name and the source of the **Instruction**
  + Invariant: the **Instruction** must be a properly formed “Wi-11 Machine” instruction
* Object: “InstructionDefinition” defined by InstructionDefinition.java
  + Represents a certain type of assembly instruction
  + Implemented as a **String** representing the instruction name, an array of **int** that represent the binary values of the final binary code, an array of **OperandDefinition**’s, and an **int** representing the size of the instruction
  + Invariant: the provided **Instruction** must be a properly form “Wi-11 Machine” instruction
* Object: “LiteralTable” defined by LiteralTable.java
  + Represents a table of literals accumulated during the first assembler pass
  + Implemented as a **HashMap<Integer, Integer>();**
  + Invariant: each literal in the table must have a unique address
* Object: “Operand” defined by Operand.java
  + Represents a value (a **Symbol**, register id, literal, or immediate value) to be used by an **Instruction**
  + Implemented as an **OperandDefinition**, a **String**, and an **OperandType**
  + Invariant: An **Operand** must be a valid **Symbol**, register, literal or immediate value
* Object: “OperandDefinition” defined by OperandDefinition.java
  + Represents the acceptable values of **Operand**’s defined for the “Wi-11 Machine”
  + Implemented as an array of **OperandType**
  + Invariant: the **Operand** must be a valid **OperandType**
* Object: “OperandType” defined by OperandType.java
  + Represents the operand types available in the “Wi-11 Machine”
  + Implemented as an enumeration of valid operand types
* Object: “Symbol” defined by Symbol.java
  + Represents the name/value pair of a defined symbol
  + Implemented as **String** and an **int** as well as a **boolean** that determines whether or not the **Symbol** is relocatable or not
* Object: “SymbolTable” defined by SymbolTable.java
  + Represents the internal mappings of names to **Symbols**
  + Implemented as a **HashMap<String, Symbol>();**
  + Invariant: Only unique **Symbols** can be entered into the **SymbolTable**

# Component Descriptions

This section provides a detailed description of both the client- and implementation-side view of every component used in the program, with the exception of the testing package of components.

## Client-Side

* **Main**
  + Description: The **Main** component is the entry point into the assembler program. It accepts the command line arguments from the user, such as, the file containing the source code, the specified object output file, and an option to generate a listing.
* **Program**
  + Description: The **Program** component holds the in-memory representation of the assembly program. The **Program** can then be rendered into its binary representation.
* **Assembler**

## Implementer-Side

### Main

### Assembler

### ByteOperations

* Description: A utility class that is used to obtain specific bits in a given hexadecimal number.
* **ByteOperations Methods:**
* Method: parseHex
  + Description: This method converts a hex number represented in a String to an integer value.
  + Parameters: **String** hex
  + Ensures: hex is a valid hexadecimal number representation.
  + Returns: an integer value representing the input hex value.
* Method: extractValue
  + Description: This method extracts bit values from a given integer.
  + Parameters: **int** value, **int** start, **int** end
  + Requires: start < end
  + Returns: an integer value representing the extracted bits.
* Method: getHex
  + Description: This method gets a string representation of a given integer value.
  + Parameters: **int** value, **int** numCharacters
  + Ensures: All but the least significant hex digits specified by numCharacters are removed.
  + Returns: a string representation of the integer value in hexadecimal form.
* Method: getBit
  + Description: This method returns true if the bit in “value” at the specified right-aligned index is 1.
  + Parameters: **int** value, **int** index
  + Returns: A **boolean** value indicating if the specified bit is 1.

### Instruction

* Description: Each instance of this class represents an assembly instruction that contains a definition, a name and a collection of **Operand**s.

### InstructionDefinition

### LiteralTable

### Operand

### OperandDefinition

### OperandType

### Program

### Symbol

### SymbolTable