|  |
| --- |
|  |
| Documentation for SIC/XE Assembler |
| Implemented by Java |
|  |
|  |
|  |

Group Member

資訊二乙 孔憲文 D0527561

資訊二乙 陳欣惠 D0527783

資訊二乙 賈 澤 D0571783

資訊二丙 赵晨羽 D0571796

資訊二丁 賴恬甜 D0571898

Table of Contents

How to use our Programme

Introduction to our Programme

Flow Chart

Data Structure

Class Diagram

Test Case

How to use our Programme

You may import our project to Eclipse and execute the main method in SicXEAssembler.java.

Or else way you can use command prompt until come to this way:

<Path> \ <Project Name> \ Assembler.jar <Source File Name or leave it blank>

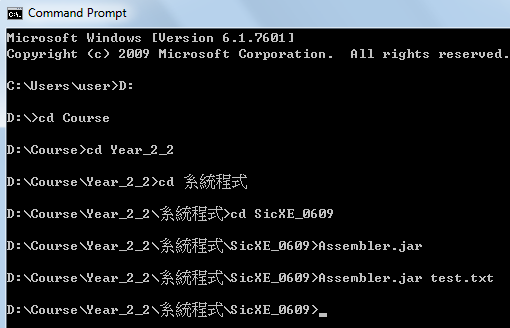
\*\*\* If you leave the Source File Name blank then it will use “test.xe” by default.

For example:

C:\...…\SicXEAssembler\Assembler.jar **or**

C:\...… \SicXEAssembler\Assembler.jar try.txt **or**

C:\...…\SicXEAssembler\Assembler.jar (other file name that you would like to be executed)



**Please make sure your source statement follow our format and intend.**

|  |  |
| --- | --- |
| Source statement | Show paragraph marks and other hidden formatting symbols |
| PROG START 1000  CLEAR T  END | https://i.imgur.com/Ec2JMxL.png |

Introduction to our Programme

**Programming Language and Development Environment used**

* Java and Eclipse

**Fundamental Function**

This part will introduce the basic functions of our assembler in both machine-dependent feature and machine independent feature as well. Machine-dependent features include instruction formats handling, addressing modes handling and program relocation. Machine-independent features include literals, symbol-defining statements and expression.

1. Instruction Formats

* Format 1, 2, 3 and 4

1. Addressing Modes

* Index, Immediate, Indirect, Direct and Relative(PC and Base)

1. Assembler Directives

- START, END, RESB, RESW, BYTE, WORD

1. Literals

- be able to recognize duplicate literals by comparing the character strings that define them

1. Symbol-Defining Statements

– EQU, ORG

1. Expression

– be able to handle the expressions which have label name, constant, mathematical symbol (+, -, \*, /) and parentheses in the correct order

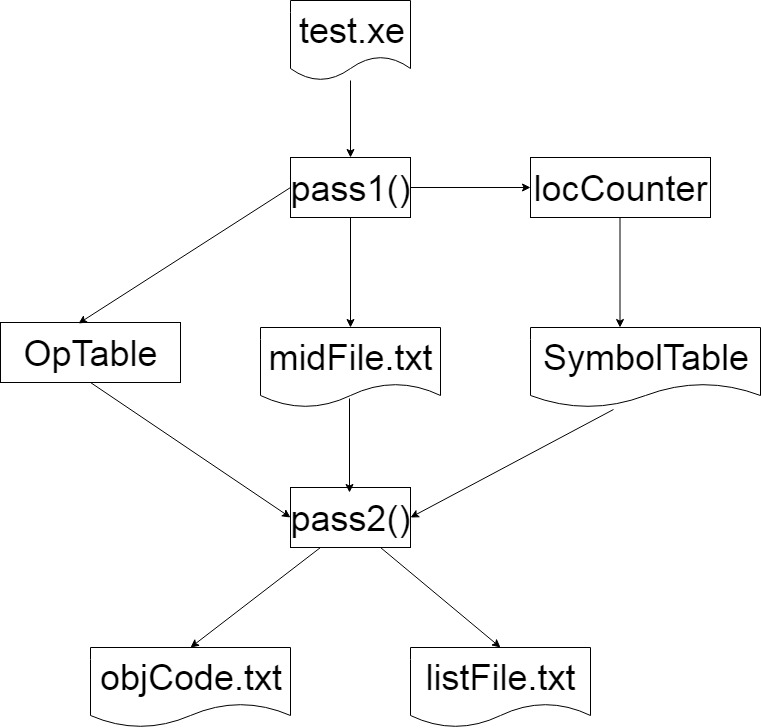
**Limitations**

The assembler that we designed was unable to handle program blocks, control sections and program linking.

BASE <Label Name> cannot BASE \* and BASE <address>

Read input line as string ,comment after operand wont be saved

Flow Chart



Data Structure

* Operation Code Table - Class OpTable
* Implemented by 3-Dimension Array
* Example:

opTable[0][0][0] = "ADD"; 🡪 Mnemonic Opcode

opTable[0][0][1] = "3"; 🡪 Format 3 Instruction

opTable[0][0][2] = "18"; 🡪 Machine Code

|  |  |  |  |
| --- | --- | --- | --- |
| opTable[][] | [0] | [1] | [2] |
| Data represent | Mnemonic Opcode | Instruction Format | Machine Code |
| Example | ADD | 3 | 18 |
| ADDF | 3 | 58 |
| CLEAR | 2 | B4 |

* Symbol Table - Class SymbolTable
* Implemented by Hashtable and 1-Dimension Array
* Example:

private Hashtable<String, String[]> hashTable;

String[] inputCopy = **new** String[3];

inputCopy[0] = **new** String(inputName);

inputCopy[1] = **new** String(value);

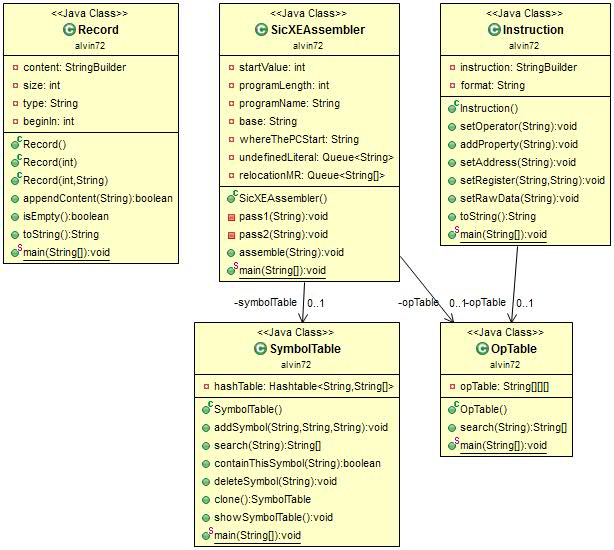
inputCopy[2] = **new** String(type);

hashTable.put(**new** String(inputName), inputCopy);

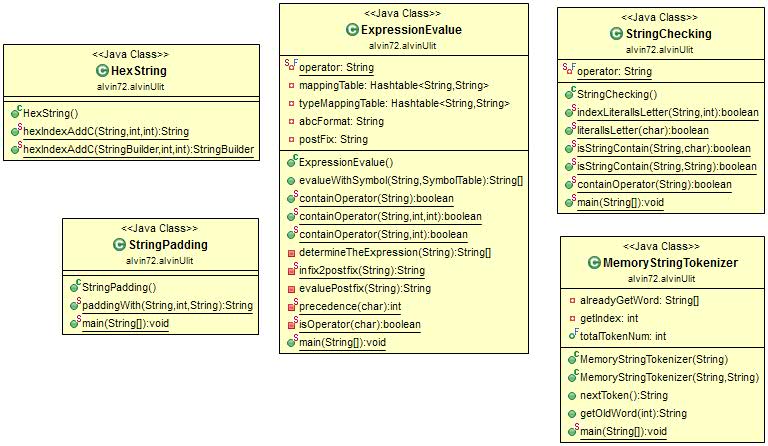
|  |  |  |  |
| --- | --- | --- | --- |
| inputCopy | [0] | [1] | [2] |
| Variable name in method definition | inputName | value | type |
| Represent | Label Name | Address | A: Absolute/  R: Relative |
| Example | COPY EQU 8 | | |
| COPY | 0000 | A |

* Location Counter - locCounter
* locCounter is a variable accumulated for address assignment
* is initialized to be the beginning address specified in the START statement
* after each source statement is processed during pass 1, instruction length or data area is added to locCounter

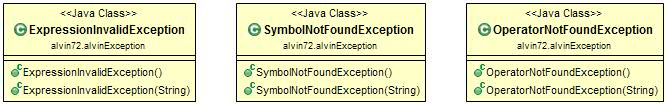
Class Diagram



Class Diagram



Class Diagram



Test Case

We write a main method in almost all the class definition to test the function of that class. So, you may ignore the main methods in other classes, only the main method in SicXEAssembler class will make changes to those output files. In addition, we prepare some assembly code for testing purpose to make sure that it works what we think it should be.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Function |  | Checked? | Page |
| Instruction Format | 1 | ✔ |  |
|  | 2 | ✔ |  |
|  | 3 | ✔ |  |
|  | 4 | ✔ |  |
| Program Relocation |  | ✔ |  |
| Addressing Modes | Index | ✔ |  |
|  | Immediate | ✔ |  |
|  | Indirect | ✔ |  |
|  | Direct | ✔ |  |
|  | PC Relative | ✔ |  |
|  | Base Relative | ✔ |  |
| Assembler Directives | START | ✔ |  |
|  | END | ✔ |  |
|  | RESB | ✔ |  |
|  | RESW | ✔ |  |
|  | BYTE | ✔ |  |
|  | WORD | ✔ |  |
| Literals | =C’ | ✔ |  |
|  | =x’ | ✔ |  |
|  | Duplicate literals | ✔ |  |
| Symbol-Defining Statement | EQU |  |  |
|  | ORG |  |  |
| Expression |  |  |  |
| Relative | Label name |  |  |
|  | \*(curent location) |  |  |
|  | Error(-R, 3\*R, R/2, |  |  |
| Absolute | Constant |  |  |
|  | =,-,\*,/ |  |  |
|  | (,) |  |  |
| Object Code | H |  |  |
|  | T |  |  |
|  | T > 30 bytes |  |  |
|  | T > 30 bytes(break on literal) |  |  |
|  | M |  |  |
|  | E |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Assembly Program | List File |  |
|  | PROG START 1000  FLOAT 3  CLEAR T  LDA B  +LDB #4096  END | line 1|location: 0000| |PROG START 1000  line 2|location: 1000|C0 | FLOAT 3  line 3|location: 1001|B450 | CLEAR T  line 4|location: 1003|030000 | LDA B  line 5|location: 1006|69101000| +LDB #4096  line 6|location: 100A| | END | 🡪program relocation  🡪instruction format 1  🡪instruction format 2  🡪instruction format 3  🡪instruction format 4 |
|  | PROG START 0  LDA TAG,X  TAG LDB #8  LDT @TAG  LDX TAG  END | line 1|location: 0000| |PROG START 0  line 2|location: 0000|03A000 | LDA TAG,X  line 3|location: 0003|690008 |TAG LDB #8  line 4|location: 0006|762FFA | LDT @TAG  line 5|location: 0009|072FF7 | LDX TAG  line 6|location: 000C| | END | 🡪index addressing mode  🡪immediate addressing mode  🡪indirect addressing mode  🡪direct addressing mode |
|  | PROG START 0  LDB #VBASE  BASE VBASE  LDA TAGPC  LDB TAGZ  TAGPC LDT #8  VBASE RESW 1  TAGB RESB 4096  TAGZ BYTE X'12'  WORD 32  END | line 1|location: 0000| |PROG START 0  line 2|location: 0000|692009 | LDB #VBASE  line 3|location: 0003| | BASE VBASE  line 4|location: 0003|032003 | LDA TAGPC  line 5|location: 0006|6B4003 | LDB TAGZ  line 6|location: 0009|750008 |TAGPC LDT #8  line 7|location: 000C| |VBASE RESW 1  line 8|location: 000F| |TAGB RESB 4096  line 9|location: 100F|12 |TAGZ BYTE X'12'  line 10|location: 1010|000020 | WORD 32  line 11|location: 1013| | END | 🡪assembler directive: START  🡪PC-relative  🡪Base-relative  🡪assembler directive: RESW  🡪assembler directive: RESB  🡪assembler directive: BYTE  🡪assembler directive: WORD  🡪assembler directive: END |
|  | PROG START 0  LDA =C'ABC'  XYZ LDB =X'1234'  LDX =C'ABC'  END | line 1|location: 0000| |PROG START 0  line 2|location: 0000|032006 | LDA =C'ABC'  line 3|location: 0003|6B2006 |XYZ LDB =X'1234'  line 4|location: 0006|072000 | LDX =C'ABC'  line 5|location: 0009| |414243 END -> LTORG  line 5|location: 000C|1234 | END | 🡪literal =C’  🡪literal =X’  🡪duplicate literal |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |