Team 3 - C6461 Assembler User Guide

Overview:

The C6461 Assembler is a two-pass assembler that translates assembly language source code into machine code for the C6461 architecture. The assembler reads assembly source files and produces both listing files (for human review) and load files (for use by the simulator).

Key Features:

1. Two-pass design for forward label reference resolution.
2. Supports 5 core C6461 instruction: LDR, LDA, LDX, JZ, HLT
3. Supports 3 assembler directives: LOC, DATA, End:.
4. Octal output format matching C6461 architecture specification.
5. Error reporting for I/O failures, parse errors and unknown instructions.
6. Cross-platform Java implementation- runs on Windows, macOS and linux.

System Requirements

- Java Development Kit (JDK): Version 21 or higher
- Operating System: Windows, macOS, or Linux
- Memory: Minimum 256MB RAM
- Disk Space: Minimal (< 1MB for assembler files)
- Input File : source.txt(plain text, UTF-8)

To Verify your Java Installation, run : java -version
Expected Output: openjdk version “21” or higher

Installation and Setup by Compiling from source

- Ensure Assembler.java and source.txt are in your working directory.
- Compile the source code by running : javac Assembler.java.
- This creates an Assembler.class file, Assembler$LineParts.class and Assembler$Token.class.

Running the Assembler

- Method 1: Using JAR File : java -jar Assembler.jar
- Method 2: Using Compiled Class: java Assembler
  - Note: the assembler reads from hardcoded filename source.txt. Place your assembly source code in a file named source.txt in the same folder as the JAR or class files before running.
- Expected output on success :
  - listing.txt - human-readable listing with addresses, machine code and source.
  - load.txt : machine-loadable address and code pairs for the simulator.

Input File format

File Requirements:

- Filename: source.txt(hardcoded-must use this exact name)
- Encoding: UTF-8 plain text
- Line Endings : Any format supported(Windows, Unix or Mac)

Assembly Language Syntax

- Labels are defined by adding a colon after the name (e.g., Start:).
- Directives control the memory layout and data storage:
  - LOC 100 sets the location counter to 100 in decimal.
  - DATA 25 allocates a single memory word and stores the decimal value 25.
  - DATA End : allocates a word and stores the resolved memory address of the label name “End”.
- Instructions follow the C6461 16-bit specification. The general format is MNEMONIC R, IX, ADDR:
  - LDR 1,0,10 loads register 1 from memory address 10.
  - LDX 2,15 loads index register 2 from memory address 15.
  - JZ 0,1,0 jumps to address 0 if register 0 equals zero.
  - HLT halts all execution.
- Comments start with a semicolon and continue to the end of the line.

Supported Instructions

- Load: LDR(format: LDR r,ix,addr) Load Register from memory address.
- Load: LDA(format: LDA r,ix,addr) Load Address into register.
- Load: LDX(format: LDX r,addr) Load Index Register from memory.
- Transfer:JZ(format: JZ r,ix,addr) Jump to address if register equals zero.
- Control:HLT(format: HLT) Halt execution.

Machine Code Encoding:

All instructions are encoded as 16-bit machine words using the following bit field layout:

- Bits 15-10: Opcode (6 bits)
- Bits 9-8: R field for register (2 bits)
- Bits 7-6: IX field for index register (2 bits)
- Bit 5: I field for indirect addressing (1 bit)
- Bits 4-0: Address or immediate value (5 bits)

All output values are written in 6-digit octal format(%06o) matching the C6461 architecture specification.

Output Files

- **Listing File(listing.txt)**

  This file contains the complete assembly listing for debugging and review. Each line shows the memory
  address in octal, the machine code in octal, and the source line with comments.

For example:

```
000006 000012 Data 10 ; memory[6] = 10
000007 000003 Data 3 ; memory[7] = 3
000016 202207 LDX 2,7
```

- **Load File (load.txt)**

  This file contains just the address and machine code pairs ready for the simulator. It looks like:

  Format: AAAAAA MMMMMM

  ```
  000006 000012
  000007 000003
  000016 202207
  ```

Both files are written simultaneously by the emitWord() method- they are always kept in sync.

Example:

Sample : source.txt

```
LOC 6 ;BEGIN AT LOCATION 6
Data 10 ;PUT 10 AT LOCATION 6
Data 3 ;PUT 3 AT LOCATION 7
Data End ;PUT 1024 AT LOCATION 8
Data 0
Data 12
Data 9
Data 18
Data 12
LDX 2,7 ;X2 GETS 3
LDR 3,0,10 ;R3 GETS 12
LDR 2,2,10 ;R2 GETS 12
LDR 1,2,10,1 ;R1 GETS 18
LDA 0,0,0 ;R0 GETS 0 to set CONDITION CODE
LDX 1,8 ;X1 GETS 1024
JZ 0,1,0 ;JUMP TO End IF R0 = 0
LOC 1024
End: HLT ;STOP
```

Steps to run:

- Place your assembly code in source.txt in the same folder as Assembler.jar.
- Open terminal and navigate to the folder : cd path/to/yourfolder
- Run the assembler: java -jar Assembler.jar
- Check the output: Assembly completed successfully.
- Review generated files: listing.txt and load.txt

Error Handling:

The assembler reports errors to the console. The following error types are handled:

- **File I/O error**: Source files not found or unreadable : Error: <IOException message>
- **Parse Error**: Malformed token or invalid number: Error: Errors processing line: <line>
- **Unknown Instruction**: Unrecognised mnemonic in encode() : Error: Unknown instruction <mnemonic>
- **Missing Operand**: Too few operands for instruction : Error: Missing operand
- **Label Error**: Undefined label reference: Error: Falls to Error processing line: <line>
- **Address Overflow**: Address exceeds 5-bit limit(>31) : Error : not checked- encoded silently.

Troubleshooting:

- **FileNotFoundException: source.txt**
  - Problem: The assembler cannot find the input file.
  - Solution: Ensure source.txt exists in the same folder as Assembler.jar.
  - Check with: ls(Mac/Linux) or dir(Windows)
- **UnsupportedClassVersionError**
  - Problem: your java version is too old.
  - Solution Install JDK 21 or higher.
  - Check with: java -version
- **Error: Unknown instruction <mnemonic>**
  - Problem: You used an instruction not yet implemented.
  - Currently supported: LDR,LDA, LDX, JZ, HLT only.
  - Solution: Use only the 5 supported instructions.
- **Error processing line: <line>**
  - Problem: Invalid operand format or undefined label reference.
  - Solution: check operand syntax, registers must be 0-3, addresses 0-31.
  - Verify label names match exactly between definition and usage.
- **Output files not generated**
  - Problem: Assembly failed before output was written.
  - Solution Check error messages in the console and fix source.txt.

Appendix:

Appendix A: File Structure

- **Source.txt** : Assembly source input, must be provided by user : Generated by user.
- **Listing.txt** : Human-readable listing with address, code and source : Generated by Assembler (pass 2)
- **Load.txt** : Machine-loaded address/code pairs for simulator : Generated by Assembler (pass 2)
- **Assembler.jar** : Executable JAR - run with java -jar Assembler.jar : Generated by Build
- **Assembler.java** : Java source code : Generated by Development

Appendix B : Two-Pass Summary

Pass 1 : firstPass() : source.txt(first read) : produces sym HashMap - label to address mapping.
Pass 2: secondPass() :source.txt(second read) : produces listing.txt and load.txt in octal