CSCI 6461 Spring 2026
Assembler Part 0 Documentation
Group 2

How to load and run the assembler:

To start, you will need Java JDK 8 or higher, our Assembler.java file, and a source assembly file named ‘source.txt’. An important note is that the program is hardcoded to read the source.txt file. The code does not currently accept command-line input. In the terminal, open the folder with the .java file and run ‘javac Assembler.java’. This command creates the Assembler.class file. Make sure the source.txt file is in the same directory as the .java file. Run ‘java Assembler’. The assembler then runs, and two output files are generated. A description of the assembler and design notes are below. One file that is created is the ‘listing.txt’. This contains the memory address in octal, machine code in octal, and the original source line. The other file that is created is the ‘load.txt’. This contains the address in octal and machine word in octal.

Design notes:

- The overall architecture for this assembler uses a two-pass design. In the main, we will run the overall function, and inside of that are the calls for pass 1 and pass 2.

The main data structures in the code:

- The symbol table, which is a mapping of strings and integers that creates our dictionary and stores each label to memory address. This is built in pass 1 and used in pass 2.
- The opcode table, which is a mapping of strings and integers that creates our opcode map and stores each instruction to decimal opcode. This is initialized in the constructor.

Pass 1:

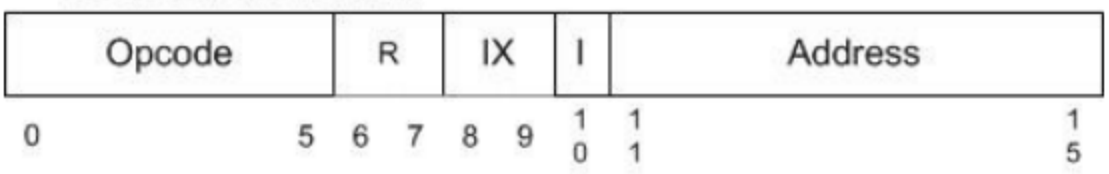
- Reads the file line by line
- Skips comments
- Finds labels ending with a ‘:’
- Stores label addresses
- Handle directives of LOC and DATA
- Increases memory location for each instruction

Pass 2:

- Rereads the source file
- Creates the machine code
- Uses the dictionary to map the labels
- Write output to the listing and load files

Instruction format:

- Each instruction is encoded as a 16-bit word



- This is constructed using the ‘int instruction’
- The ‘%06o’ is used to print values in octal

Assembler Directives:

- LOC, sets the memory location to ‘n’ and does not increment
- DATA, stores a word at current location and increment

Error Handling:

- Repeated labels
- Missing LOC operand
- Invalid LOC value
- Unknown instruction
- Invalid indirect bit
- Invalid numeric values
- If an error occurs, the assembler will print an error message and stop execution

Walking through the source file:

- Stores numbers in memory
- Loads registers with values
- Sets R0 = 0
- Loads X1 with address 1024
- Uses JZ to jump to End
- Executes HLT