Assembler - Linker - Loader

Dheeraj Khatri, Dhruv Kohli, Rahul Kadyan

IITG CS244 System Programming Lab
Lab Instructor: Prof.Santosh Biswas, Prof.Arnab Sarkar

Assembler-Linker-Loader for 8085 assembly language

Contents

1 Introduction

1.1 PURPOSE

The over reaching goal of the project is to prepare an application that converts the application defined syntax respecting code to a GNUSIM 8085 executable code through a usual process of Assembling, Linking and Loading the code. The application developed, aims to help a general audience including CS students professors, to easily write a code using a simple application defined syntax and hence convert it to the corresponding GNUSIM 8085 executable code. The application is published under GPL and is open for the developers.
Tha application is developed using JAVA and LibGDX library and can be deployed on:
1.Desktop
2.Android
3.WebPage
4.Iphone

1.2 INSTRUCTION SET

The instruction set contains all the instructions predefined in 8085-Assembly. Apart from that there a dozen more instructions defined to make programming easier.

ADDMM &m1,&m2,&m3: Add 2 numbers stored in 'm2' and 'm3' memory locations. Stack starts from 'm1' memory location

ADDMR &m1,&m2: Add number in the register to the number in 'm2' memory. Stack starts from 'm1' memory location

ADDVV &m1,&m2,v1,v2: Add 2 variables 'v1' and 'v2' and store in 'm2' memory location. Stack starts from 'm1' memory location

ADDVR &m1,&m2,v1: Add variable 'v1' to number stored in 'm2' memory location. Stack starts from 'm1' memory location

SUBMM &m1,&m2,&m3: Subtract numbers stored in 'm3' from 'm2' memory location. Stack starts from 'm1' memory location

SUBMR &m1,&m2: Subtract number in register from number in 'm2' memory location. Stack starts from 'm1' memory location

SUBVV &m1,&m2,v1,v2: Subtract 2 variable 'v2' from 'v1' and store in 'm2' memory location. Stack starts from 'm1' memory location

SUBVR &m1,&m2,v1: Subtract variable 'v1' from number stored in 'm2' memory location. Stack starts from 'm1' memory location

MINM &m1,&m2: Find minimum between the numbers stored in 2 memory locations

MAXM &m1,&m2: Find maximum between the numbers stored in 2 memory locations

MINMM &m1,&m2,&m3: Find minimum among the numbers stored in 3 memory locations

MAXMM &m1,&m2,&m3: Find maximum among the numbers stored in 3 memory locations

SWP &m1,&m2: Swap the elements present in 2 memory locations

MUT &m1, &m2: Multiply a number at memory location 'm2' with 2. Stack starts from memory location m1

1.3 REFERENCES

Assemblers Handout provided.
http://en.wikipedia.org/wiki/Assembly$_{l}anguage$
http://en.wikipedia.org/wiki/Assembly$_{l}anguageAssembler$
http://en.wikipedia.org/wiki/Loader$_{(}computing)$
http://en.wikipedia.org/wiki/Linker$_{(}computing)$
http://docs.oracle.com/javase/7/docs/api/
http://libgdx.badlogicgames.com/documentation.html

1.4 OVERVIEW

Input
Software accepts multiple assembly files written using the defined instruction set.
Assembler - Pass1
In this part the input codes are converted to native 8085 code. Macros expansion is also taken care in here. To ease the programming for user, user can use his predefined macros within another macro.
Assembler - Pass2
Symbol Table for each file is generated and is stored in filename.table. After this all the labels are replaced by relative address values.
Linker
Extern variables are handled in here. All the files are linked with each other.
Loader
The user is asked for the memory location where he wants to load his program. The programs are then dynamically loaded into those specific memory locations.The ouput of filename.s.8085 file can then be run on GnuSim8085

2 Requirements

General:

JAVA/JVM or .jar execution supporting system

I/O: Monitor, Keyboard, Mouse

Specific:

Android: support for OpenGL 2.0

Optional:

Desktop: support for OpenGL 2.0

Iphone: support for OpenGL 2.0

3 Architecture / Design

3.1 CODE & CONSTRUCTION PRINCIPLES

Design specs of an assembler:

(i) Identify the information necessary to perform a task.

(ii) Design a suitable data structure to record the information.

In our applicaton we used Map and List Contatiners to store the gathered information appropriately.

Map.put(key, value) stores the key-value pair in the corresponding map

Map.get(key) retrieves the value at the corresponding key.

Key and value both are objects.

Similarly, List.add(value) appends the list with the value and List.get(index) retrieves the value stored at the integer index specified.

(iii) Determine the processing necessary to obtain and maintain the information and perform the task.

The Main processes/methods involved in our code are:

1. macroPreprocess : Prepreocesses Macros and creates a table
2. opCodePreprocess: preprocesses opCodes and stores length of opCodes with their actual code too using the above mentioned contatiners.
3. createSymbolTable: creates a symbol table.
4. replaceTable: replaces opcodes with their actual code.
5. linkCode: links files generated after replacing opcodes
6. loadCode: loads the code at the user defined location

4 End User Manual

4.1 USERS

-Run system$_p$*rogramming.jarrunnable*.
-Copy the code on to the screen and save it on the stack.
Copy multiple number of codes on the screen and consequently save the m on the stack using the button provided.
-Finally Assemble Pass 1 the code using the corresponding button.
-Output of assemble pass 1 will be the symbol table of the all the codes provided as the input.
-These symbol tables can be accessed by entering the index of the file and then clicking on ”Focus on Queried File”.
-Then one can proceed with the pass 2, linking and loading sequentially with the outputs shown at every step.
-The application demands the user to input an integer representing the location to load the input file.
-Finally the output file is shown which can be copied into the GNUSIM 8085 and hence can be executed.

Developers-
Dheeraj Khatri(120101021)
Dhruv Kohli(120123054)
Rahul Kadyan(120122028)