Computer Organization

(Two-pass Assembler Project)

❑ Objective

To write a Two-pass Assembler for the 12-bit architecture having instructions as follows:

Opcode	Meaning	Assembly Opcode
0000	Clear accumulator	CLA
0001	Load into accumulator from address	LAC
0010	Store accumulator contents into the address	SAC
0011	Add address contents to the accumulator contents	ADD
0100	Subtract address contents from accumulator contents	SUB
0101	Branch to address if accumulator contains zero	BRZ
0110	Branch to address if accumulator contains a negative value	BRN
0111	Branch to address if accumulator contains a positive value	BRP
1000	Read from the terminal and put it in address	INP
1001	Display value in an address on terminal	DSP
1010	Multiply accumulator and address contents	MUL

1011	Divide accumulator contents by address content. The quotient in R1 and remainder in R2	DIV
1100	Stop Execution	STP

❑Two-Pass Assembler

An assembler is a translator, that translates an assembler program into a conventional machine language program. Basically, the assembler goes through the program one line at a time and generates machine code for that instruction. Then the assembler proceeds to the next instruction. In this way, the entire machine code program is created. For most instructions, this process works fine, for example for instructions that only reference registers, the assembler can compute the machine code easily, since the assembler knows where the registers are.
Basically, we scan the code twice. The first time, just count how long the machine code instructions will be, just to find out the addresses of all the labels. Also, create a table that has a list of all the addresses and where they will be in the program. This table is known as the symbol table. On the second scan, generate the machine code, and use the symbol table to determine how far away from jump labels are, and to generate the most efficient instruction.

❑Implementation

In this project, we are given a text file containing our assembly language program in the form of an assembly code. We have to convert this into object code including the mapping of virtual memory space. We also have to include all the possible errors that can be generated during the conversion of assembly code.

To complete the above task, we have created five python files, three text files. The files contain the following information:

- **Input.txt**- This text file contains the assembly code that is to be converted into object code.

- **Main.py-** This python file contains the main script which is to be executed to generate the object code. Majorly, it invokes two functions Pass1 and Pass2.

- **Opcode.py-** It contains the Opcode Table given in the assignment document along with functions such as ifVariable(), ifOpcode(), ifLabel() to check if a token is variable, opcode or label respectively. It also contains some getter and checker functions.

- **SymbolTable.py**- It contains a dictionary symTbl that contains all the variables and labels together and maintains all the addresses allocated during pass1. Functions provided allow for addition to this dictionary these variables and labels and updating their addresses. At the end it just writes

- **Pass1.py-** This python file contains the code for the first traversal of the assembly code and generates SymbolTable.txt if any labels and variables are used, and also handles all the possible errors in the assembly code.

- **SymbolTable.txt-** It contains all the variables and labels and their corresponding addresses.

- **Pass2.py-** This python file contains the code for the second and final traversal of the assembly code. It uses symbol_table.txt to map the virtual addresses to the object code.

- **Output.txt-** It contains the final machine code with virtual addresses assigned. Although the output is a continuous stream of binary values, we have distributed it into 3 columns for readability. The first column represents the virtual address, the second column represents the Opcode and the third column represents the address of variables or labels(All zero addresses represent CLA or STP).

❑Error Handling

We have taken care of the following errors.

Serial Number	Error Encountered
1)	The line is blank. (The given input line contains no character)
2)	Extra White spaces between opcodes and variables
3)	Label defined does not point to any instruction.
4)	Arguments missing for the given opcode.
5)	Too many arguments. (The given opcode is supplied with more than required arguments)
6)	The following arguments cannot be identified as an opcode.

7)	Opcode missing for the given label. (Any label requires an opcode so that it can branch to).
8)	The given attribute should be a variable, not a label.
9)	The given instruction is supplied with no valid opcode.
10)	The following Opcode cannot be given as an argument.
11)	Line must have an opcode as an instruction.
12)	Stop command is not given for the above program.
13)	The given argument cannot find a label to branch.
14)	The used symbol in the argument is not defined.
15)	Cannot branch to an opcode/variable, requires a label.
16)	Memory overflow, cannot exceed more than 255 variables or labels.

Note- We have stored all the variables and labels in SymbolTable.txt.

- Vidhu Chaudhary