Project
Assembly Language Programs

Paras Mehan
Roll No.2018062

Osheen Sachdev
Roll No.2018059

Overview

The code inputs a code in assembly language from the file “code.txt” and converts into machine language code saved as “machine_code.txt”. This is a 2 pass assembler and hence the assembler traverses the instructions twice to generate the machine code. Errors are raised at appropriate places in case the given assembly code contains errors and assembly is aborted.

Running the Code

1. Un-zip the folder.
2. Write the assembly code in the “code.txt” file.
3. Change your directory to the folder.
4. Run the following command: python3 Main.py
5. Errors are displayed on the console.
6. The machine code would be saved in the “machine_code.txt” file.

Details of the Code for Assembler

1. File Name: pass_one.py
    - Description :
        i. pass_one contains the function first_pass(name_of_file) that is used to execute the first pass of an assembler.
        ii. It reads the code line by line and for each instruction between START and END assembler directives and calls the get_instruction function on them.
        iii. The get_instruction function decodes the instruction.
        iv. The decoded instruction is then stored.
        v. After reading all the instructions and simultaneously storing all the labels and variables in the symbol table, the symbol table is checked to make sure no labels have been left undefined.
        vi. The presence of STP is then checked.
        vii. Then variables are assigned virtual memory space.
        viii. Finally, the instructions and symbol table is written into a temporary file.
    - Functions :
        i. assign_memory_to_variables()
            1. Description : Assign memory
            2. Input : None
            3. Returns : None
        ii. assign_opcode()
            1. Description : Assign Opcode if it is a valid opcode

2. Input: Opcode in string format.
3. Returns: None
4. Exception:
    a. invalid opcode - no matching opcode found in opcode table

iii. assign_operands()
1. Description: Checks for operands and insert in the symbol table
2. Input: instruction
3. Returns: None
4. Exceptions:
    a. if Opcode supplied with the wrong number of operands.

iv. check_for_stop()
1. Description: Check if STP is missing at the end of code or not.
2. Input: None
3. Returns: None
4. Exceptions :
    a. STP Missing at end of code.

v. check_symbol_table()
1. Description: Check if any label in symbol table has not been defined in the code
2. Input: None
3. Returns: None
4. Exceptions :
    a. if Label is not defined.

vi. createJSON()
1. Description: Save a dictionary as a JSON file
2. Input: nameOfFile - string, dict_data - dictionary

3. Returns: None

vii. first_pass()
1. Description: Executes pass one of the assembler.
2. Input: FIle name for which code needs to be translated
3. Return: None

viii. get_instruction()
1. Description: Decodes a line and extracts label, mnemonic, operands from a line(If there is any)
2. Input: Line in a string Format
3. Returns: A dictionary of label, mnemonic, operands
    a. If they are not available, there will be None, in place of it.
    b. operands would be a list of operands.
4. Exceptions:
    a. When Label format is not correct
    b. When the OP/CODE is not correct

ix. put_in_symbol_table()
1. Description: Insert symbol, symbol type, and its address into the symbol table
2. Input: symbol - string, symbol type- string, address- int
3. Returns: None
4. Exceptions:
    a. Symbol declared multiple times
    b. The same name used as a variable and label both

2. File Name: pass_two.py

- Description :
    i. pass_two contains the function second_pass() that is used to execute the second pass of an assembler.
    ii. It reads the symbol table and instruction table from a file and assign a physical memory space to each instruction and variable, and writes into a final output file.
    iii. It uses translate_instruction(instruction, symbol_table) which translate instruction to 8 bit binary form using a get_binary_address(address) function.
- Functions :
    i. get_binary_address()
        1. Description: Returns the 8-bit binary address in string form from an integer address in decimal form
        2. Input: address - int
        3. Returns: binary address - string
    ii. second_pass()
        1. Description: Executes the second pass of the assembly process
        2. Input: None
        3. Returns: None
    iii. translate_instruction()
        1. Description: Function to Translate instructions in binary form.
        2. Input : instruction,symbol_table
        3. return : translated_instruction : String

3. File Name: Main.py
    - Description :

i. Its a wrapper class for the assembler.
ii. It imports the pass_one() and pass_two function and calls the two functions.
iii. It reports the errors on the console.

Comments

Comments after the assembly code are supported.
Syntax is : <Assembly Code> + “;” + <comments> (there should be a semi-colon to separte the comments from the assembly code.)

Errors Reported

- START Assembler directive not found.
- START Assembler directive supplied with the wrong number of operands.
- Address for START Assembler directive is of the Wrong Format.
- END Assembler directive not found.
- A symbol used as LABEL and VARIABLE both.
- A Label Declared multiple times.
- Ileagal OP-Code.
- OP-Code supplied with the wrong number of operands.
- A label used, but not Defined.
- STP Missing at the end of the code.

Input Format

- The assembly code should begin with the START assembler directive and end with the END assembler directive.
- The label should be given in the format LABEL: Instruction