ASSEMBLER

HOW TO RUN :

- Open terminal and go to the Assembler directory.
- Type “python3 code.py”.
- Symbol table, Opcode Table and Machine Code will be printed for the sample input.txt file in the folder.
- Also 3 files will be created in the same directory representing Symbol Table, Opcode Table and Machine Code.

DESCRIPTION OF THE FILE:

- The zip file consists of the Assembler Documentation, input.txt, code.py, opcode.txt, label.txt, machinecode.txt, opcodetable.txt.
- Input.txt consists of a sample input to generate a sample Machine code. The Assembly code which needs to be changed to the machine code should be pasted in the input.txt.
- Code.py contains the actual code to convert assembly code into machine code.
- Opcode.txt contains all the opcodes that have been defined. If you want to add a new opcode definition, then a row must be added to Opcode.txt file. Initially defined opcodes are shown at the bottom of this document.
- machinecode.txt shows the output for the sample input in the file.
- Label.txt shows all the labels.
- Opcodetable.txt shows all the opcodes provided.

WORKING:

- **OPCode table check :**
- The Opcode table is checked , and is checked if same instruction is assigned multiple OPCODES .If true error is raised
-
- **First Pass -**
  - The Symbol table is made from code
  - Every time a Label is encountered it is added in the symbol table
  - After the entire code is iterated over, the symbols are checked for definition , if not found , an error is raised (forward referencing error)
  -
- **Second Pass -**
  - Using the symbol table generated : we use function eval_line() to process every line
  - The function eval_line() is split in to multiple parts using th flow-chart describes in the slides :There are three types of the lines encountered:
    - MRI: Memory reference instructions :INP 157,BRZ L1

        - If the memory instruction refers to the a LABEL, it goes ahead and checks in the symbol table for the label , finds the binary address associated with it , and sets the last 8 bits of the Machine code
        - The first 4 bits are set as the opcode
- ORG Instructions: START X(Address to start LC at),END
    - If start : set LC to 100 : go to next line
    - If end : stop program
- Non MRI : CLA
    - Sets the first four bits as the opcode , and then uses
- Label definition : Checks for the symbol ":" used
        - Generates the machine code of the instruction defined after ":" , and stores it at the value provided by the LC

FILES GENERATED:

- label.txt will be generated which shows the symbol table.
- opcodetable.txt will be generated which shows the opcode table which consists of Opcodes used , their code , operand used with them and instruction length.
- machinecode.txt will be generated which shows the machine code as output.

ERRORS**:**

1)A symbol has been used but not defined

2)A symbol has been defined more than once

3)The name in the opcode field is not a legal opcode

4)An opcode is not supplied with enough operands

5)An opcode is supplied with too many operands

6)The START/END statement is missing

7)Invalid syntax

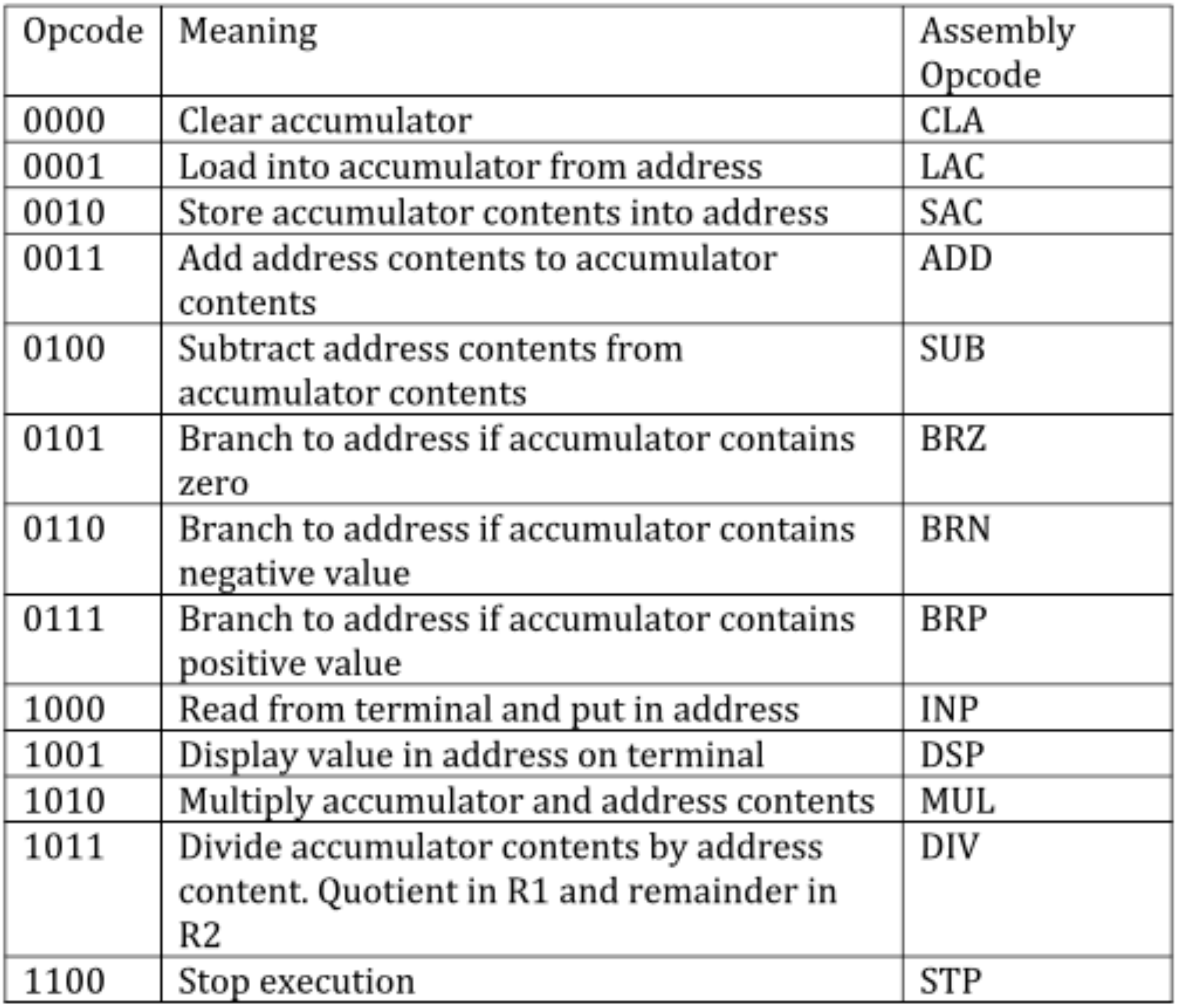
8) Variables not defined

ASSUMPTIONS:

- Code only has variables and Labels used in it

- Opcodes are defined only once
- Total virtual memory size does not exceed 2**8 bytes
- START and END compulsory
- For comments , start the statement with “#” or “//” or “/*”

OPCODE TABLE



Opcode	Meaning	Assembly Opcode
0000	Clear accumulator	CLA
0001	Load into accumulator from address	LAC
0010	Store accumulator contents into address	SAC
0011	Add address contents to 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 negative value	BRN
0111	Branch to address if accumulator contains positive value	BRP
1000	Read from terminal and put in address	INP
1001	Display value in address on terminal	DSP
1010	Multiply accumulator and address contents	MUL
1011	Divide accumulator contents by address content. Quotient in R1 and remainder in R2	DIV
1100	Stop execution	STP