Adrian Azan

Purpose:

This assembler takes in T34 instructions and interprets each instruction into its machine code format. The interpretation is done line by line. This assembler supports 56 instructions and 13 different addressing modes. Figure one is an example of an input file. Anything that comes after a '*' is ignored as a comment. All other instructions and labels are read in line by line and processed as such.

```
********************************
*      SAMPLE PROGRAM 1     *
********************************
*
     ORG  $F000
*
START    SEI
     CLD
     LDX  #$FF
     TXS
     LDA  #$00
*
ZERO     STA  $00,X
     DEX
     BNE  ZERO
     END
```
Figure 1

Several different dictionaries are used to match up instructions with machine code. These dictionaries are organized based on the first addressing mode of each instruction. For example, all instructions that have an implied addressing mode are in implied dictionary. The instructions are grouped in this way so that their base address (the first addressing mode) can be added to reach other addressing modes. For example the instruction ORA and EOR are both immediate and they both subtract 4 from the base machine code to be interpreted as a zero page addressing mode.

Labels such as START and ZERO (Figure 1) are also stored in dictionaries. The input file is scanned through twice. The first time is focused on finding bad opcode, duplicate labels, and executing pseudo instructions (ORG, EQU and END) and most importantly filling the symbol table with labels. After the first pass, the amount of bytes and file are reset for the second pass. In the second pass we are more focused on translating instructions to machine code.

The line of input is divided into several sections: a label, an instruction, and operand, and the opcode. The operand is searched for in the symbol table and is translated to its memory address if found. Afterwards the instruction is checked In the previously mentioned instruction dictionaries. Each instruction found is also checked for the addressing mode and changed accordingly before printing the line. The address and machine code are then printed to the screen along with the original line of input from the file.

After printing to the console, all machine instructions and the address they exist at are outputted to a file. The lables are sorted both alphabetically and numerically and those are printed out to the console. After printing to the screen another file is prompted for to repeat the process.

Usage:

The T-34 emulator takes an assembly file name <file_name.s>. It processes this file and outputs an object file <file_name.o> in the following format:

```
F000: 78
F001: D8
F002: A2 FF
F004: 9A
F005: A9 00
F007: 95 00
F009: CA
F00A: D0 FB
```

Adrian Azan
After the file is processed, you will be prompted for another file name to process. Enter any file name that does not exist in the directory to exit the program.

Functions:

commentCheck(line):

This will search the current line for comments (* or ;) and return everything up to that point

labelCheck(line):

Searches the designated column (0-9) for any numbers or letters and returns it as a label string

instructionCheck(line):

Searches the designated column (10-13) for any letters and returns it instruction string

operandCheck(line):

Searches the designated column (14-25) for anything that is neither a space nor a newline and returns that as a operand string

pseudoCheck(instruction):

Checks the instruction to see if it is any instruction which does not add to the PC count. Searches through an array of designated pseudo instructions. This list includes ORG, EQU, and END.

impliedCheck(instruction):

Checks if instruction is either an implied instruction or a branch instruction. No other addressing mode exists for these instructions so their base hex code is returned

immediateCheck(operand):

Checks if the operand matches with any of the addressing modes for immediate instructions. The addressing modes are checked by using the length of the operand along with special characters that might be found such as commas, or certain letters. If none of the checks match than there is an error with the operand syntax and a -100 is returned as an error flag.

Immediate3Check(operand):

Checks if the operand matches with any of the addressing modes for the subgroup of immediate instructions that only have 3 addressing modes. The addressing modes are checked by using the length of the operand along with the # symbol. If none of the checks match than there is an error with the operand syntax and a -100 is returned as an error flag.

Adrian Azan

Immediate5Check(operand):

Checks if the operand matches with any of the addressing modes for the subgroup of immediate instructions that only have 5 addressing modes. The addressing modes are checked by using the length of the operand along with the # or , symbol. If none of the checks match than there is an error with the operand syntax and a -100 is returned as an error flag.

zeroCheck(operand):

Checks if the operand matches with any of the addressing modes in instructions that begin with a zero page addressing mode. Operand is checked by using its length and if any special characters are inside. -100 is returned if no addressing mode matches operand

jumpCheck(operand):

Checks if the operand matches with any of the addressing modes in the jump instructions. The current way of checking is not as elegant as the others. Because of the way that labels are stored in hex in python, it is very difficult to check if a label is being used for the operand in a consistent way, so very specific ifstatements were used for error checking.

accumCheck(operand):

Checks if operand matches with any addressing modes for instructions whose base addressing mode is accumulator. Returns 0 if there is no operand

operandFormat(operand):

Because of the way that python stores its hex values, this function changes the operand to match with the format of the example outputs. The beginning 0x is removed from any hex value and then all numbers and letters are returned in a uppercase string format

badOpcode(instruct):

Checks if instruction is in any of the instruction arrays. If the instruction can not be matched with an existing one than True is returned.

operation(operand):

Checks if operand is an operation. Can add a label with a number, multiply a label with a number, and subtract a number from the label.

Known Issues:

The assembler will only work with operands that are in the HEX number format.

Operations can only be in the format of label <symbol> number and will only work with adding multiplying and subtracting.

CHK sadly does not work. I tried implementing it in several ways but could not get It right and unfortunately ran out of time