Adrian Azan

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
```

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.

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 ay 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 a 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