PROJECT 6

ASSEMBLER

A computer only understands 0 or 1. So we need a way to convert our human readible language to Binary.

Enters Assembler.

An assembler converts the human readible language to Binary. We use the same rules we used before in last project to create the assembler.

STEPS (High Level)

* Clean the file. Get rid of white spaces, Comments & empty lines.
* Handle A & C instructions. Convert number to binary if A instruction Or use C instruction table to convert instruction to Binary.
* Handle variables, Labels & symbols.

We will have 3 files.
Parser.Py -> Clean file
Instructions.Py -> Binary
Main.Py -> Handles both files.

Lets break each step down to smaller steps.

CLEANING THE FILE

We need to write code in our desired programming lang (I have used Python) & we should be able to work with any .asm file.

Small Steps

- Create an argument parser to handle CLI arguments & like an input.

- Take our file as input
- Take each line of the file & store it in a list.
- check for comments in each line & look for "//" keyword to identify comments.
- Delete the part after "//"
- Remove white spaces i.e. spaces from each line.
- Remove empty lines (These will be created when we delete comments)
- Create an array with all the required lines.
- Pass this array back to our "main" file.

As the code is big for a screenshot, you can look at the commented code on GITHUB.

ASSEMBLY → BINARY

This will be our second file "instructions.py"

STEPS

- Take the input from main.py (array given by the Parser)
- for each item in the array check if it starts with "@" symbol. If it does, its an A-instruction, else C-Instruction

A-INSTRUCTION

- replace the "@" symbol in the line with " " (empty (no space)) so that we can change it to an integer
- if the @"value" is not an integer, it will be variables, we [illegible] that later. Lets' just assume its int.

- Now we need to change this number to binary. Create a global binary array which will carry these values for each iteration & create a final array to store binary values which will be later saved as file.
- Create a function to convert decimal to binary, store it in binaryarray, now change the binary array[0] to 0, bcoz it's A-instr
- Now convert each integer in binaryarray to string, concatenate & store it in final array.

C-INSTRUCTION

- If the input is not a A-instruction, it's a C-Instruction.
- First we need to create a dictionary to store all the key value pair of C-Instruction

C-instruction specification

A dictionary for Computation

Symbolic syntax: *dest* = *comp* ; *jump*

Binary syntax: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

comp		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1
a==0	a==1						

dest	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

dictionary with dest & Binary keyval pair

jump	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Another dict for jump

Examples:

Symbolic: `M=1`

Binary: `1110111111001000`

- Now append binary array with [1, 1, 1] bcoz all C-Instructi starts with it.
- Split the input string using .split["="], and store it in array called Equal.
- Take Equal[1] and check if it contains ";", if it does, split the array into two and call it SemiColon.
- Take SemiColon[0] and pass it to Computation dict, get the binary values & append to binary array
- Take Equal[0], pass it to dest dictionary, append binary values to binary array
- Take SemiColon[1], pass it to jump dictionary, append binary values to binary array.
- Now convert each integer in binaryarray to string, concatenate it & store it in final array.

↑ do all this for each value in input array or line in the file.

HANDLE SYMBOLS

To handle symbols, we will use the technique suggest in the course. We will run the code in two passes to store the values in symbols dict. [Pass 1 (Handle Labels)]

- Create a symbols dict & store all the predefined symbols
- create a function called first pass & create a variable called counter (to keep track of lines)
- Loop through all the values in input array (which is basically each line of program), look for "(" left Paranthesis in the line.

↳ Left Paranthesis indicate a label.

- remove paranthesis from the line & store the string in the symbol dict as key & counter as value.
- if there is no paranthesis in the line, increase counter by 1.

[Handle variables (Pass 2)]

- create a function called Pass 2, loop through all input values (lines) & look for "@" symbol.
- temporarily remove "@" and check if it can be converted to int. if it can skip, else store this as key in symbol's dict.
- For the value, we need to create a global variable with value 16. if we have a variable use the global variable as value, and increment it by one.

THAT'S It. RUN Pass 1 & 2 before other functions, you should get right Ans.

Output final array to our main file & create a .hack file using the array.

FOR DETAILS & IMPLEMENTATION CHECK OUT CODE IN GITHUB


VOILA!