The 65py2 Assembler

A modern implementation for a classic processor - written in Python
@author - Jim DiCesare

Great name? Now how does it work?

- Who: Friendly neighborhood certificate student
- What: assembly file -> binary file
- Why an Assembler: First principles
- Why Python: I did not know Python, text processing
- Disclaimer: NOT vasm clone, similar but a lot less functionality (also did I mention I did not know Python before this project?)

Agenda:

- ➔ Requirements Gathering
- ➔ Design
- ➔ Implementation
- ➔ Testing
- ➔ Demonstration
- ➔ Questions

Overview/Requirements Gathering

- What is the thing?
- What does it need to be able to do?
- Read the data sheet
- Incremental development (for brevity speak to each phase overall)
  - 1. ~~Read ins, variables, numbers~~
  - 2. ~~Functions~~
  - 3. Jmp and jsr instructions
  - 4. Zp, inc addressing

Design

- RE for pattern matching
- How will it function:
  - Open file
  - Read into data structures
  - Write to binary file
- 5 files:
  - Main.py
  - Sreader.py
  - Ops.py
  - Opcodes.py
  - Jmp_ins.py -> jmp_ins class

- Limited compared to vasm
  - Must have 1 function
  - Can only jump between functions
  - No nested function

Implementation

- Data structures:
  - Opcodes dictionaries by addressing mode
  - Functions dictionary - dict of list of numbers
  - Program counter - list
  - Jmp_ins object
- 2 passes on assembly file
  - Variable assignment
  - Functions builder
- 2 passes functions dict
  - jmp/jsr format
  - Writing out

Testing

- Test Plan written for Q+A
- Unit testing
  - INCREMENTED as methods built/functionality added
  - Python's unittest framework
- Integration testing - bash script compares vasm output to mine
- System - load programs onto target processor

Demonstration

Questions?

https://github.com/dic3jam/65py2-Assembler