Natalia Gilbertson, Thomas Kercheval, Saam Amiri
3/16/2018
Disassembler: Test Plan

TEST FILE

```
68K_Disassembler/src/test_code/op_codes_to_test.x68
     ORG at $10000
Also, our disassembler itself! Most ops are supported. (not
DBF)
     ORG at $1000
```

Testing Stages

1. Design Testing
2. Module Testing
3. Integration Testing

Design Testing

We tested our design by reviewing it thrice before coding. Further testing was done on our design by referring to it while implementing. Anything wrong with our design documents was changed when a bug was found.

Module Testing

Due to our modular approach, we were able to test each module independently by mocking the interface of other modules. I/O tested by incrementing the address register by the amount that Op-Codes would increment it for a NOP operator, then printing out addresses for a line.

Opcodes tested by mocking the buffer and address to read from IO, then determining the opcode to print, adding it to the buffer (waiting to print), setting the format flag (for EA to properly decode the rest of it).

EA tested by mocking the information given by Op Codes and decoding the effective address (writing the proper ASCII bytes to the output buffer for output by I/O).

Integration Testing

After we integrated our modules, we used an extensive testing suite to test every available EA mode on every supported opcode (>700 tests). We worked through every opcode to make sure that they were processed correctly. Several bugs were found this way (and were promptly fixed).

Our test file is located in “68K_Disassembler/src/test_code/” and is titled “op_codes_to_test.x68”. We execute “io_NataliaGilbertson.x68” to begin our

disassembler, then loaded our test file into the 68K memory. The test file is ORG's at 10000, where we begin disassembling, then specify to go to some large value, so that we run every test.

Validation of our disassembler was done by using the opcode spreadsheet (provided with out flowcharts) to ensure that every bit was tested. Removing ambiguity from any potential opcodes trying to sneak in.

We ORG'd our code at different points in memory to make sure that everything worked correctly. The first time that we did this, it broke our program. Since we were using Address Indirect with Indexing, our displacement values became too large (16bits).

Robustness testing was done by loading various programs that we have written over the quarter and seeing how the disassembler performed on them. Our program's performance was satisfactory for what we tested.