It’s not a bug, It’s a feature

By the name of Skyrim and Bethesda

# Overview

Within this section, I will try my best to tell you all of the possible “features” within the disassembler that I am aware of

# Displaying of immediate hex data

For the 68k disassembler, everything is seen as hex. Input such as decimal, binary, hex are all converted to hex. So, within my disassembler, I will out put all the immediate data as a hex value

For example, #5 will be output as #$0005 or #$00000005. Depending on the data mode of the op code (byte, word, long). Byte and word only output 4 at a time. Where as long will yield the later result

# Parsing and reaching ending address

If the starting and ending address are relative to each other. Then the program will be able to check when the starting address reach the ending address

For example: The starting address is 0000 7000. And the program will end at 00007050.

If you entered 0000 7000 and ending address 00007050 (or anything lesser than that but still greater than starting address), the program will be able to detect that and display a prompt

If you entered the ending address a mile away from starting. Let’s say 00008000. Then once it reaches the first FFFF, then it will stop without a prompt

That’s because the instructions decode 4 nibbles at a time, when it reach FFFF, or opcode start with F, it will stop because there are no opcode that start with F

However, For opcode that deals with printing out data that includes F, like printing out the address FFFF3658, then it’s no problem since the program already know it as a data that need to print out

So: If an invalid instruction is at the end of the source code that deal with F, it won’t be able to recognize that F is just a data instead of end of program. Then it will stop

I’ve been thinking of multiple ways to do this but have not yet reach a conclusion. Because who knows how many bits forward the data of an invalid instruction will be.

# Automatic conversion

Within the program, some command may automatically convert to another version. For example, SUB might automatic convert to SUBQ when dealing with immediate data.

Another interesting aspect is within the test file I included

OR D3, 0(A3,D5)

Since this deals with unsupported EA mode, It will not process the destionation

It will print out OR D3, DATA $whatever the opcode is

And the next 4 nibbles 5000, which was supposed to be the data of this statement, was interpreted as ADDQ #8,D0 because of the first nibble

# Limit range of branch

For branching such as BSR, BRA,Bcc I made it so that it will print out the actual address, instead of displacement. Currently, it can only print 4 nibbles at most, instead of the full 8 nibbles. This can cause issues of another branch statement that deals with value outside of 4 nibbles was executed because it won’t know that it need to get the next 4 nibbles

# 

# Similar structure of opcode

There are many instructions that have the similar structure that I have no idea how to differentiate. And can only hope to not see them while disassembling

For example: ADD and ADDX. They both have similar opcode that I don’t know how to separate them. Or SUB and SUBX