CSS 422: Hardware and Computer Organization

Disassembler – Exceptions Report

Team – Single Precision REEE

# Bugs or Defects

## AND (Unsupported)

There seems to be a very odd buffer mutilation that occurs using **AND.X Dn, ABS** or **AND.X ABS, Dn** where **ABS** refers to any absolute long, and **Dn** any data register. **X** is for whatever size field. When **CUR\_OP\_CODE** (current 4 hex instruction set) is set to some sort of **AND** instruction, it loops fine for the first time (exception of **AND.W $00004900,D6** where some sort of crash occurs). After the loop, it mutilates the next read instruction set. For example, if **MOVEQ** is set as the next instruction set, you’d expect **$7XXX** to appear. Instead after running say **AND.L D2, $00004000** the next instruction set seems to now reflect **$0000** which is then resolved to **ORI** due to misidentification. This appears to be an **IO** issue as OP and EA modes run without issue until the next word.

## ADDQ (Unsupported)

There is a similar buffer mutilation that seems to occur same as **AND** where instead of mutilating it to **$0000** it’s mutilated to **$4900**. The specific command is **ADDQ.B #8, $8000** which seems to do buffer mutilation. This appears to be an **IO** issue as OP and EA modes run without issue until the next word.

## BCLR

From the GoldenCrystal handbook on size specifications for OP codes, it’s claimed that BCLR can have a size of a byte or a long. However, when a byte size for BCLR is tested, it will result in an invalid size addressing mode. Based on the 7-6 typical size bits, BCLR should be always a size of long given the bits are 10.

## BRA

Unable to test BRA for a 32-bit displacement, found 3/14/2019. We were unable to force the BRA OP code instruction set to ever get $60FF which it claims is the 32-bit displacement. We were however able to get it 1 bit before 32-bit with the max value of $6000 7FFF before the EASy68k Simulator would claim the branch instruction displacement is out of range or invalid.

This is documented in the Motorola 68k Handbook (Page 4-55) where BRA is claimed to be able to use $60XX (where XX is 8-bit displacement if it’s not $00 or $FF) or $6000 for a 16-bit displacement, or $60FF for a 32-bit displacement. We were unable to force a test for a 32-bit displacement for BRA ever, even with specifying BRA.L as a sign specifier.

We found that if we forced BRA to jump an odd byte (such as $1) we could get $60FF, however it’s clear that when just jumped say a single byte, that’s not a long or 32-bit.

## MOVEM

Print formatting for MOVEM has issues regarding the huge amounts of variations for MOVEM prints. It seems like the EA print that does work is regarding **MOVEM.X Dx-Dy/Ax-Ay,An/Dn** would work however that’s a complex case and the EA role derived the rest of MOVEM from that complex case. Unfortunately this has issues with other prints such as a simple address reversal of **MOVEM.X Ax-Ay/Dx-Dy,An/Dn** would break the print formatting. It seems that this issue is due to a slight misidentification in the ordering of registers and prints. While the general registers and mode are recognized the way they are printed to the buffer is incorrect.

# 