Lab Exercise Number 06

68000 Integer Arithmetic with Error Detection

Lab Partners:

Gabriel Stroe

Mostapha Baydoun

**Academic Honor Code:**

*"I have neither given nor received unauthorized aid in completing this work, nor have I presented someone else’s work as my own"*

Date Submitted: 3/31/2016

# Introduction:

The purpose of this lab is to focus on the proper usage of the easy68k to detect Integer Arithmetic error. We became very familiar with the TrapV command where it showed how to detect if an overflow had happened. We also got familiar with data types and the condition codes. Now, we know that detection of certain procedures like overflow or divide by zero or if a data type conversion from a word to a long and vise versa, condition codes are the ones for us look

# Required Resources:

|  |  |
| --- | --- |
| Lab Resource Identification: | |
| Easy68k Assembler | v5.15.04 |

# Lab Description:

This lab is an extension of the previously lab. In the previous lab we had a few mathematical functions that had certain conditions where they would not return the results. In this lab we addressed those issues by using exception handlers. These exception handlers would notify the users when there is an error. The exceptions we looked for was divide by zero, overflow when adding, and a conversion error when dividing numbers (the divs function only returns 16 bit answers even with 32 bit registers).

# Setup and Procedure:

The setup involved taking our lab five code and coping that into a new file for lab six. Next we needed to add the divide by zero exception. This was very easy to do as the processor has an internal flag for divide by zero so we only had to had a location to go to and display the error message for this one.

For the overflow error we had to create the error message. Originally we expected to use the TRAPV to check for an overflow when adding and multiplying. The problem is overflow in those operations set the carry and extend bit not the overflow bit. So we used a branch to sub routine method and checked if the carry bit was set.

For the conversion overflow. Since this error code only happen when dividing as the 68k cannot return DIVS results that are longer than 16 bits, we use TRAPV to check and see if the result returned is smaller than the actual calculation. If it is the overflow bit is set and TRAPV goes to the exception handler.

# Results:

We were able generate an output message whenever Arithmetic error has occurred, overflow, divide by zero and a conversion overflow in this lab. A divide by zero explains itself when we divide by zero. Overflow when there is an arithmetic operation that outputs a datatype different than the input and the Carry condition code is set. A conversion overflow, a change in data type from a Long to a word and from a word to a long.

# Conclusion:

This lab helped us understand how to detect Arithmetic errors if occurred and how to alert the user that those errors have occurred. We also got more familiar with condition codes and what do they represent and what can interpreted from their status. This lab took us the longest so far about 3 weeks and a half. We had some problems at the beginning where we were able to alert the user that a divide by zero has happened but for the conversion overflow, but after some research we were able to purposely make a conversion overflow in our program and outputted a message that says that happened.