Experiment No. 3
EXCEPTION PROCESSING and SYSTEM CONTROL
ECE 441

Peter Chinetti

September 19, 2013

Date Performed:	September 19, 2013
Partners:	Zelin Wu
Instructor:	Professor Saniie

1 Introduction

1.1 Purpose

The purpose of this experiment is to acquaint the student with the following topics:

- 68000 Exception Processing
- TUTOR Exception Handling
- 68000 System Controls

1.2 Background

6-1 of the Motorola User Manual gives a good background on the material in this lab. According to that section 'The exception processing state is associated with interrupts, trap instructions, tracing, and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing provides an efficient context switch so that the processor can handle unusual conditions. The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.'

2 Lab Procedure and Equipment List

2.1 Equipment

- SANPER System
- Computer with TUTOR software

2.2 Procedure

Execute each sample program and record data when requested.

3 Results, Analysis and Discussion

3.1 Sample Program 3.1.A

3.1.1 Source

```
MOVE.W D0,A1       ;Load D0 with the odd address
MOVE.W D1,(A1)+    ;Cause exception as an odd address is accessed
BRA $2000          ;Infinite loop
```

3.1.2 Why did the Address Trap occur?

The address traps occur 'when the processor attempts to access a word or long-word operand or an instruction at an odd address' according to 6-19 of the M68K user manual. In this particular example, D0 is loaded with 0xFF, which is the number 255 in decimal. Clearly, 255 is an odd address, so it generates the error.

3.2 Sample Program 3.1.B

3.2.1 Source

```
MOVE.B $FFFFFF,D0    ;Attempt to load a non-existent address
BRA $2000            ;Infinite loop
```

3.2.2 Why did the Bus Trap occur?

According to Wikipedia,
There are two main causes of bus errors:

- Non-existent address
- Unaligned access

This particular Bus Trap was generated in the non-existent address mode, specifically when the CPU tried to access memory address 0xFFFFFF.

3.3 Sample Program 3.1.C

3.3.1 Why did the Illegal Instruction occur?

According to 6-14 of the M68K user manual,
'Illegal instruction is the term used to refer to any of the word bit patterns that do not match the bit pattern of the first word of a legal M68000 instruction. If such an instruction is fetched, an illegal instruction exception occurs.'
Furthermore:
'Three bit patterns always force an illegal instruction trap on all M68000-Family-compatible microprocessors. The patterns are: $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for Motorola system products. The third pattern, $4AFC, is reserved for customer use (as the take illegal instruction trap (ILLEGAL) instruction).'

3.4 Sample Program 3.1.D

3.4.1 Source

```
ANDI.W #$0700,SR   ; Attempt to AND to Status Register
BRA $2000          ; Infinite Loop
```

3.4.2 Why did the Privilege Violation occur?

According to the M68k user manual (6-15), ANDing an immediate value to the Status Register is a priviliged instruction. When that code was executed, the processor was running in user mode, so that caused the Privilege Violation.

3.5 Sample Program 3.1.E

3.5.1 Source

```
DIV D1,D2 ; Divide by zero
BRA $2002   ; Infinite loop (on this instruction)
```

3.5.2 Why did the Divide by Zero Exception occur?

According to the M68k user manual (6-14), 'A signed divide (DIVS) or unsigned divide (DIVU) instruction forces an exception if a division operation is attempted with a divisor of zero.' In this example, D1 contained zero and the execution of DIVU forced the error.

3.6 Sample Program 3.1.F

3.6.1 Source

```
CHK.W D6,D7 ; Check to see if 0x3010 [D7] < 0x3000 [D6]
BRA $2002   ; Infinite loop
```

3.6.2 Why did the CHK Instruction Exception occur?

According to the 68K instruction set, 'The contents of the low-order word in the data register specified in the instruction are examined and compared with the upper bound at the effective address. The upper bound is a two's complement integer. If the data register value is less than zero or greater than the upper bound contained in the operand word, then the processor initiates exception processing.' In this case, an exception was raised as the value being checked was too high.

3.7 Sample Program 3.1.G

3.7.1 Why did the LINE 1010 Emulator Exception occur?

According to 6-14 of the M68K user manual,
'Word patterns with bits 1512 equaling 1010 or 1111 are distinguished as unimplemented instructions, and separate exception vectors are assigned to these patterns to permit efficient emulation.'

3.8 Sample Program 3.1.H

3.8.1 Why did the LINE 1111 Emulator Exception occur?

According to 6-14 of the M68K user manual,
'Opcodes beginning with bit patterns equaling 1111 (line F) are implemented in the MC68020 and beyond as coprocessor instructions. These separate vectors allow the operating system to emulate unimplemented instructions in software.'

3.9 Sample Program 3.2

3.9.1 Source for Procedure 1

```
    ORG     $950
START:                      ; first instruction of program
    MOVE.L #$2000,A5
    MOVE.L #$201A,A6
    MOVE.B #227,D7
    TRAP #14
    MOVE.B #228,D7
    TRAP #14
```

3.9.2 Source for Procedure 5

```
MOVE.B $50000,D0 <CR> ;Incite bus error
BRA $1000             ;Infinite loop
```

3.9.3 Source for Procedure 12

```
    ORG     $950
START:                  ; first instruction of program

* Put program code here
    MOVE.L #$2000,A5      ;Init regs
    MOVE.L #$201A,A6
    MOVE.B #227,D7        ;Print string
    TRAP #14
    MOVE.W (A7)+, D0      ;Move SSW from stack to D0
    MOVE.W #232,D7        ;Convert to ASCII encoded HEX
    TRAP #14
    MOVE.B #$20, (A6)+    ;Put in a space
    MOVE.L (A7)+, D0      ;Move BA from stack to D0
    MOVE.W #230, D7       ;Convert to ASCII encoded HEX
    TRAP #14
    MOVE.B #$20, (A6)+    ;Put in a space
    MOVE.W (A7)+, D0      ;Move IR from stack to D0
    MOVE.W #232,D7        ;Convert to ASCII encoded HEX
    TRAP #14
    MOVE.B #$20, (A6)+    ;Put in a space
    MOVE.W #227, D7       ;Print string
    TRAP #14
    MOVE.W #228, D7      ;Return to TUTOR
    TRAP #14


    END     START         ; last line of source
```

3.9.4 Functions and features useful in a Bus Error Exception routine

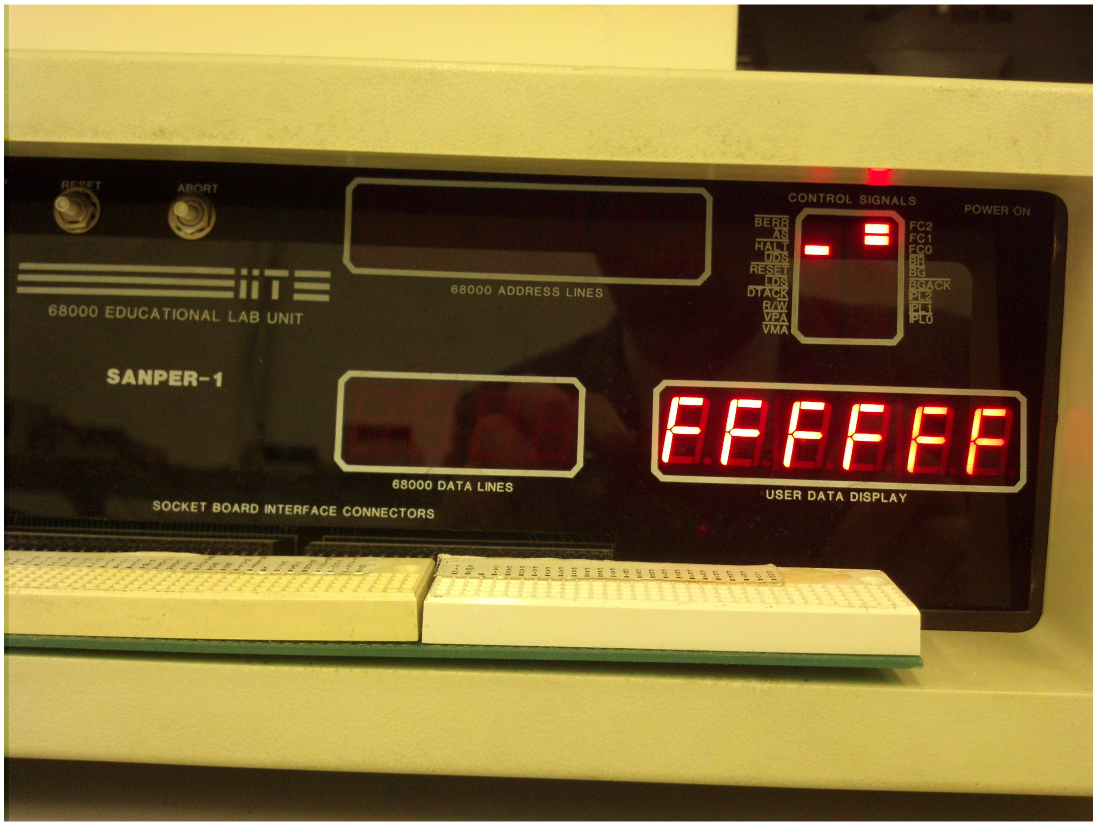
Either the ability to debug, namely printouts of location and context of the error as well as dropping into TRACE mode, or a good way to clean up the error and continue through it.

3.9.5 Why didn't 'A BUS ERROR JUST OCCURRED' appear on the screen after the program was executed a second time?

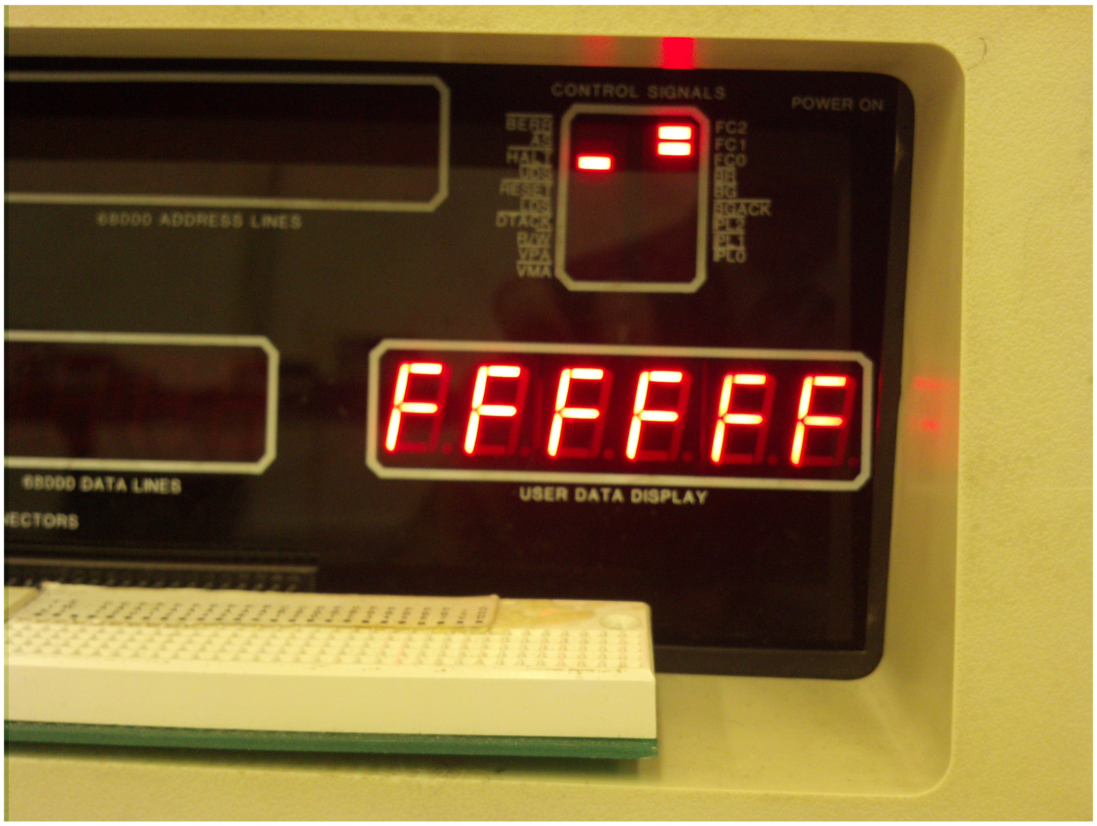
The Reset switch was depressed, so the Bus Error Vector was reset to default values, not the address of the custom routine.

3.10 Sample Program 3.3

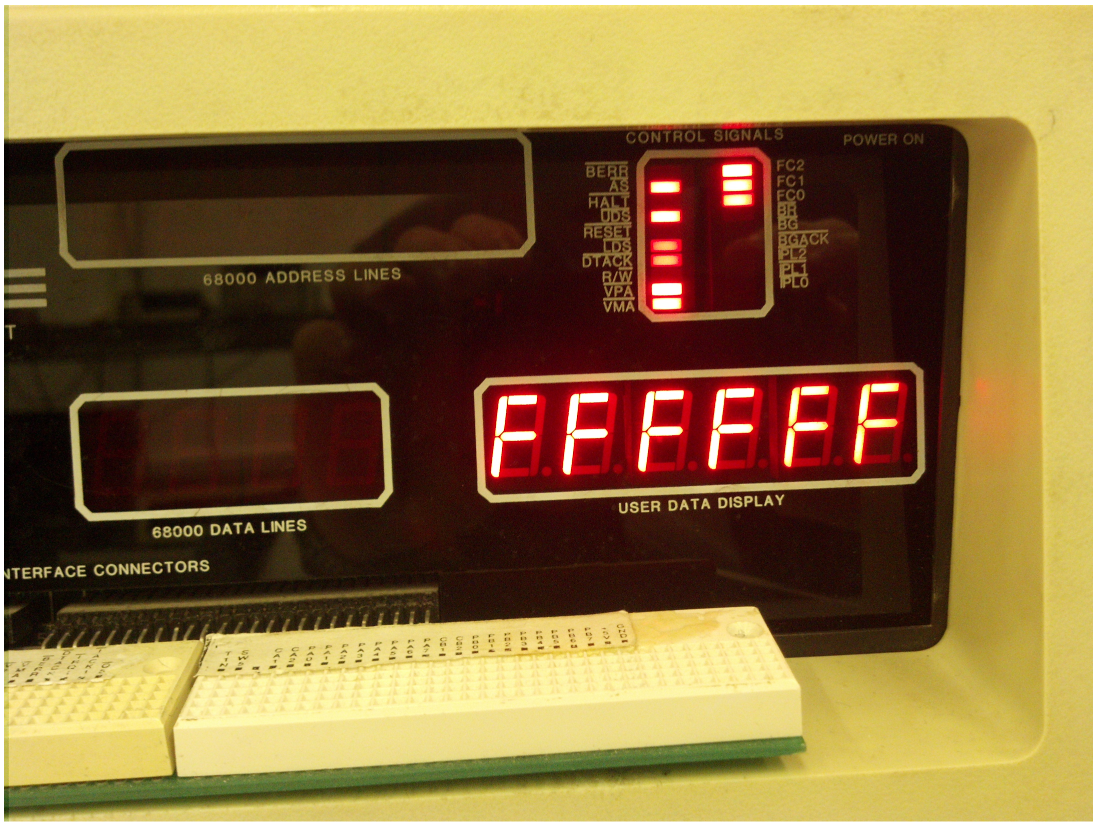
3.10.1 Photos



After Halting



After Pushing 'Abort' notice: no change



After Pushing 'Reset' notice: resets to normal operation

3.10.2 Source for Procedure 1

```
MOVE.B $50000,D0 <CR>  ;Incite bus error
BRA $1000              ;Infinite loop
```

3.10.3 What halted the processor?

First, the program from procedure one generated a Bus Error. When the Bus Error was generated, the CPU tried to access the recovery routine stored at address 8.$Earlierintheprocedure, wehadmodifiedtheaddresstobecome$FFFFFE. When the processor tried to load that address, it generated a Second bus error. Thinking that it was broken, the processor HALTed.

3.10.4 What (in general) halts the processor?

Double bus error are either generated by faulty error handling code (a problem), or hardware problems (a big problem).

3.10.5 What does a double bus fault do to the processor's HALT signal

It engages the signal, completely stopping execution on the CPU, but also preventing further damage.

3.10.6 Difference between 'Reset' and 'Abort' buttons

The abort button simply aborts the currently running code and returns to TUTOR. The Reset button aborts the code, resets the processor and resets the vector tables.

3.10.7 Difference between 'Reset' button and 'RESET' instruction

According to the 68K instruction set when the RESET instruction is processed: 'The reset line is asserted, causing all external devices connected to the 68000's RESET* output to be reset. The RESET instruction is privileged and has no effect on the operation of the 68000 itself. This instruction is used to perform a programmed reset of all peripherals connected to the 68000's RESET* pin.' The 'Reset' button actually affects the processor.

4 Conclusions

This experiment was accomplished. TUTOR was introduced, as well as M68k instructions. From this building block, students can work on more and more complex programs for SANPER and can continue to learn about the functioning of the machine.