CECS 525-01

Fall 2011

Project 2

Name	Report (20 Points)	Demo (10 Points)	Questions (10 Points)
Group 1 Nick Searcy: Luke Spicer Adrian Fletcher Conor Heine			

Motorola 68000 Programming

[Group 1: Adrian Fletcher, Conor Heine, Nick Searcy, Luke Spicer]

CECS 525-01

Abstract

The team developed a familiarity with the custom firmware written by the textbook author for the Motorola 68000, making changes to the firmware, loading and running user programs from within the firmware, developing and leveraging common firmware tasks. Additionally, the team garnered experience with the Renesas M16C QSK by developing an application to leverage the system hardware. In the lab, the team engineered a log-on routine that disallowed access to the loaded firmware until a valid user-name and password are entered. The team implemented exception vectors and trap calls in the EEPROM-based firmware to handle common errors generated in or by code as well as to facilitate the development of future software. The team also included a help routine that, after log on, allows user to view a list of available commands and a short description of their function(s). Finally, the team designed an application to leverage the M16C to measure the current temperature.

Body

Part 1:

The modified firmware differs from the original firmware in 17 locations. In addition, the line endings were all modified to be Windows (CRLF).

In the first changes, the program origin, stack vector, exception vector, ACIA control vector, and data vector are each moved. Also, the auxiliary ACIA vector is equated with the console ACIA control vector. Next, the monitor program is changed to display the modified banner lines which are added in a later change. Next, the ACIA constants are changed. Next, all BTST.B operations are changed to BTST. Next, the X_SET subroutine is essentially removed.

Monitor Source Analysis:
Note - Routines, functions, and sections written and/or modified by the students are discussed in Part 2 below and - thus - not discussed here.

Header:

The head of the MONITOR source sets the exception handling vectors to the appropriate locations so that they may be loaded into ROM when the chips are programmed. Directly below this, space is allocated for ROM-specific variables based upon their specified sizes.

Reset:

This location is where the firmware resorts to upon reset. The data area is saved in A6 and is used in the rest of the software as a reference offset for all variables. It then sets up the monitor for displaying a welcome message and command prompt before requiring the user to log in. After successful log in, the warm entry point is reached, and it is here that the user can type in commands to execute.

SET_ACIA:

Sets the 6850 UART Control Register, initializing serial communications.

NEWLINE:

Prints a newline to the screen.

PSTRING:

Assumes the precondition that the string (in the form of a sequential array of characters) pointed to by A4 is null-terminated. Iterates through the characters including and following A4 until a null character is reached, displaying each character to the screen.

HEADING:

Same as PSTRING followed by NEWLINE

GETLINE:

Assumes that A3 points to the next free entry in the line buffer, A2 points to the end of the line buffer, A1 points to the head of the line buffer, and D0 holds the character to be stored (GETCHAR dependent). This function gets a character input from the user, stores the character into memory, increments the store pointer and necessary address pointers, and repeats this process until the enter key is pressed.

TIDY:

Reads the input from the line buffer while ignoring any spaces until the end of line is found. A0 points to the line buffer and A1 points to the tidied up line.

EXECUTE:

After a command line input is tidied, an attempt is made to match the first entry in the line buffer with an available command in the user table (if it exists). If the user table doesn't exist, then the COMTAB is tried. If the comtab doesn't exist, then the absolute address of the command is attempted.

HEX, BYTE, WORD, LONGWD, PARAM:

Gets one, two, four, or eight hexadecimal characters, or a long word from the line buffer and puts it in D0. Sets bit 0 of D7 if input error occurs.

PARAM:

Reads a parameter from the line buffer and stores it in D0 and PARAMTR(A6). Bit 1 of D7 set on error.

OUT1X, OUT2X, OUT4X, OUT8X:

Outputs 1, 2, 4, or 8 hexadecimal characters to the screen. The character to be output is in D0.

JUMP:

Causes execution to begin at the address in the line buffer.

MEMORY:

Shortcut: "MEM"

Displays and/or modifies the contents of memory.

LOAD:

Loads hex data in "S" format from serial communications line.

DUMP:

Transmits hex data in "S" format to host computer.

TM:

Enters transparent mode.

SET_DCB, IO_REQ, CON_IN, CON_OUT, AUX_IN, AUX_OUT, BUFF_IN, BUFF_OT, IO_OPEN, EX_DIS, BR_GET, RESTORE:

Associated IO functions that are specific to firmware functions. They have not been modified or used explicitly by the students yet.

GETLINE:

Gets a line of text from the user by iterative calls of GETCHAR until the enter key is pressed.

GETCHAR:

Gets a character from the user, either echoes, doesn't echo, or echoes stars based on the value of the echo flag variable, and stores the input in D0.

Footer:

Fixed variables and string declarations. Also, the list of available commands in the COMTAB.

	Original	Modified

1	`*  TSBUG2 - 68000 monitor - version of 23 July 1986`	`*  TSBUG2 - 68000 monitor - version of 05 January 2010`
		
		`  ORG  $0`
		`  DC.L  STACK,RESET`

2	`STACK  EQU  $00000800  Stack_pointer`
	`ACIA_1  EQU  $00010040  Console ACIA control`
	`ACIA_2  EQU  ACIA_1+1  Auxilary ACIA control`
	`X_BASE  EQU  $08  Start of exception vector table`
		`STACK  EQU  $4400  Stack_pointer`
		`ACIA_1  EQU  $8001  Console ACIA control`
		`ACIA_2  EQU  ACIA_1  Auxilary ACIA control`
		`X_BASE  EQU  $4000  Start of exception vector table`

3	`DATA  EQU  $00000C00  Data origin`	`DATA  EQU  $4800  Data origin`

4	`*`
	`  ORG  $00008000  Monitor origin`
	`  DC.L  STACK  Reset stack pointer`
	`  DC.L  RESET  Reset vector`
	`RESET  EQU  *  Cold entry point for monitor`
		`*`
		`  ORG  $1000  Monitor Origin`
		`RESET:  EQU  *`

5	`  BSR.S  SETACIA  Setup ACIAs`
	`  BSR  X_SET  Setup exception table`
	`  BSR  SET_DCB  Setup DCB table in RAM`
	`  LEA.L  BANNER(PC),A4  Point to banner`
	`  BSR.S  HEADING  and print heading`
	`  MOVE.L  #$0000C000,A0  A0 points to extension ROM`
		`  BSR.S  SETACIA  Setup ACIAs`
		`  BSR.L  X_SET  what change did i do here`
		`  BSR.L  SET_DCB  Setup DCB table in RAM`
		`  BSR.S  NEWLINE`
		`  LEA.L  BANNER(PC),A4`
		`  BSR.S  PSTRING`
		`  BSR.S  NEWLINE`
		`  LEA.L  MODIFY(PC),A4`
		`  BSR.S  PSTRING`
		`  BSR.S  NEWLINE`
		`  LEA.L  WHY(PC),A4`
		`  BSR.S  PSTRING`
		`  BSR.S  NEWLINE`
		`  LEA.L  WHERE(PC),A4`
		`  BSR.S  PSTRING`
		`  BSR.S  NEWLINE`
		`  LEA.L  ADDRE(PC),A4`
		`  BSR.S  PSTRING`
		`  BSR.S  NEWLINE`
		`  MOVE.L  #$3000,A0  A0 points to extension ROM`

6	`  MOVE.B  #$15,(A0)  Set up ACIA1 constants (no IRQ,` ` MOVE.B #$15,1(A0) RTS* low, 8 bit, no parity, 1 stop)	`  MOVE.B  #$19,(A0)  Set up ACIA1 constants (no IRQ,` ` MOVE.B #$19,1(A0) RTS* low, 8 bit, no parity, 1 stop)
7	`  BTST.B  #0,D7  Test for input errors`	`  BTST  #0,D7  Test for input errors`

8	LOAD4 BTST.B #3,D7 Test for checksum error	LOAD4 BTST #3,D7 Test for checksum error
9	BTST.B #0,D1 Test RDRF	BTST #0,D1 Test RDRF
10	BTST.B #0,D1 Test RDRF bit (any input?)	BTST #0,D1 Test RDRF bit (any input?)
11	BTST.B #0,D2 and poll ACIA until next char received	BTST #0,D2 and poll ACIA until next char received
12	CON_OT3 BTST.B #1,D1 Repeat	CON_OT3 BTST #1,D1 Repeat
13	BTST.B #3,D7 D7(3) set if open error	BTST #3,D7 D7(3) set if open error
14	BTST.B #6,D0 Test input for lower case	BTST #6,D0 Test input for lower case
15	X_SET LEA.L X_BASE,A0 Point to base of exception table MOVE.W #253,D0 Number of vectors - 3 X_SET1 MOVE.L #X_UN,(A0)+ Store uninitialized exception vector DBRA D0,X_SET1 Repeat until all entries preset SUB.L A0,A0 Clear A0 (points to vector table) MOVE.L #BUS_ER,8(A0) Setup bus error vector MOVE.L #ADD_ER,12(A0) Setup address error vector MOVE.L #IL_ER,16(A0) Setup illegal instruction error vect MOVE.L #TRACE,36(A0) Setup trace exception vector MOVE.L #TRAP_0,128(A0) Setup TRAP #0 exception vector MOVE.L #BRKPT,184(A0) Setup TRAP #14 vector = breakpoint MOVE.L #WARM,188(A0) Setup TRAP #15 exception vector MOVE.W #7,D0 Now clear the breakpoint table	X_SET MOVE.W #7,D0 Now clear the breakpoint table
16	BANNER DC.B 'TSBUG 2 Version 23.07.86',0,0 CRLF DC.B CR,LF,'?',0	BANNER DC.B 'TSBUG 2 Version 23.07.86',0,0 MODIFY DC.B 'Modified by Eugene A. Rockey Jr. Jan.05.2010',0,0 WHY DC.B 'for use with the M68000 Minimal Computer Configuration',0,0 WHERE DC.B 'University of Louisville',0,0 ADDRE DC.B 'CECS 525, www.cs.louisville.edu',0,0 CRLF DC.B CR,LF,'TS2MON >',0
17	END $8000	END RESET

Part 2:

Help code:

```
HElP    LEA.L   HELPPAGE,A1      Load Starting Address of Array of HELP Strings
HELP1   MOVE.L  (A1)+,A4         Post increment to next HELP string label, store in A4
        BSR     PSTRING          Print string pointed to in A4
        BSR     NEWLINE          Print NewLine
        TST.L   (A1)             Test to see if we're at end of HELP label array
        BEQ     ENDHELP          If yes (then end)
        BRA     HELP1            No continue to print next label and HELP string
ENDHELP RTS
```

Help Table and Index:

```
HELPPAGE DC.L
HJUMP,HMEM1,HMEM2,HLOAD1,HLOAD2,HDUMP1,HDUMP2,HTRAN,HNOBR1,HNOBR2,HDISP,HGO1,HGO2,HBRGT,HPLAN,HKIL
L,HGB,HREG1,HREG2,HHELP,$00000000
HJUMP    DC.B     ' JUMP <address> causes execution to begin at <address>.',0,0
HMEM1    DC.B     ' MEMORY <address> examines contents of <address>',0,0
HMEM2    DC.B     '     and allows them to be changed.',0,0
HLOAD1   DC.B     ' LOAD <string> loads S1/S2 records from the host.',0,0
HLOAD2   DC.B     '     <string> is sent to host.',0,0
HDUMP1   DC.B     ' DUMP <string> sends S1 records to the host and is',0,0
HDUMP2   DC.B     '     preceeded by <string>',0,0
HTRAN    DC.B     ' TRAN enters the transparant mode and is exited by ESC,E.',0,0
HNOBR1   DC.B     ' NOBR <address> removes the breakpoint at <address> from the',0,0
HNOBR2   DC.B     '     BP table. If no address is given all BPs are removed.',0,0
HDISP    DC.B     ' DISP displays the contents of the pseudo registers in TSK_T.',0,0
HGO1     DC.B     ' GO <address> starts program execution at <address> and loads',0,0
HGO2     DC.B     '     regs from TSK_T.',0,0
HBRGT    DC.B     ' BRGT puts a breakpoint in the BP table - but not in the code.',0,0
HPLAN    DC.B     ' PLAN puts the breakpoints in the code.',0,0
HKILL    DC.B     ' KILL removes breakpoints from the code.',0,0
HGB      DC.B     ' GB <address> sets breakpoints and then calls GO.',0,0
HREG1    DC.B     ' REG <reg> <value> loads <value> into <reg> in TASK_T.',0,0
HREG2    DC.B     '     Used to preset registers before a GO or GB.',0,0
HHELP    DC.B     ' HELP displays the user available functions with instructions.',0,0
```

The help subroutine refers to an index of all the help pages. It loads the first address of the first help line, stores this address in A4, calls the print string function, loads the address of the next help line, and repeats until the final entry in the index, which is 0. The print string function takes care of displaying character strings by using the put character function.

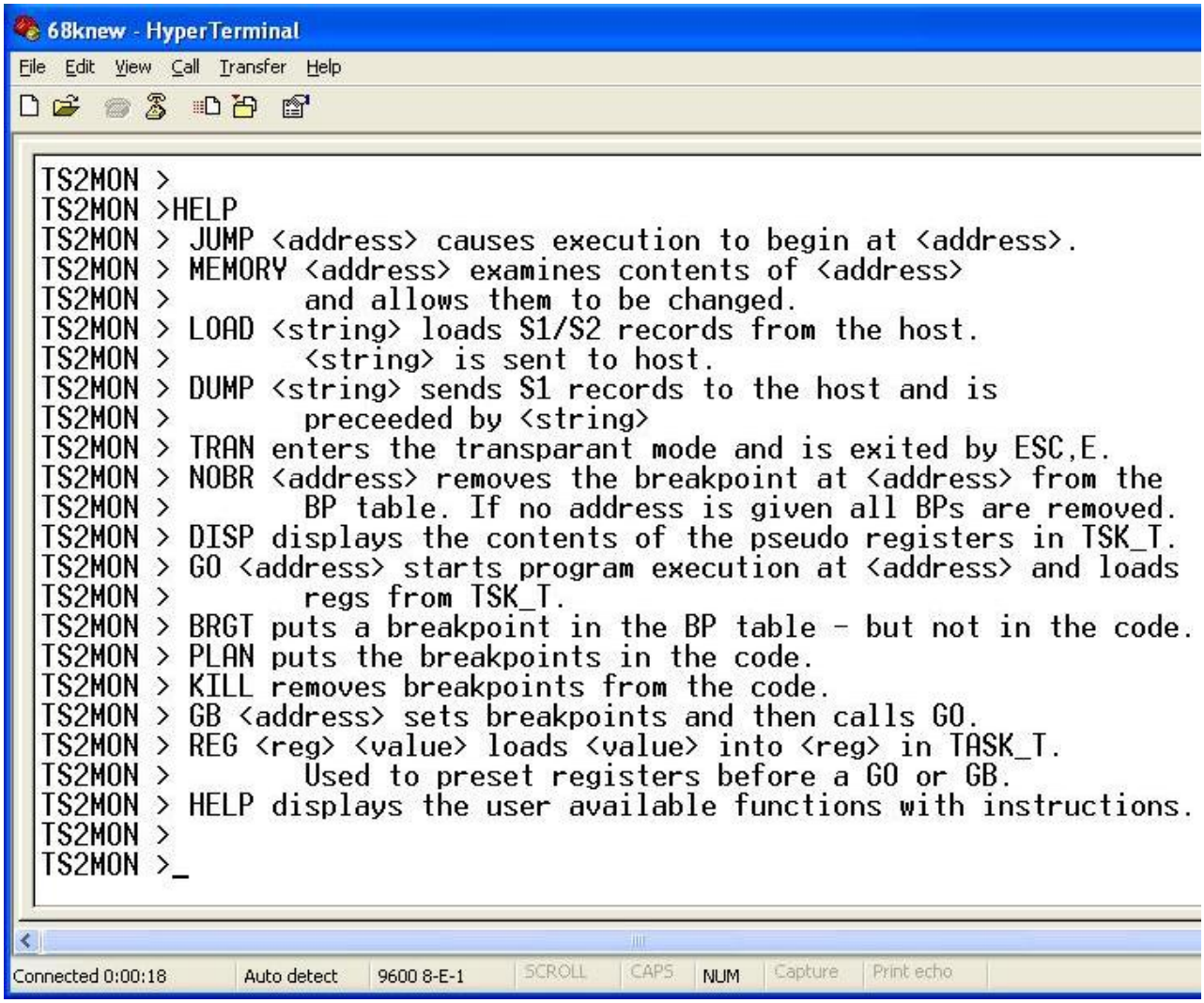

Figure 1: Demonstration of help command.

Login code:

```
LOGIN   EQU             *
        MOVE.L  #0,D5           Use D5 to store the number of incorrect attempts
ASKUN   BSR     NEWLINE
        LEA.L   ASKUNAME,A4     Ask for a username
        BSR     PSTRING
        CLR.B   ECHO(A6)
        BSR     GETLINE         Get username
        LEA     VUNAME,A4       Load the address of the valid username to A4
        BSR     VALID8          Validate the username
        CMP     #1,D0           Is it valid?
        BEQ     PWO             If so, ask for the password
        BSR     NEWLINE         Otherwise, ask for them to try again
        LEA.L   INVUN,A4
        BSR     PSTRING
        ADDQ    #1,D5           Increment the number of incorrect attempts
        CMP     #3,D5           Have we reached max attempts?
        BNE     STOK
        BSR     LOCKUP          If the number of incorrect attempts is 3, lock up
STOK    BRA     ASKUN
PWO     MOVE.L  #0,D5           Reset incorrect attempts - 3 tries each @ username and pw
```

```
ASKPW   BSR     NEWLINE
        LEA.L   ASKPWORD(PC),A4
        BSR     PSTRING         Ask for a password
        MOVE.B  #2,ECHO(A6)
        BSR     GETLINE
        CLR.B   ECHO(A6)
        LEA     VPWORD,A4       Load the address of the valid password to A4
        BSR     VALID8          Validate the password
        CMP     #1,D0           Is it valid?
        BEQ     ALLGOOD         Then, we've successfully authenticated
        BSR     NEWLINE         Otherwise, ask for them to try again
        LEA.L   INVPW,A4
        BSR     PSTRING
        ADDQ    #1,D5           Increment the number of incorrect attempts
        CMP     #3,D5           Have we reached max attempts?
        BNE     STOK2
        BSR     LOCKUP          If the number of incorrect attempts is 3, lock up
STOK2   BRA     ASKPW
ALLGOOD RTS

LOCKUP  EQU     *
        BSR     NEWLINE
        LEA.L   LOCKED(PC),A4   ; Tell the user they've been locked out
        BSR     PSTRING
LOCKLP  NOP                     ; Infinite loop
        BRA     LOCKLP
        RTS                     ; Just to be safe

VALID8  EQU             *
        MOVE.L  #1,D0           Set D0 to true to start
VLOOP   CMP.B   #0,(A4)         Check to see if character in string is null
        BNE     NOTNULL         If its not null, continue
        CMP.B   #0,(A1)         Check to see if the user input is newline
        BRA     DONE            If it is, then we are done, return true...
NOTNULL CMP.B   (A1)+,(A4)+     Compare the characters and postincrement
        BNE     SETF            If they're not equal, set false flag
        BRA     VLOOP           Loop
SETF    MOVE.L  #0,D0           Sets our D0 flag to false
DONE    RTS
```

The login routine makes use of `getline` and `pstring` in order to display and to receive input. The valid username has been written previously in the source code as a character string. The address of the valid username is loaded into A4 and then passed to the validation subroutine.

The validation subroutine verifies each character of the user entered username against the valid username. The validation subroutine exits when there is a character mismatch after the validation flag is set false. Alternatively, the subroutine exits after the null character terminates the valid username character string (this exit leaves the validation flag true).

The login routine then performs the same function on the valid password and user password.

If either validation subroutine exits with the validation flag set false, the login program enters lockup which is just a `bra` and a `nop`.

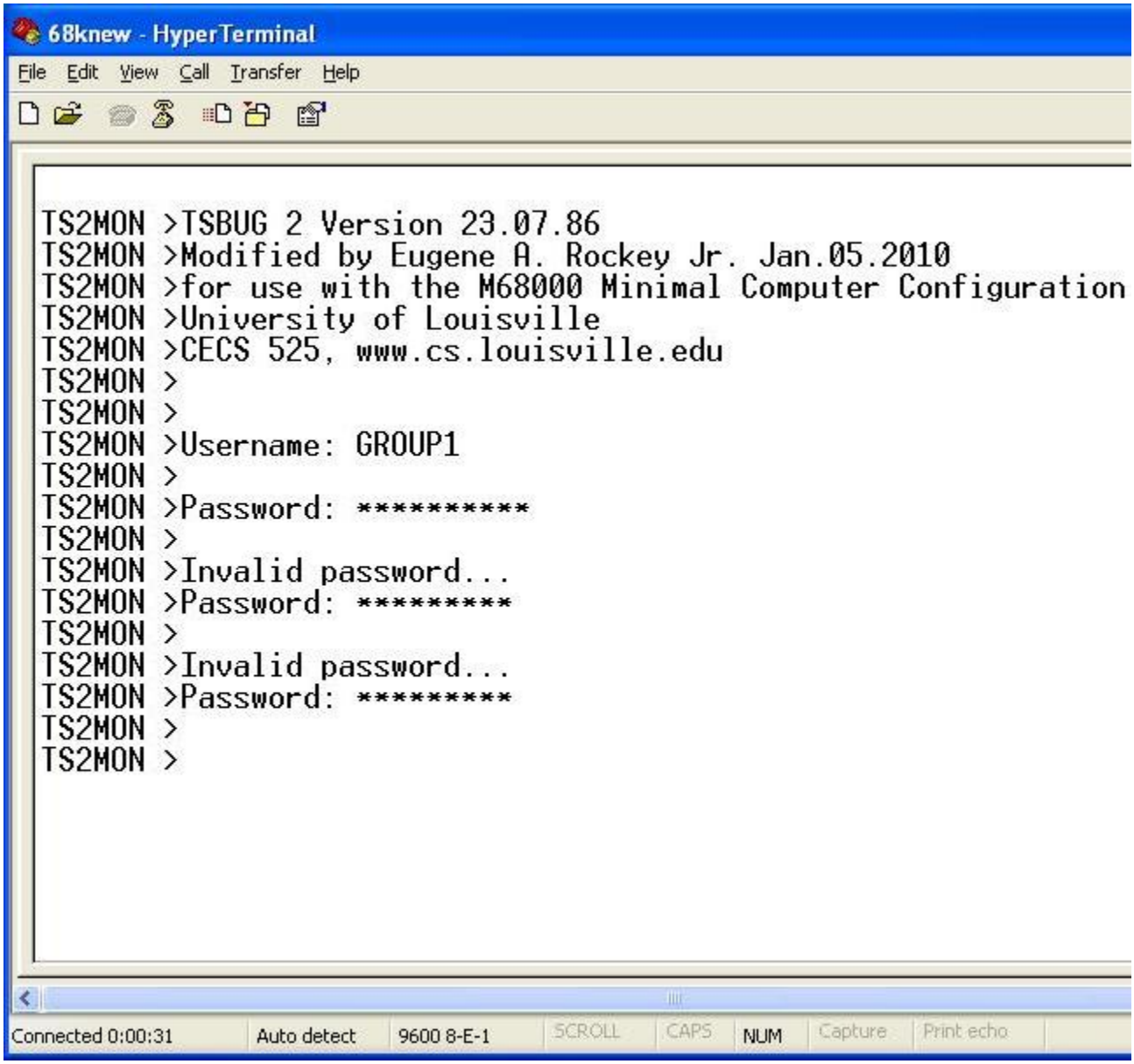

Figure 2: Demonstration of successful password entry on third try.

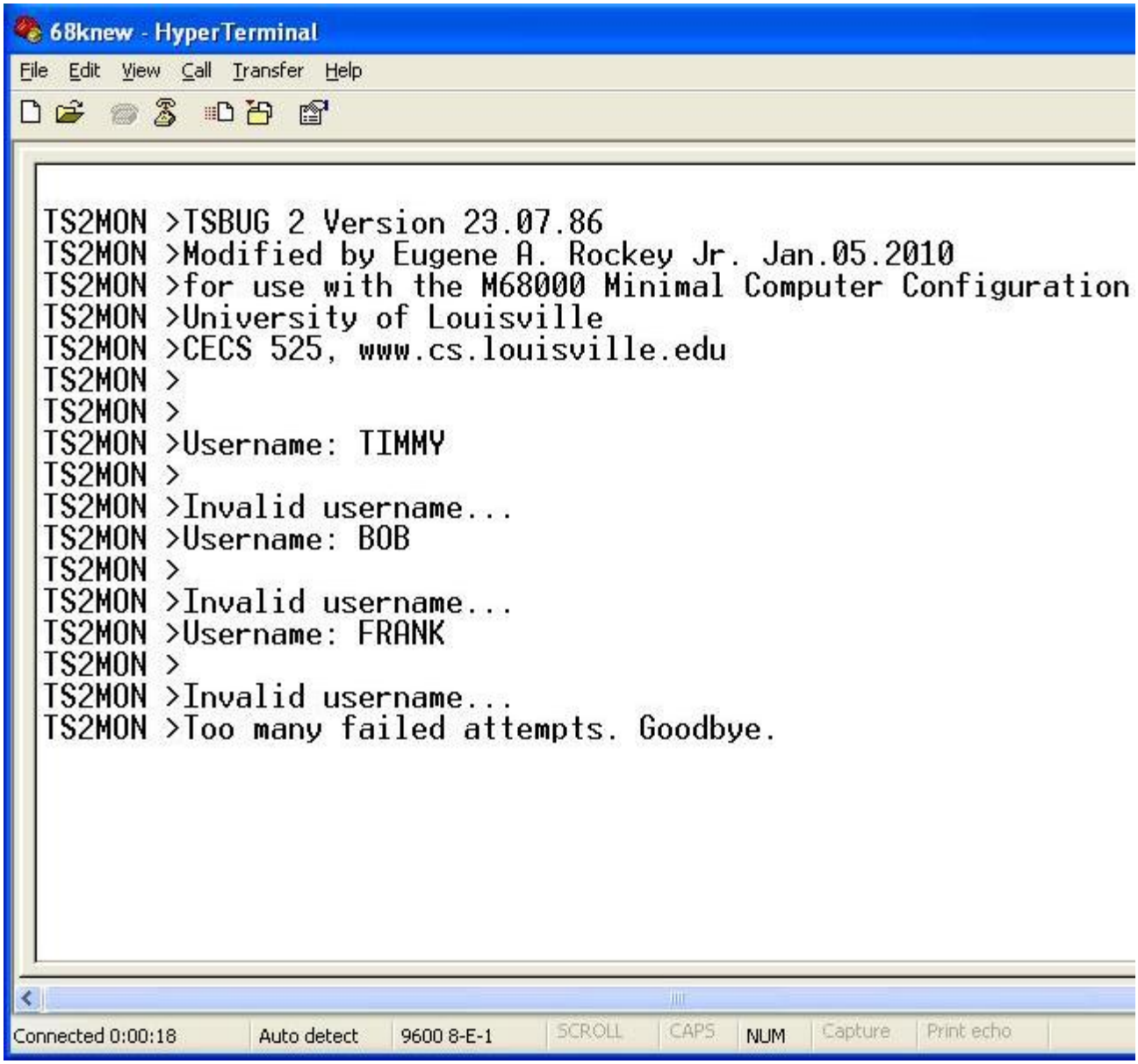

Figure 3: Demonstration of Username lockup.

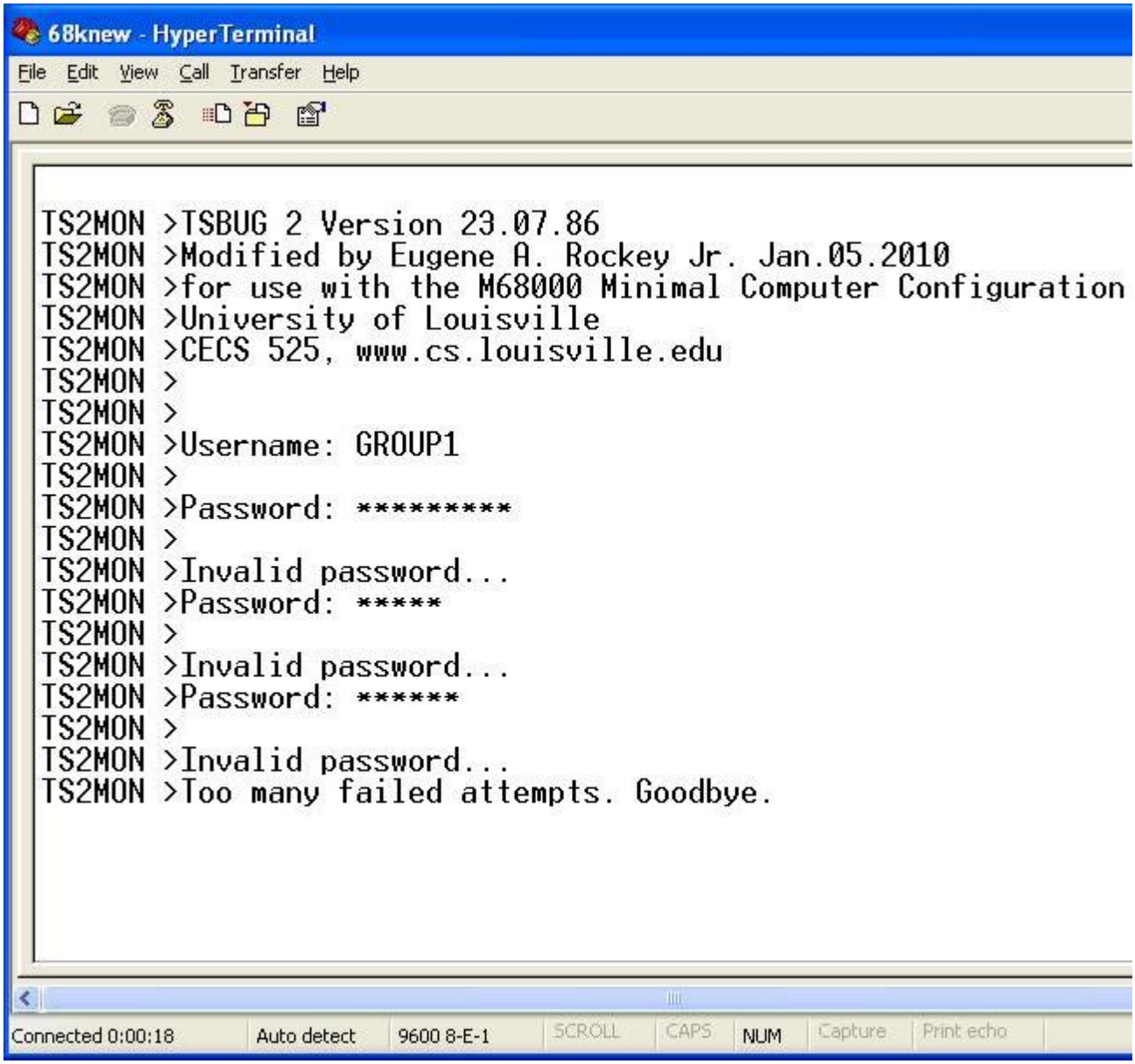

Figure 4: Demonstration of password lockup.

Part 3:

Vector Labels:

```
ORG       $0000
DC.L      STACK,RESET
DC.L      BUS_ER,ADD_ER,IL_ER,DIV0_ER *,X_UN,X_UN,PRV_ER,X_UN     *$0008 through $0027
ORG       $0020
DC.L      PRV_ER
ORG       $0080
DC.L      TRAP_0
ORG       $00B8
DC.L      BRKPT,WARM
```

The labels prv_er (privilege error) and div0_er (divide by zero error) are unique vector labels for two new exception handling routines added by the team in lab 2. Each print an error message specific to the exception and then are handled as a group 1 or group 2 exception.

Exception handlers added in Lab 2:

```
PRV_ER  EQU      *              Privilege Violation Error (group 1) exception
        MOVE.L   A4,-(A7)Save A4
        LEA.L    MES13(PC),A4    Point to error message string
        BSR      HEADING         Pring the error message as a heading
        MOVE.L   (A7)+,A4Restore A4
        BRA      GROUP1          Deal with group 2 exception
*
DIV0_ER EQU      *              Divide by zero error (group 2) exception
        BSR.S    GROUP2  Treat as group 2 exception
        MOVE.L   A4,-(A7)Save A4
        LEA.L    MES14(PC),A4    Point to error message string
        BSR      HEADING         Pring the error message as a heading
        MOVE.L   (A7)+,A4Restore A4
        BSR      EX_DIS          Display saved registers
        BRA      WARM            Return to monitor
```

Firmware routines:

```
TRAP_0  EQU     *                User links to  TS2BUG via TRAP #0
        CMP.B   #0,D1            D1 = 0 = Get character
        BNE.S   TASK1
        BSR     GETCHAR
        RTE
TASK1   CMP.B   #1,D1            D1 = 1 = Print character
        BNE.S   TASK2
        BSR     PUTCHAR
        RTE
TASK2   CMP.B   #2,D1            D1 = 2 = Newline
        BNE.S   TASK3
        BSR     NEWLINE
        RTE
TASK3   CMP.B   #3,D1            D1 = 3 = Get parameter from buffer
        BNE.S   TASK4
        BSR     PARAM
        RTE
TASK4   CMP.B   #4,D1            D1 = 4 = Print string pointed at by A4
        BNE.S   TASK5
        BSR     PSTRING
        RTE
TASK5   CMP.B   #5,D1            D1 = 5 = Get a hex character
        BNE.S   TASK6
        BSR     HEX
        RTE
TASK6   CMP.B   #6,D1            D1 = 6 = Get a hex byte
        BNE.S   TASK7
        BSR     BYTE
        RTE
TASK7   CMP.B   #7,D1            D1 = 7 = Get a word
        BNE.S   TASK8
        BSR     WORD
        RTE
TASK8   CMP.B   #8,D1            D1 = 8 = Get a longword
        BNE.S   TASK9
        BSR     LONGWD
        RTE
```

```
TASK9   CMP.B   #9,D1            D1 = 9 = Output hex byte
        BNE.S   TASK10
        BSR     OUT2X
        RTE
TASK10  CMP.B   #10,D1           D1 = 10 = Output hex word
        BNE.S   TASK11
        BSR     OUT4X
        RTE
TASK11  CMP.B   #11,D1           D1 = 11 = Output hex longword
        BNE.S   TASK12
        BSR     OUT8X
        RTE
TASK12  CMP.B   #12,D1           D1 = 12 = Print a space
        BNE.S   TASK13
        BSR     PSPACE
        RTE
TASK13  CMP.B   #13,D1           D1 = 13 = Get a line of text into
        BNE.S   TASK14           the line buffer
        BSR     GETLINE
        RTE
TASK14  CMP.B   #14,D1           D1 = 14 = Tidy up the line in the
        BNE.S   TASK15           line buffer by removing leading
        BSR     TIDY             leading and multiple embeded spaces
        RTE
TASK15  CMP.B   #15,D1           D1 = 15 = Execute the command in
        BNE.S   TASK16           the line buffer
        BSR     EXECUTE
        RTE
TASK16  CMP.B   #16,D1           D1 = 16 = Call RESTORE to transfer
        BNE.S   TASK17           the registers in TSK_T to the 68000
        BSR     RESTORE          and therefore execute a program
        RTE
TASK17  CMP.B   #17,D1            D1 = 17 = Call Fibinacci Sequence Generator
        BNE.S   TASK18
        BSR     FIBGEN
TASK18  RTE
```

The team added task 17 a way to call a Fibonaci sequence generator program. This application was a user application in project 1 which could be loaded into RAM. By adding it to the firmware it has become a function which can be accessible from any future user application through the use of the trap #0 call.

```
FIBGEN  LEA.L   FMES1(PC),A4     Point to first of FibGen Instruction strings, store in A4
        BSR     PSTRING                 Print string pointed to in A4
        BSR     NEWLINE                 Print NewLine
        LEA.L   FMES2(PC),A4     Point to second of FibGen Instruction strings, store in A4
        BSR     PSTRING                 Print string pointed to in A4
        BSR     NEWLINE                 Print NewLine
        LEA.L   FMES3(PC),A4     Point to third of FibGen Instruction strings, store in A4
        BSR     PSTRING                 Print string pointed to in A4
        BSR     NEWLINE                 Print NewLine
        LEA.L   FMES4(PC),A4     Point to fourth of FibGen Instruction strings, store in A4
        BSR     PSTRING                 Print string pointed to in A4
        BSR     NEWLINE                 Print NewLine

FIBLOOP BSR     NEWLINE                 Print NewLine
        LEA.L   FMES5(PC),A4     Point to user input prompt string, store in A4
        BSR     PSTRING                 Print string pointed to in A4

        BSR     GETCHAR          Get a character from input device
        SUB.B   #$30,D0          Convert to binary

        CMP     #0,D0                   if user chose "0" exit and end
        BEQ     FIBEND
        BSR     NEWLINE                 Print newline
```

```
        CMP     #7,D0                    Compare the user input (in DO)
        BGT     BADINPUT         to 7, if greather than, they gave bad input
        CMP     #0,D0                    Compare the user input (in D0)
        BLT     BADINPUT         to 0, if less than, they gave bad input

        MOVE.B  #0,FN2(A6)               intialize fibonacci counters
        MOVE.B  #1,FN1(A6)               Fn-2 = 0, Fn-1 = 1
        MOVE.B  D0,D3                    COUNT = D0 (the users choice of lines)
        MOVE.B  #'0',D0                  print the first character of the sequence
        BSR     PUTCHAR


        CMP.B   #2,D3                    if COUNT < 2 then COUNT = 1
        BLT     FIBLOOP         we're done so ask for input again

        MOVE.B  #' ',D0         print the second character of the sequence
        BSR     PUTCHAR         and a space
        MOVE.B  #'1',D0
        BSR     PUTCHAR

        CMP.B   #2,D3                    If count == 2, we're done so ask for input a again
        BEQ     FIBLOOP


        SUBI.B  #2,D3                    subtract 2 from the count the user gave us
*to make up for the first 2 numbers in the sequence already being printed
FIBFOR  CMP.B   #0,D3           If D3 == 0, then we've printed the entire seqence
        BEQ     FIBLOOP                  So ask for input from the user again
        SUBI.B  #1,D3           If D3 != 0, then decrement D3 and print next value in sequence

        MOVE.B  #' ',D0                  display a blank space between number
        BSR     PUTCHAR                  using PUTCHAR

        CLR     D0                               Clear D0
        ADD.B   FN2(A6),D0               add FN2 to FN1
        ADD.B   FN1(A6),D0               store this sum in D0 to display
        ADD.B  #$30,D0                   Max number an ASCII character
        BSR     PUTCHAR                  Display character
        SUB.B   #$30,D0                  convert ASCII back to value

        MOVE.B  FN1(A6),FN2(A6)  FN2= previous FN1
        MOVE.B  D0,FN1(A6)               FN1= prevous fibonacci number


        BRA     FIBFOR                   Branch always to the start of the For loop

BADINPUT BSR    NEWLINE                  Print newline
        LEA.L   FMES6(PC),A4    Move pointer to bad input error message
        BSR     PSTRING                  Print string pointed to in A4
        BSR     NEWLINE                  Print NewLine


        BRA     FIBLOOP                  User entered bad data so ask for input again
FIBEND   BSR    NEWLINE                  Print newline
        RTS
```

The following code was removed from the monitor program because it was unnecessary. The x_set subroutine and the x_base were used in the original monitor program as a way to load the addresses labels of the various exception handlers into their proper exception vectors. This program is unnecessary for use with our minimal computer system because the memory addresses of the exception vectors have been mapped to EEPROM (ROM) which cannot be written to during program execution. The x_set routine was therefore removed and replaced by ORG statements and DC.W calls which hard-coded the exception handler labels into their respective exception vectors.

```
X_SET   LEA.L   X_BASE,A0          Point to base of exception table
        MOVE.W  #253,D0            Number of vectors - 3
```

```
X_SET1  MOVE.L  #X_UN,(A0)+       Store unitialized exception vector
        DBRA    D0,X_SET1         Repeat until all entries preset
        SUB.L   A0,A0             Clear A0 (now points to base of exception table again)
        MOVE.L  #BUS_ER,8(A0)     Setup bus error vector
        MOVE.L  #ADD_ER,12(A0)    Setup address error vector
        MOVE.L  #IL_ER,16(A0)     Setup illegal instruction error vector
        MOVE.L  #DIV0_ER,20(A0)   Setup divide by zero error vector
        MOVE.L  #PRV_ER,32(A0)    Setup privilege violation error vector
        MOVE.L  #TRAP_0,128(A0)   Setup TRAP #0 exception vector
        MOVE.L  #BRKPT,184(A0)    Setup TRAP #14 vector = breakpoint
        MOVE.L  #WARM,188(A0)     Setup TRAP #15 exception vector
X_SET   MOVE.W  #7,D0             Now clear the breakpoint table
        LEA.L   BP_TAB(A6),A0     Point to table
X_SET2  CLR.L   (A0)+             Clear an address entry
        CLR.W   (A0)+             Clear the corresponding data
        DBRA    D0,X_SET2         Repeat until all 8 cleared
        RTS
```

The following user program was loaded with the use of the monitor program into RAM at address $4C00. Using the monitor command JUM 4C00 allowed the user program to be run, which allows the user to select from a variety of exections/trap calls to force on the sytem. Through the use of this user program the exception handler routines and added trap 0 task were tested and verified.

```
                ORG     $4C00
START                                   * first instruction of program
TOP             LEA.L   INSTR(PC),A4    * Print instructions
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1
                TRAP    #0
                LEA.L   E1(PC),A4
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1
                TRAP    #0
                LEA.L   E2(PC),A4
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1
                TRAP    #0
                LEA.L   E3(PC),A4
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1
                TRAP    #0
                LEA.L   E4(PC),A4
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1
                TRAP    #0
                LEA.L   E5(PC),A4
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1
                TRAP    #0
                LEA.L   E6(PC),A4
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1
                TRAP    #0
                LEA.L   E7(PC),A4
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1
```

```
                TRAP    #0

                CLR.L   D0              * Clear D0 so it can take user input
                MOVE.L  #0,D1           * Get user input
                TRAP    #0
                SUBI    #$30,D0         * Convert ASCII number to integer value

                CMP.B   #1,D0
                BNE             NOT1
                MOVE.L  #17,D1          * FIBGEN task
                TRAP    #0
NOT1            CMP.B   #2,D0
                BNE             NOT2
                TRAP    #15                     * Test trap 15
NOT2            CMP.B   #3,D0
                BNE             NOT3
                TRAP    #14                     * Test trap 14
NOT3            CMP.B   #4,D0
                BNE             NOT4
                MOVE.W  #1,A4           * Test Address Error Exception
                MOVE.W  #3,(A4)
NOT4            CMP.B           #5,D0
                BNE             NOT5
                ILLEGAL                         * Test illegal instruction exception
NOT5            CMP.B   #6,D0
                BNE             NOT6
                MOVE.L  #10,D3          * Move 5 into D3
                DIVU    #0,D3           * Divide 10 by the contents of D3 (which is 5)
                BRA             $1000           * Returns to command line
NOT6            CMP.B   #7,D0
                BNE             NOT7
                STOP    #2                      * Test Privilege Viloation Exception
NOT7            MOVE.L  #2,D1
                TRAP    #0
                LEA.L   INV,A4          * Invalid input
                MOVE.L  #4,D1
                TRAP    #0
                MOVE.L  #2,D1           * Task 2
                TRAP    #0                      * of Trap 0 Prints a newline
                MOVE.L  #2,D1           * Task 2
                TRAP    #0                      * of Trap 0 Prints a newline
                BRA             TOP

* Variables and Strings

INSTR   DC.B    'Which exception would you like to induce?',0
E1              DC.B    '1: Trap0 Task17 - Fibonacci task',0
E2              DC.B    '2: Trap15 - (Warm)',0
E3              DC.B    '3: Trap14 - (Breakpoint)',0
E4              DC.B    '4: Address Error',0
E5              DC.B    '5: Illegal Instruction',0
E6              DC.B    '6: Divide by 0',0
E7              DC.B    '7: Privilege Violation',0
INV             DC.B    'Invalid input... try that again',0

        END     START                           * last line of source
```

Because this program is loaded external to the firmware, it can benefit from using the routines coded into the firmware and accessed through the tasks of trap 0. This way, a new routine to get a line of user input does not have to be written for every application. This application uses task 4 to print a string in the program and task 2 print a newline.

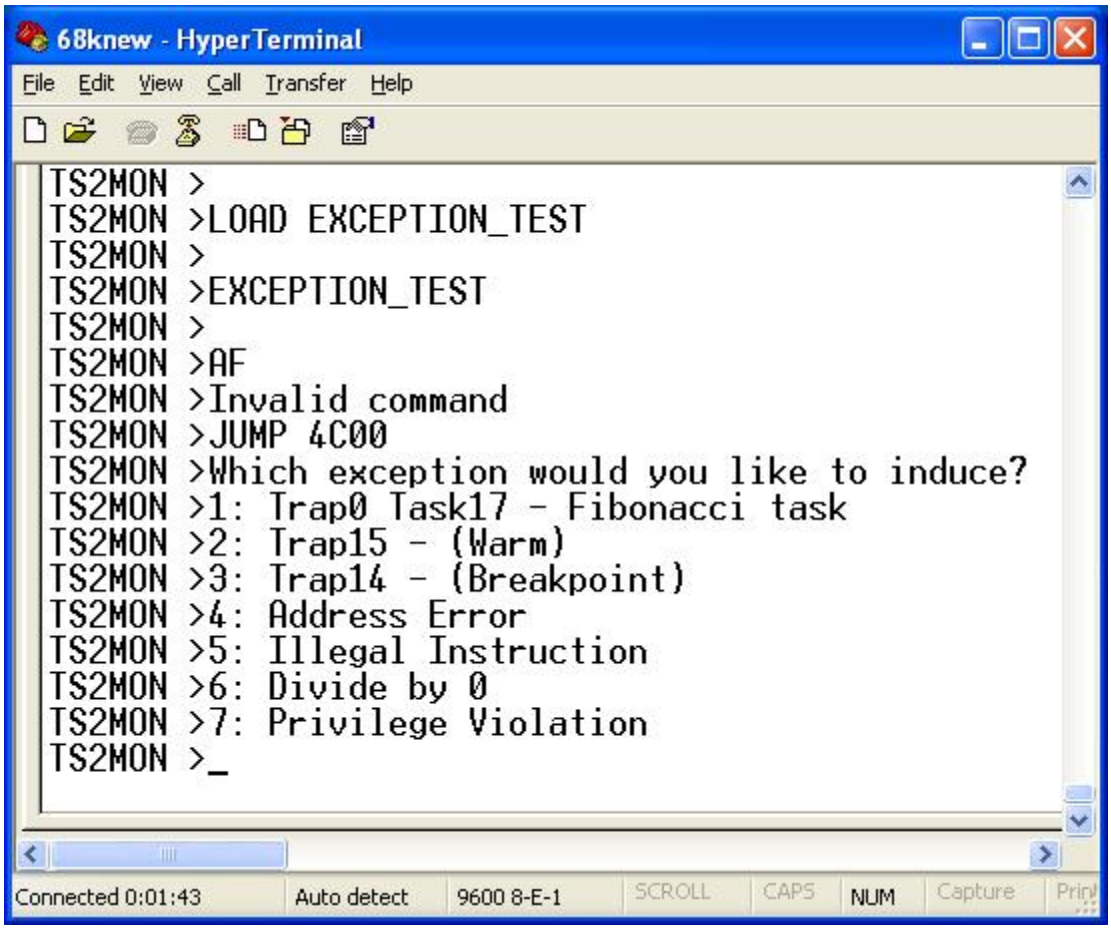

Figure 5: Menu of exception application.

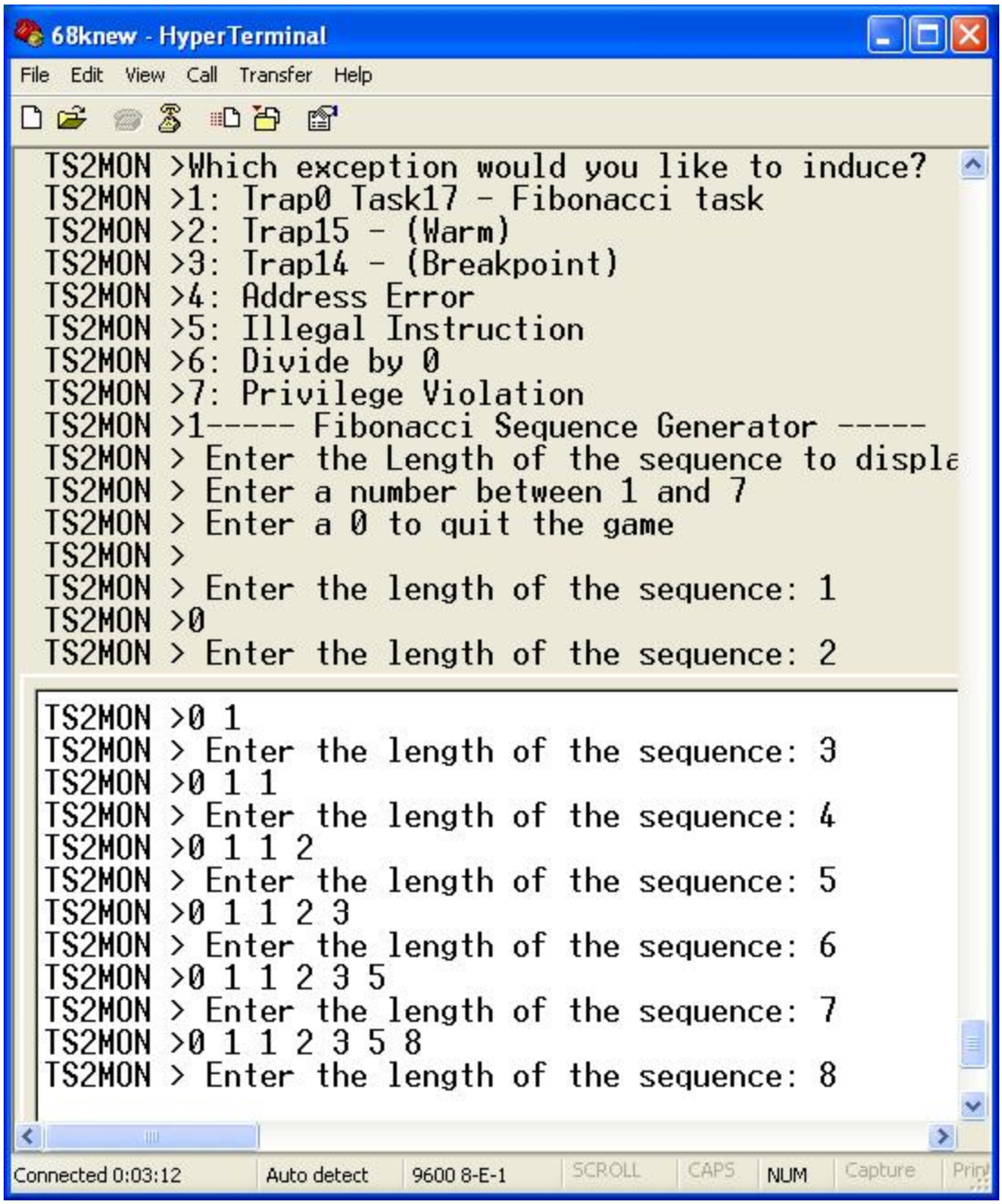

Figure 6: Demonstration of Fibonaci sequence generator in Trap 0 Task 17.

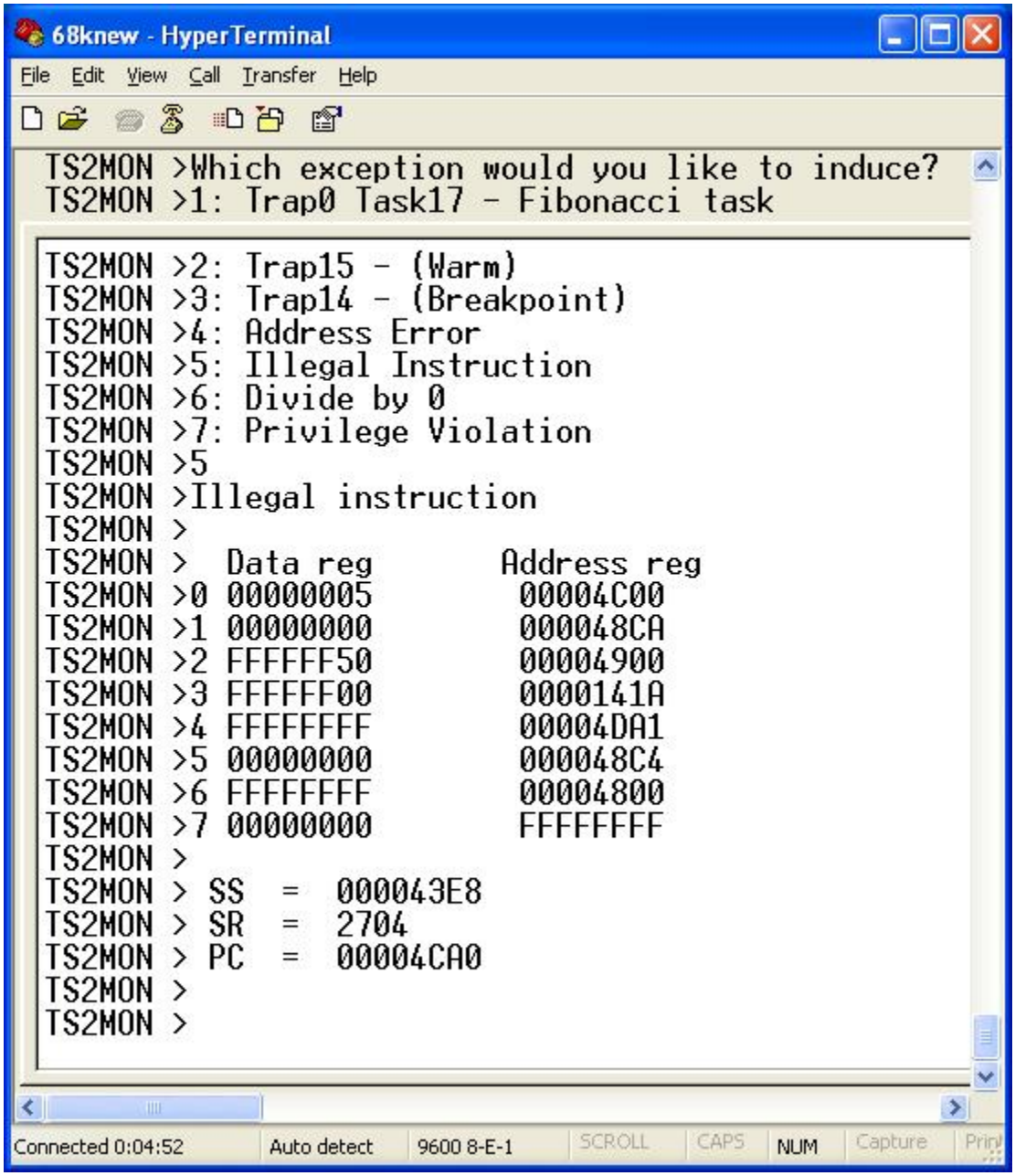

Figure 7: Demonstration of Illegal Instruction exception.

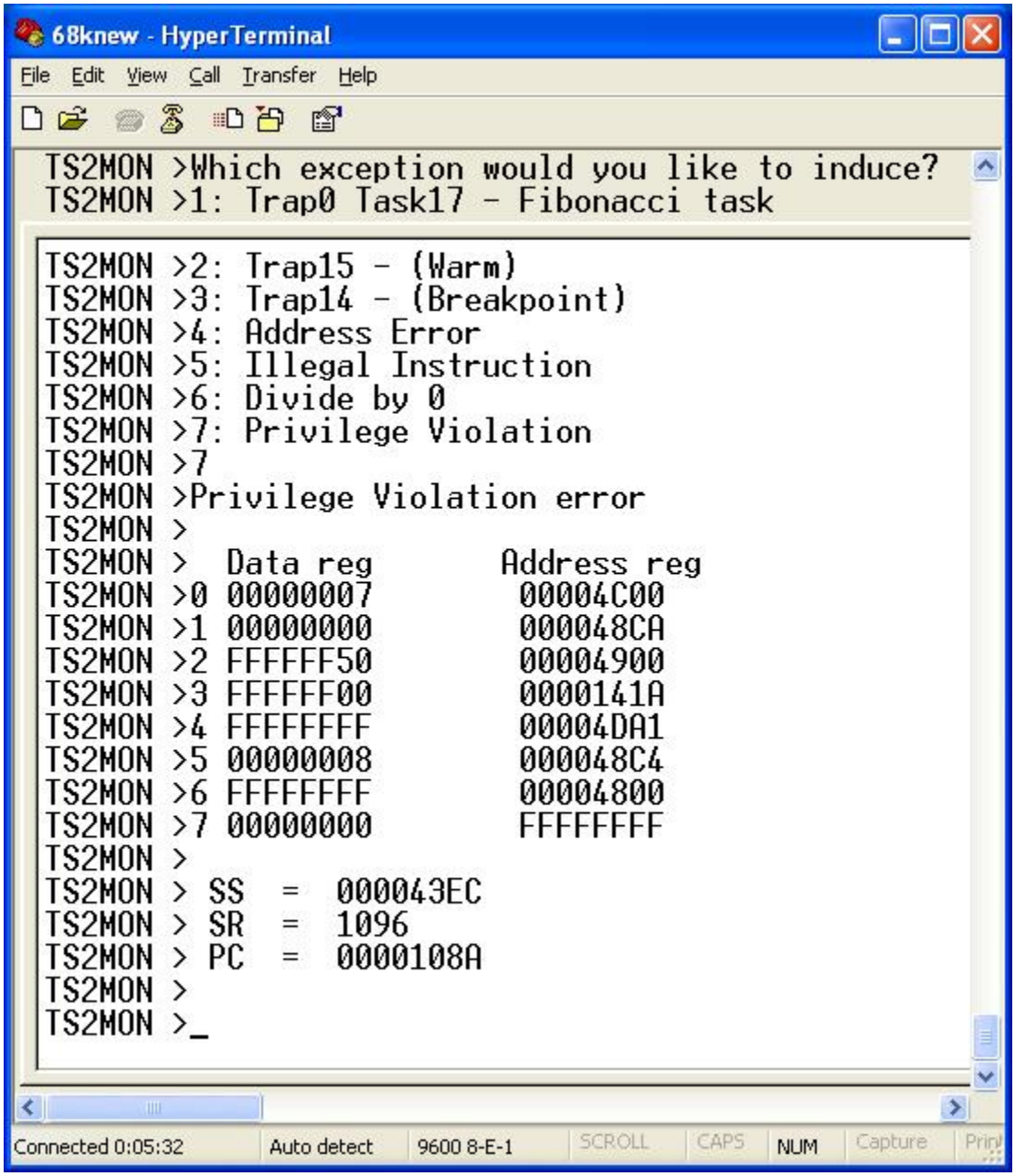

Figure 8: Demonstration of Privilege Violation exception.

Part 4:

In this part the Renesas M16 QSK tutorial program was used as the basis for code to create a new application which sampled the on-board thermistor R20 and then displayed temperature in degrees Fahrenheit on its LCD. The program looped indefinitely, continually displaying the current temperature being sensed by R20. The equation:

R(T) =R0/(eB(1/T0-1/T)) and Microsoft Excel were used to create a linear regression of the temperature profile. Although the thermistors response is not linear with temperature,

for any given sub-region a very accurate linear response can be produced. Because the temperatures around room temperature (and human body temperature) were to be measured, a regression around the temperatures from 20 C to 40 C was chosen. In the C code application that was written for the M16C this temperature in C was converted to F before being displayed on the LCD.

B	3940	Temp K	Temp C	Temp F	R	Vout	A/D
To	298.15	293.1895	20.0395	68.07109	12505.49	1.833691	568
Ro	10000	293.2758	20.1258	68.22643	12456.14	1.830469	567
		293.3621	20.21211	68.3818	12407	1.827246	566
Vref	3.3	293.4484	20.29844	68.53719	12358.08	1.824023	565
Rref	10000	293.5348	20.38478	68.6926	12309.37	1.820801	564
A/D Counts	1024	293.6211	20.47114	68.84805	12260.87	1.817578	563
		293.7075	20.55751	69.00352	12212.58	1.814355	562
Part Number	B57321V2103+060	293.7939	20.6439	69.15902	12164.5	1.811133	561
		293.8803	20.73031	69.31456	12116.63	1.80791	560
		293.9667	20.81674	69.47012	12068.97	1.804688	559
		294.0532	20.90318	69.62573	12021.51	1.801465	558
		294.1396	20.98965	69.78136	11974.25	1.798242	557
		.	.	.	.	.	.
		.	.	.	.	.	.
		.	.	.	.	.	.

Figure 9: Excerpt from Excel spreadsheet used to generate the linear regression.

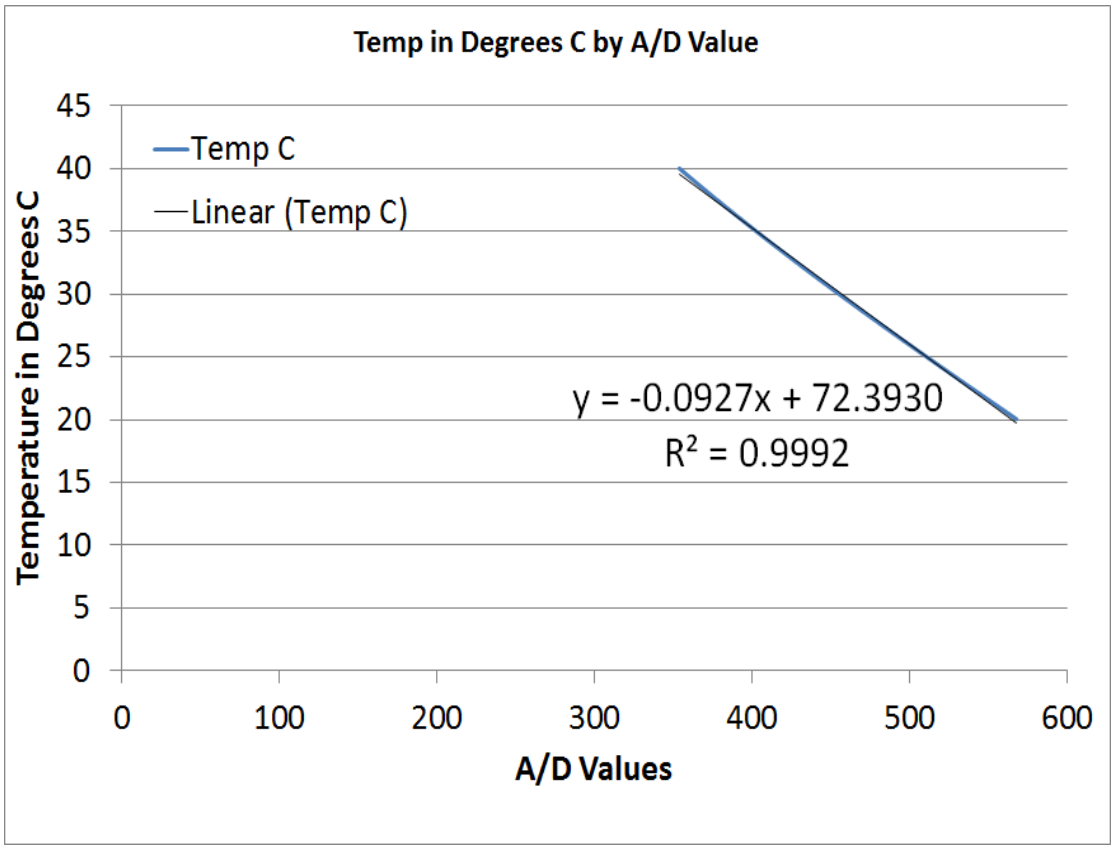

Figure 10: Graph of calculated thermistor temperature with corresponding A/D value and regression.

The demonstrated electronic thermometer was accurate within the specified range and demonstrated excellent refresh rate - responding very quickly to the changes in temperature produced by being touched with a warm human finger or a puff of warm human breath.

Software

Part 2:

Help Function Code:

Code added to Comtab routine:

```
    DC.B    4,4
    DC.B    'HELP'                  HELP displays the user available functions with instructions
    DC.L    HELP-COMTAB
```

Help Subroutine:

```
HElP    LEA.L   HELPPAGE,A1     Load Starting Address of Array of HELP Strings
HELP1   MOVE.L  (A1)+,A4        Post increment to next HELP string label, store in A4
        BSR     PSTRING         Print string pointed to in A4
        BSR     NEWLINE         Print NewLine
        TST.L   (A1)            Test to see if we're at end of HELP label array
        BEQ     ENDHELP         If yes (then end)
```

```
        BRA     HELP1           No continue to print next label and HELP string
ENDHELP RTS
```

Help Table and Index:

```
HELPPAGE DC.L
HJUMP,HMEM1,HMEM2,HLOAD1,HLOAD2,HDUMP1,HDUMP2,HTRAN,HNOBR1,HNOBR2,HDISP,HGO1,HGO2,HBRGT,HPLAN,HKIL
L,HGB,HREG1,HREG2,HHELP,$00000000
HJUMP    DC.B     ' JUMP <address> causes execution to begin at <address>.',0,0
HMEM1    DC.B     ' MEMORY <address> examines contents of <address>',0,0
HMEM2    DC.B     '     and allows them to be changed.',0,0
HLOAD1   DC.B     ' LOAD <string> loads S1/S2 records from the host.',0,0
HLOAD2   DC.B     '     <string> is sent to host.',0,0
HDUMP1   DC.B     ' DUMP <string> sends S1 records to the host and is',0,0
HDUMP2   DC.B     '     preceeded by <string>',0,0
HTRAN    DC.B     ' TRAN enters the transparant mode and is exited by ESC,E.',0,0
HNOBR1   DC.B     ' NOBR <address> removes the breakpoint at <address> from the',0,0
HNOBR2   DC.B     '     BP table. If no address is given all BPs are removed.',0,0
HDISP    DC.B     ' DISP displays the contents of the pseudo registers in TSK_T.',0,0
HGO1     DC.B     ' GO <address> starts program execution at <address> and loads',0,0
HGO2     DC.B     '     regs from TSK_T.',0,0
HBRGT    DC.B     ' BRGT puts a breakpoint in the BP table - but not in the code.',0,0
HPLAN    DC.B     ' PLAN puts the breakpoints in the code.',0,0
HKILL    DC.B     ' KILL removes breakpoints from the code.',0,0
HGB      DC.B     ' GB <address> sets breakpoints and then calls GO.',0,0
HREG1    DC.B     ' REG <reg> <value> loads <value> into <reg> in TASK_T.',0,0
HREG2    DC.B     '     Used to preset registers before a GO or GB.',0,0
HHELP    DC.B     ' HELP displays the user available functions with instructions.',0,0
```

Login code:

```
LOGIN   EQU             *
        MOVE.L  #0,D5           Use D5 to store the number of incorrect attempts
ASKUN   BSR     NEWLINE
        LEA.L   ASKUNAME,A4     Ask for a username
        BSR     PSTRING
        CLR.B   ECHO(A6)
        BSR     GETLINE         Get username
        LEA     VUNAME,A4       Load the address of the valid username to A4
        BSR     VALID8          Validate the username
        CMP     #1,D0           Is it valid?
        BEQ     PWO             If so, ask for the password
        BSR     NEWLINE         Otherwise, ask for them to try again
        LEA.L   INVUN,A4
        BSR     PSTRING
        ADDQ    #1,D5           Increment the number of incorrect attempts
        CMP     #3,D5           Have we reached max attempts?
        BNE     STOK
        BSR     LOCKUP          If the number of incorrect attempts is 3, lock up
STOK    BRA     ASKUN
PWO     MOVE.L  #0,D5           Reset incorrect attempts - 3 tries each @ username and pw
ASKPW   BSR     NEWLINE
        LEA.L   ASKPWORD(PC),A4
        BSR     PSTRING         Ask for a password
        MOVE.B  #2,ECHO(A6)
        BSR     GETLINE
        CLR.B   ECHO(A6)
        LEA     VPWORD,A4       Load the address of the valid password to A4
        BSR     VALID8          Validate the password
        CMP     #1,D0           Is it valid?
        BEQ     ALLGOOD         Then, we've successfully authenticated
        BSR     NEWLINE         Otherwise, ask for them to try again
        LEA.L   INVPW,A4
        BSR     PSTRING
```

```
        ADDQ    #1,D5           Increment the number of incorrect attempts
        CMP     #3,D5           Have we reached max attempts?
        BNE     STOK2
        BSR     LOCKUP          If the number of incorrect attempts is 3, lock up
STOK2   BRA     ASKPW
ALLGOOD RTS

LOCKUP  EQU     *
        BSR     NEWLINE
        LEA.L   LOCKED(PC),A4   ; Tell the user they've been locked out
        BSR     PSTRING
LOCKLP  NOP                     ; Infinite loop
        BRA     LOCKLP
        RTS                     ; Just to be safe

VALID8  EQU             *
        MOVE.L  #1,D0           Set D0 to true to start
VLOOP   CMP.B   #0,(A4)         Check to see if character in string is null
        BNE     NOTNULL         If its not null, continue
        CMP.B   #0,(A1)         Check to see if the user input is newline
        BRA     DONE            If it is, then we are done, return true...
NOTNULL CMP.B   (A1)+,(A4)+     Compare the characters and postincrement
        BNE     SETF            If they're not equal, set false flag
        BRA     VLOOP           Loop
SETF    MOVE.L  #0,D0           Sets our D0 flag to false
DONE    RTS
```

Part 3:

Exception/Trap Test: G1_LAB2_PC_ExceptionTests.x68

```
*-----------------------------------------------------------
* Program:      G1_LAB2_PC_ExceptionTests.x68
* Written by :  Luke Spicer, Adrian Fletcher, Conor Heine
* Date :        9/27/2011
* Description:  Tests the exception handlers added in project 2 part 3
*-----------------------------------------------------------
        ORG   $4C00
START                                   * first instruction of program
TOP     LEA.L   INSTR(PC),A4            * Print instructions
        MOVE.L      #4,D1
        TRAP        #0
        MOVE.L      #2,D1
        TRAP        #0
        LEA.L       E1(PC),A4
        MOVE.L      #4,D1
        TRAP        #0
        MOVE.L       #2,D1
        TRAP        #0
        LEA.L       E2(PC),A4
        MOVE.L      #4,D1
        TRAP        #0
        MOVE.L      #2,D1
        TRAP        #0
        LEA.L       E3(PC),A4
        MOVE.L      #4,D1
        TRAP        #0
        MOVE.L       #2,D1
        TRAP        #0
```

```
        LEA.L       E4(PC),A4
        MOVE.L      #4,D1
        TRAP        #0
        MOVE.L      #2,D1
        TRAP   #0
        LEA.L       E5(PC),A4
        MOVE.L      #4,D1
        TRAP        #0
        MOVE.L      #2,D1
        TRAP        #0
        LEA.L       E6(PC),A4
        MOVE.L      #4,D1
        TRAP        #0
        MOVE.L      #2,D1
        TRAP        #0
        LEA.L       E7(PC),A4
        MOVE.L      #4,D1
        TRAP        #0
        MOVE.L      #2,D1
        TRAP        #0

        CLR.L       D0          * Clear D0 so it can take user input
        MOVE.L      #0,D1       * Get user input
        TRAP        #0
        SUBI        #$30,D0     * Convert ASCII number to integer value

        CMP.B       #1,D0
        BNE         NOT1
        MOVE.L      #17,D1      * FIBGEN task
        TRAP        #0
NOT1    CMP.B       #2,D0
        BNE         NOT2
        TRAP        #15         * Test trap 15
NOT2    CMP.B       #3,D0
        BNE         NOT3
        TRAP        #14         * Test trap 14
NOT3    CMP.B       #4,D0
        BNE         NOT4
        MOVE.W      #1,A4       * Test Address Error Exception
        MOVE.W      #3,(A4)
NOT4    CMP.B       #5,D0
        BNE         NOT5
        ILLEGAL                 * Test illegal instruction exception
NOT5    CMP.B       #6,D0
        BNE         NOT6
        MOVE.L      #10,D3      * Move 5 into D3
        DIVU        #0,D3       * Divide 10 by the contents of D3 (which is 5)
        BRA         $1000       * Returns to command line
NOT6    CMP.B       #7,D0
        BNE         NOT7
        STOP        #2          * Test Privilege Viloation Exception
NOT7    MOVE.L      #2,D1
        TRAP        #0
        LEA.L       INV,A4      * Invalid input
        MOVE.L      #4,D1
        TRAP        #0
```

```
        MOVE.L      #2,D1           * Task 2
        TRAP        #0              * of Trap 0 Prints a newline
        MOVE.L      #2,D1           * Task 2
        TRAP        #0              * of Trap 0 Prints a newline
        BRA         TOP
```

* Variables and Strings

```
INSTR   DC.B    'Which exception would you like to induce?',0
E1      DC.B    '1: Trap0 Task17 - Fibonacci task',0
E2      DC.B    '2: Trap15 - (Warm)',0
E3      DC.B    '3: Trap14 - (Breakpoint)',0
E4      DC.B    '4: Address Error',0
E5      DC.B    '5: Illegal Instruction',0
E6      DC.B    '6: Divide by 0',0
E7      DC.B    '7: Privilege Violation',0
INV     DC.B    'Invalid input... try that again',0

END    START                    * last line of source
```

Exception handlers added in Lab 2:

```
PRV_ER  EQU      *              Privilege Violation Error (group 1) exception
        MOVE.L  A4,-(A7)Save A4
        LEA.L   MES13(PC),A4    Point to error message string
        BSR     HEADING         Pring the error message as a heading
        MOVE.L  (A7)+,A4Restore A4
        BRA     GROUP1          Deal with group 2 exception
*
DIV0_ER EQU     *               Divide by zero error (group 2) exception
        BSR.S   GROUP2  Treat as group 2 exception
        MOVE.L  A4,-(A7)Save A4
        LEA.L   MES14(PC),A4    Point to error message string
        BSR     HEADING         Pring the error message as a heading
        MOVE.L  (A7)+,A4Restore A4
        BSR     EX_DIS          Display saved registers
        BRA     WARM            Return to monitor
```

Firmware routines:

```
TRAP_0  EQU     *                User links to  TS2BUG via TRAP #0
        CMP.B   #0,D1            D1 = 0 = Get character
        BNE.S   TASK1
        BSR     GETCHAR
        RTE
TASK1   CMP.B   #1,D1            D1 = 1 = Print character
        BNE.S   TASK2
        BSR     PUTCHAR
        RTE
TASK2   CMP.B   #2,D1            D1 = 2 = Newline
        BNE.S   TASK3
        BSR     NEWLINE
        RTE
TASK3   CMP.B   #3,D1            D1 = 3 = Get parameter from buffer
        BNE.S   TASK4
        BSR     PARAM
        RTE
TASK4   CMP.B   #4,D1            D1 = 4 = Print string pointed at by A4
        BNE.S   TASK5
        BSR     PSTRING
```

```
        RTE
TASK5   CMP.B   #5,D1           D1 = 5 = Get a hex character
        BNE.S   TASK6
        BSR     HEX
        RTE
TASK6   CMP.B   #6,D1           D1 = 6 = Get a hex byte
        BNE.S   TASK7
        BSR     BYTE
        RTE
TASK7   CMP.B   #7,D1           D1 = 7 = Get a word
        BNE.S   TASK8
        BSR     WORD
        RTE
TASK8   CMP.B   #8,D1           D1 = 8 = Get a longword
        BNE.S   TASK9
        BSR     LONGWD
        RTE
TASK9   CMP.B   #9,D1           D1 = 9 = Output hex byte
        BNE.S   TASK10
        BSR     OUT2X
        RTE
TASK10  CMP.B   #10,D1          D1 = 10 = Output hex word
        BNE.S   TASK11
        BSR     OUT4X
        RTE
TASK11  CMP.B   #11,D1          D1 = 11 = Output hex longword
        BNE.S   TASK12
        BSR     OUT8X
        RTE
TASK12  CMP.B   #12,D1          D1 = 12 = Print a space
        BNE.S   TASK13
        BSR     PSPACE
        RTE
TASK13  CMP.B   #13,D1          D1 = 13 = Get a line of text into
        BNE.S   TASK14          the line buffer
        BSR     GETLINE
        RTE
TASK14  CMP.B   #14,D1          D1 = 14 = Tidy up the line in the
        BNE.S   TASK15          line buffer by removing leading
        BSR     TIDY            leading and multiple embeded spaces
        RTE
TASK15  CMP.B   #15,D1          D1 = 15 = Execute the command in
        BNE.S   TASK16          the line buffer
        BSR     EXECUTE
        RTE
TASK16  CMP.B   #16,D1          D1 = 16 = Call RESTORE to transfer
        BNE.S   TASK17          the registers in TSK_T to the 68000
        BSR     RESTORE         and therefore execute a program
        RTE
TASK17  CMP.B   #17,D1           D1 = 17 = Call Fibinacci Sequence Generator
        BNE.S   TASK18
        BSR     FIBGEN
TASK18  RTE
```

Fibgen Routine:

```
FIBGEN  LEA.L   FMES1(PC),A4    Point to first of FibGen Instruction strings, store in A4
        BSR     PSTRING                 Print string pointed to in A4
        BSR     NEWLINE                 Print NewLine
        LEA.L   FMES2(PC),A4    Point to second of FibGen Instruction strings, store in A4
        BSR     PSTRING                 Print string pointed to in A4
        BSR     NEWLINE                 Print NewLine
        LEA.L   FMES3(PC),A4    Point to third of FibGen Instruction strings, store in A4
        BSR     PSTRING                 Print string pointed to in A4
        BSR     NEWLINE                 Print NewLine
        LEA.L   FMES4(PC),A4    Point to fourth of FibGen Instruction strings, store in A4
        BSR     PSTRING                 Print string pointed to in A4
```

```
        BSR     NEWLINE                 Print NewLine

FIBLOOP BSR     NEWLINE                 Print NewLine
        LEA.L   FMES5(PC),A4     Point to user input prompt string, store in A4
        BSR     PSTRING                 Print string pointed to in A4

        BSR     GETCHAR          Get a character from input device
        SUB.B   #$30,D0          Convert to binary

        CMP     #0,D0                   if user chose "0" exit and end
        BEQ     FIBEND
        BSR     NEWLINE                 Print newline


        CMP     #7,D0                   Compare the user input (in DO)
        BGT     BADINPUT         to 7, if greather than, they gave bad input
        CMP     #0,D0                   Compare the user input (in D0)
        BLT     BADINPUT         to 0, if less than, they gave bad input

        MOVE.B  #0,FN2(A6)              intialize fibonacci counters
        MOVE.B  #1,FN1(A6)              Fn-2 = 0, Fn-1 = 1
        MOVE.B  D0,D3                   COUNT = D0 (the users choice of lines)
        MOVE.B  #'0',D0                 print the first character of the sequence
        BSR     PUTCHAR


        CMP.B   #2,D3                   if COUNT < 2 then COUNT = 1
        BLT     FIBLOOP         we're done so ask for input again

        MOVE.B  #' ',D0         print the second character of the sequence
        BSR     PUTCHAR         and a space
        MOVE.B  #'1',D0
        BSR     PUTCHAR

        CMP.B   #2,D3                   If count == 2, we're done so ask for input a again
        BEQ     FIBLOOP


        SUBI.B  #2,D3                   subtract 2 from the count the user gave us
*to make up for the first 2 numbers in the sequence already being printed
FIBFOR  CMP.B   #0,D3           If D3 == 0, then we've printed the entire seqence
        BEQ     FIBLOOP                 So ask for input from the user again
        SUBI.B  #1,D3           If D3 != 0, then decrement D3 and print next value in sequence

        MOVE.B  #' ',D0                 display a blank space between number
        BSR     PUTCHAR                 using PUTCHAR

        CLR     D0                              Clear D0
        ADD.B   FN2(A6),D0              add FN2 to FN1
        ADD.B   FN1(A6),D0              store this sum in D0 to display
        ADD.B  #$30,D0                  Max number an ASCII character
        BSR     PUTCHAR                 Display character
        SUB.B   #$30,D0                 convert ASCII back to value

        MOVE.B  FN1(A6),FN2(A6)  FN2= previous FN1
        MOVE.B  D0,FN1(A6)              FN1= prevous fibonacci number


        BRA     FIBFOR                  Branch always to the start of the For loop

BADINPUT BSR    NEWLINE                 Print newline
        LEA.L   FMES6(PC),A4    Move pointer to bad input error message
        BSR     PSTRING                 Print string pointed to in A4
        BSR     NEWLINE                 Print NewLine


        BRA     FIBLOOP                 User entered bad data so ask for input again
FIBEND   BSR    NEWLINE                 Print newline
        RTS
```

Part 4:

TempRead.c

```
#include "QSKDefines.h"
#include "proto.h"
#include "extern.h"

/*****************************************************************************/
/*                                                                           */
/*  DATE       :Thursday, Sep 9, 2011                                        */
/*                                                                                 */
/*  DESCRIPTION :      Contains the main code to read the the                 */
/*                     the thermistor A/D and display the                     */
/*                     temperature  in degrees F on the LCD                   */
/*  CPU GROUP   :62P                                                         */
/*                                                                           */
/*  Copyright (c) 2009 by BNS Solutions, Inc.                                      */
/*  All rights reserved.                                                           */
/*                                                                           */
/*****************************************************************************/

int disp_count;                         // LED control variable
uint A2DValue;
uint A2DValuePot;
uint A2DValueTherm;
uchar A2DProcessed;

void main(void)
//-----------------------------------------------------------------------------
//  Purpose:    The MCU will come here after reset.
//
//
//  Rev:       1.0     Initial Release
//
//  Notes:             None
//-----------------------------------------------------------------------------
{
    float temp;

    MCUInit();
    InitDisplay("G1-L2-P4");
    InitUART();
    //BNSPrintf(SERIAL, "\n\rLab2P4\n\r");
      TimerInit();
    ADInit();

while(1) {
    if (A2DProcessed == TRUE) {
                                                    // only update the display when a
                                                    // new value is available
                A2DProcessed = FALSE;               // Each time a new value is
                                                    // available note for next loop
        temp = -0.094758*(A2DValueTherm+31)+73.989706;   // Convert A/D value to
                                                           //temp in degrees C
        temp = (temp*9.0)/5.0 + 32.0;        // Convert degrees C to degrees F
        BNSPrintf(LCD,"\t%0.2f%cF ",temp, 223);
                                             // Display 'temp"degreesymbol"F' on LCD
        }
    }
}


void TimerInit(void)
//-----------------------------------------------------------------------------------------
//  Purpose:    This will set up the A0 timer for 1ms and the A1 as counter
//
//
//  Rev:       1.0     Initial Release
```

```
//
//  Notes:             None
//-------------------------------------------------------------------------------
{
   /* Configure Timer A0 - 1ms (millisecond) counter */
   ta0mr = 0x80;    // Timer mode, f32, no pulse output
   ta0 = (unsigned int) (((f1_CLK_SPEED/32)*1e-3) - 1); // (1ms x 12MHz/32)-1 = 374

   /* Configure Timer A1 - Timer A0 used as clock */
   ta1mr = 0x01;       // Event Counter mode, no pulse output
   ta1 = 0x3FF;        // initial value - max value of ADC (0x3FF)
   trgsr = 0x02;       // Timer A0 as event trigger

/* The recommended procedure for writing an Interrupt Priority Level is shown
   below (see M16C datasheets under 'Interrupts' for details). */

DISABLE_IRQ            // disable irqs before setting irq registers - macro defined in
                       // skp_bsp.h
   ta1ic = 3;          // Set the timer A1's IPL (interrupt priority level) to 3
   ENABLE_IRQ          // enable interrupts macro defined in skp_bsp.h

   /* Start timers */
   ta1s = 1;           // Start Timer A1
   ta0s = 1;           // Start timer A0
}

void ADInit(void)
//-------------------------------------------------------------------------------------------
--------------
//  Purpose:    Set up the A2D for one shot mode.
//
//
//  Rev:       1.0     Initial Release
//
//  Notes:             None   ------------------------------------------------------------
---------------------
{

   /* Configure ADC - AN0 (Analog Adjust Pot) */
   adcon0 = 0x80;    // AN0, One-shot, software trigger, fAD/2
   adcon1 = 0x28;    // 10-bit mode, Vref connected.
   adcon2 = 0x01;    // Sample and hold enabled
}
```

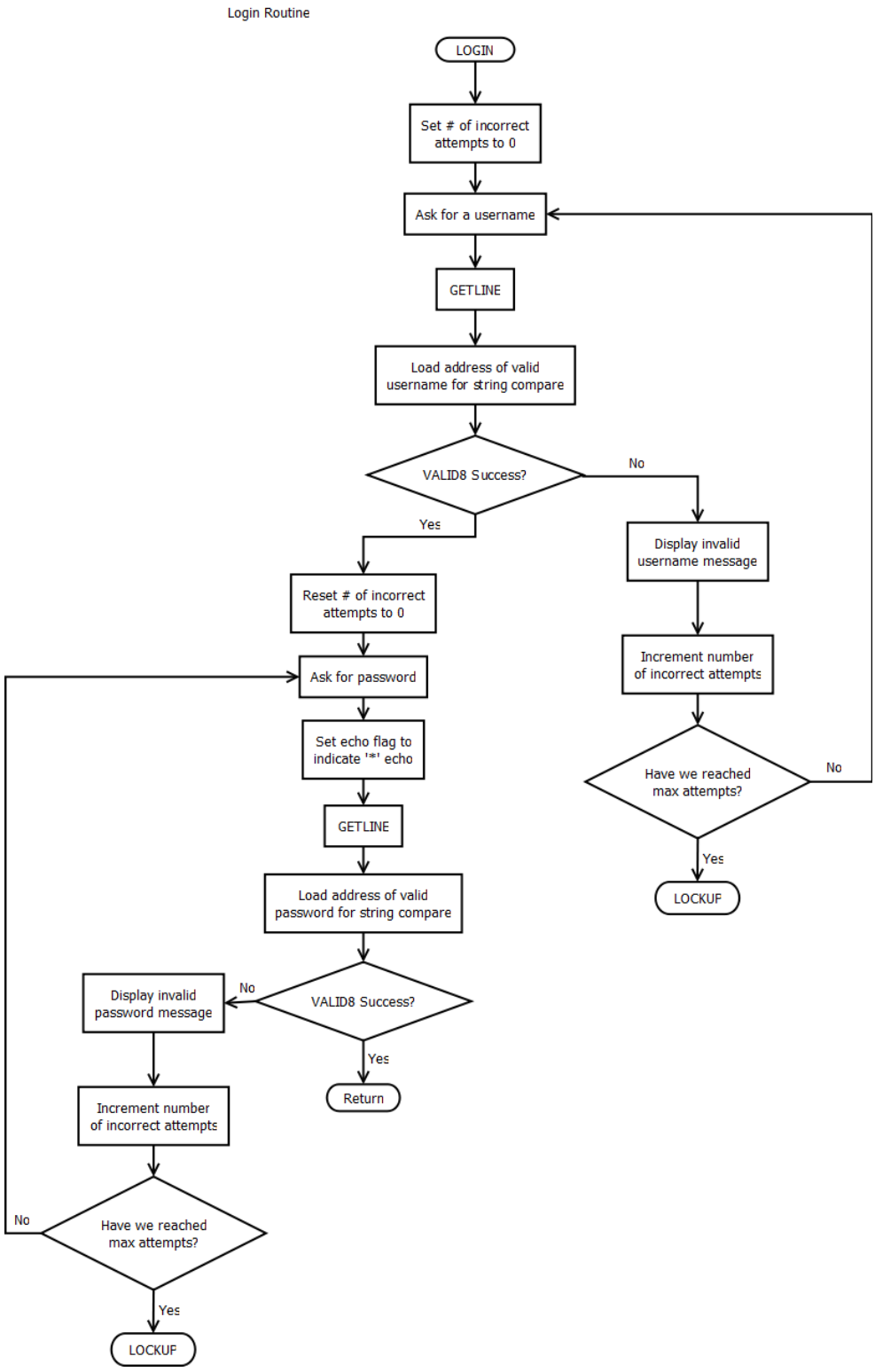

Figure 9: LOGIN Routine Software Flowchart

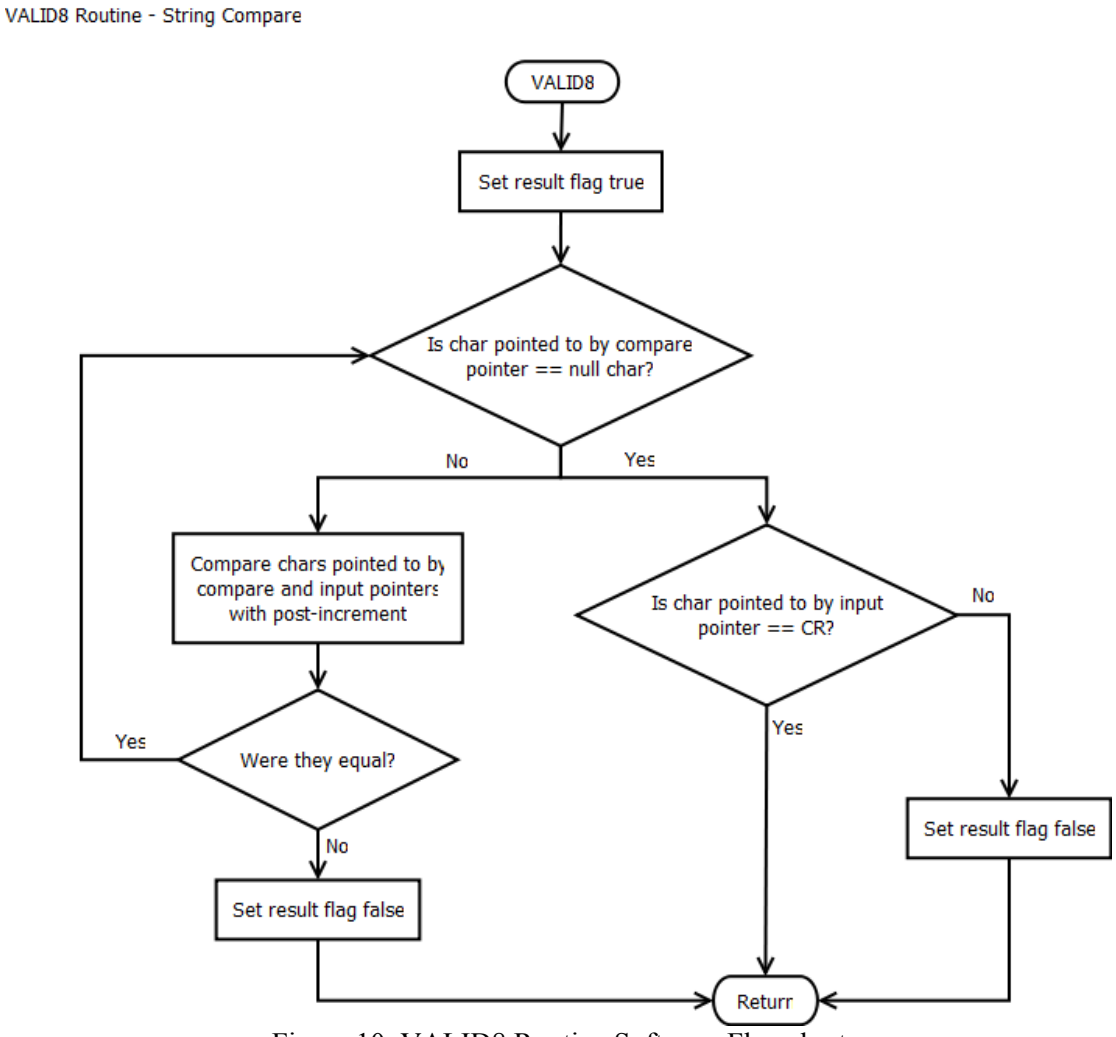

Figure 10: VALID8 Routine Software Flowchart

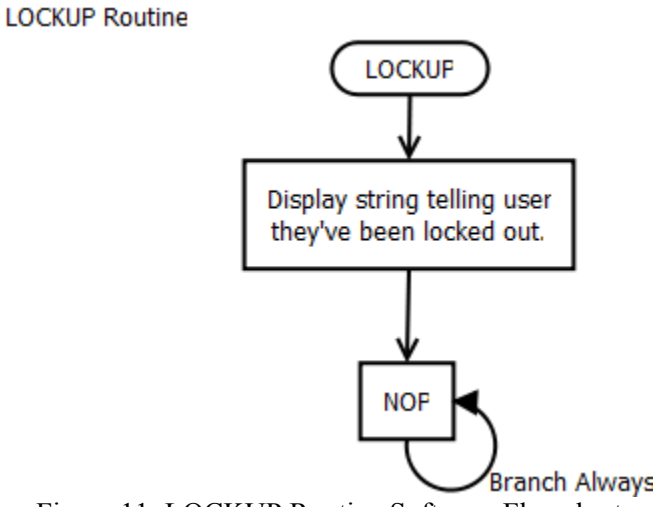

Figure 11: LOCKUP Routine Software Flowchart

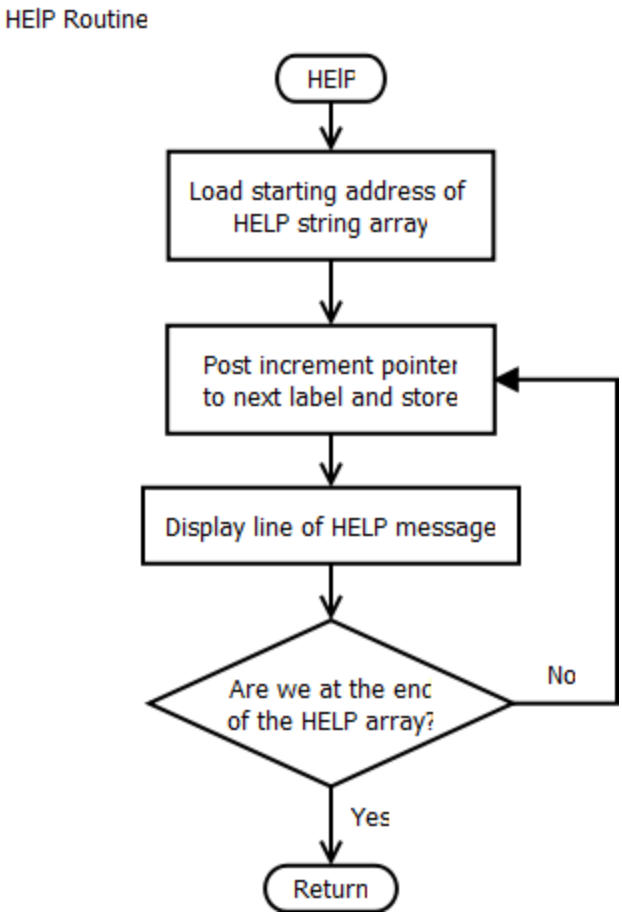

Figure 12: Help Function Software Flowchart

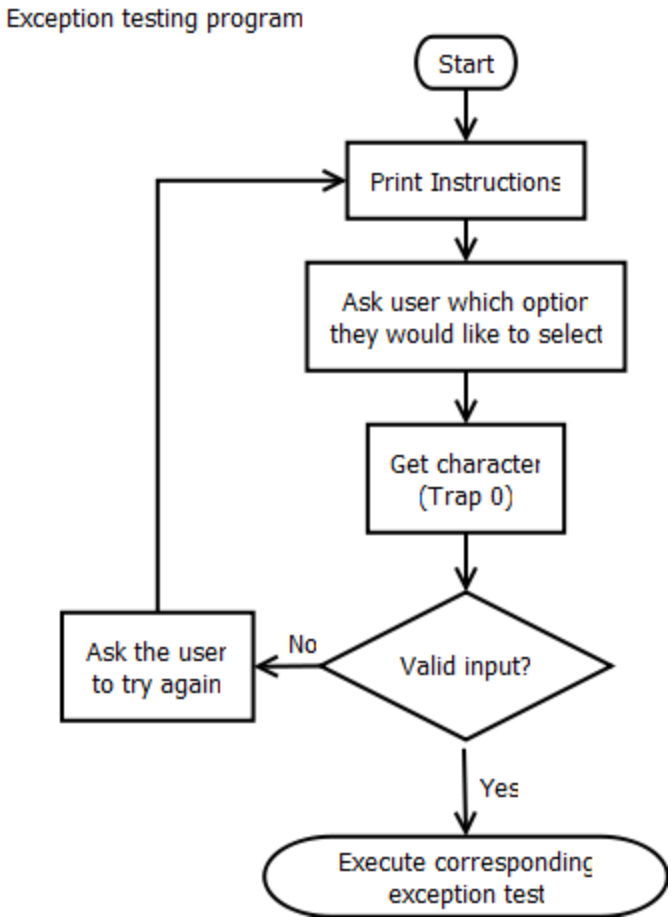

Figure 13: Exception Test Program Software Flowchart

Analysis

For this lab, the team focused on introducing the M68k's interrupt handling system. After background research on the system and the performance of simple guided tests, the team learned how to manipulate the exception table to reside in firmware and react appropriately to respective errors. This will prove beneficial for future projects as well as embedded systems design as a whole because the students will understand how to create their own interrupts and handle them appropriately.

The most difficult problem encountered with the vector table was its physical location in EEPROM. The vector table was initially stored in RAM through the X_SET routine that mapped the corresponding memory locations - and, thus, could be overwritten - so, the exception table had to be moved to the first $3FFF memory locations at the head of the firmware source. This microcomputer setup maps the first $3FFF memory locations to ROM by default, and the workaround decided upon was to DC.B all exceptions to their proper locations at the header of the firmware source.

Students also gained experience in modifying and adding functionality to the firmware. After examining and testing the source, the students implemented both "log in" and "help" functions in the firmware. Although the log in routine does not implement any kind of user name or password encryption in code, it does provide a simple form of security for the simple microcomputer. The help function also familiarized the students with utilizing data structures in subroutines by iterating through an array of null-terminated strings and displaying their contents to the screen. This is a dynamic function because any help listing can be added to the help array and have its associated functionality displayed to the screen whenever "HELP" is called.

In Part 4 further skills and experiences were gained with the M16C QSK. The students used the QSK thermistor and LCD along with the M16's A/Ds to create a basic electronic thermometer and display. This exercise taught the team about the use of datasheets and schematics when interfacing embedded systems with external devices. Also supplementary programs, like Microsoft Excel, were used to generate the linear regression which approximated the thermistors non-linear response to temperature. By using a linear regression over a subset of the thermistors full range instead of a 3rd or 4th order polynomial over the whole range, the system is able to processes the data much faster and still read accurate temperatures over the target range.

Conclusion

The team successfully implemented basic functions in the firmware - the ‘help’ and ‘login’ functions - thereby demonstrating an ability to leverage the existing routines in the firmware. In the case of the login function, the existing firmware routines were actually augmented as a method for echoing asterisks was added.

The team then developed the means to use the aforementioned firmware functionality in applications stored external to the firmware. The team successfully set up a an exception table to ensure that errors in external applications are handles gracefully. In addition, Trap calls were expanded by adding routine from project 1. The team then demonstrated the use of these firmware tools in an application loaded external to the firmware.

Supplementally, the team designed a straightfoward application for the M16C development kit. This application successfully incorporated the hardware built into the QSK.

The team learned some valuable lessons over the course of the design mentioned above. The team experienced the perils of ROM memory when the vector table was set up. Most significantly, though, the team fine-tuned a technique that will be useful throughout the rest of the 525 class and programming for microprocessors in general: the expert use of firmware functionality to simplify the development of applications. With well-written exception handlers and trap calls, the team will be better equipped to quickly generate powerful applications.

References

CECS 525 Project 2 Report Template by Eugene Rocky

The 68000 Microprocessor Fifth Edition by James L. Antonakos