```
; University of Illinois at Chicago, Dept. of Electrical and Computer Engineering
; ECE 367 -Microprocessor-Based Design
; Semester: Spring 2013

; Experiment Title: Count (Not So) Simple Three Function Calculator
; Experiment Description: This is a calculator that will allow a user to do
addition,
;                              subtraction, or multiplication using a three digit
number
;                              of any sort.  You can use C for backspace as well.
; Date: 3/27/2013
; Updated: 3/27/2013
; Version: 1
; Programmer: Mitchell Hedditch
; Lab Session: Tuesday 8AM-10:50AM
; Programming Notes:
;     1. STORE ALL VALUES IN VARIABLES (EVEN WHEN PASSING TO OTHER ROUTINES)
;     2. COMMENT ALL BLOCKS OF CODE
;     3. ORDER SUBROUTINES IN ORDER OF FIRST USED
;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
; REMAINING TASKS
;     1. UPDATE INPUT FROM 1 BYTE TO 2 BYTES
;     2. IMPLEMENT CL FUNCTION
;     3. ALLOW SIGNED BITS
;     4. IMPLEMENT ADDITION
;     5. IMPLEMENT SUBTRACTION
;     6. IMPLEMENT MULTIPLICATION
;     7. IMPLEMENT DIVISION TO 2 DECIMAL ACCURACY
;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


; Define symbolic constants
REGBAS          EQU $0000                     ; REGISTER BLOCK STARTS AT $0000
PortA           EQU $0000                     ; PortA address (relative to Regbase
i.e. offset)
DDRA            EQU $0002                     ; PortA Data Direction control register
offset
PortM           EQU $0250                     ; PortM offset (actual address of
PortM)
DDRM            EQU $0252                     ; PortM Data Direction control register
offset
PortT           EQU $0240                     ; PortT offset (actual address of
PortT)
DDRT            EQU $0242                     ; Actual Data Direction Register for
PortT
PortE           EQU $0008                     ; PortE LABEL (XIRQ' INTERRUPT)
; TIMER SYMBOLIC CONSTANTS
TSCR1   EQU $0046                     ; TIMER SYSTEM CONTROL REGISTER - WITH FAST
FLAGS
TSCR2   EQU $004D                     ; TIMER SYSTEM CONTROL REGISTER 2 - NO FAST
FLAGS
TFLG1   EQU $004E                     ; TIMER INTERRUPT FLAG1 REGISTER
TFLG2           EQU $004F                     ; TIMER INTERRUPT FLAG2 REGISTER
TIOS    EQU $0040                     ; TIMER INTERRUPT OUTPUT COMPARE
TCNT    EQU $0044                     ; TIMER COUNTER REGISTER - 16 BIT, INPUT
CAPTURE/OUTPUT COMPARE REQUIRED
TC0     EQU $0050                     ; TIME I/O COMPARE SELECT 0 REGISTER TO
LOCATION $50 HEX
TC1     EQU $0052                     ; TIME I/O COMPARE SELECT 1 REGISTER TO
LOCATION $52 HEX
TIE             EQU $004C                     ; TIMER TCi INTERRUPT ENABLE REGISTER
; INTERRUPT CONSTANTS
IRQCR           EQU $001E                     ; IRQ CONTROL REGISTER ADDRESS LABEL
```

```
; SERIAL COMMUNICATION INTERFACE
SPCR1           EQU $00D8
SPCR2           EQU $00D9
SPIB            EQU $00DA
SPSR            EQU $00DB
SPDR            EQU $00DD
ENABLE  EQU $02                         ; LCD ENABLE at PM1
RCK     EQU $08         ; RCK connect to PM3
;UNKNOWN
INITRG          EQU $0011
INITRM          EQU $0010
PLLCTL  EQU $003A
; CLOCKS
CLKSEL  EQU $0039
CRGFLG  EQU $0037
SYNR    EQU $0034
REFDV   EQU $0035
COPCTL  EQU $003C                       ; COMPUTER OPERATING PROPERLY CONTROL LOCATION
; VARIABLES
TEST            EQU $3800                       ; DEFINE LOCATION FOR TEST BYTE STORAGE
FOR DEBUGGING
SAVE_X          EQU $3802                       ; Defines location for the storage of
the X index register
SAVE_Y          EQU $3804                       ; Defines location for the storage of
the Y index register
; TIMER VARIABLES
TIME_COUNT      EQU $3814                       ; MEM ADDRESS TO STORE TIME FOR SECONDS
TMR_FLAG        EQU $3808                       ; DEFINES LOCATION FOR STORAGE OF TIMER
FLAG
                                                ; FLAG= 0->NOTHING; 1->TIMER FIRED
; GENERAL FLAGS
INVALID_KEY     EQU $3809                       ; DEFINES LOCATION FOR STORAGE OF
INVALID KEY FLAG
XIRQ_FLAG       EQU $3815                       ; PAUSE FOR XIRQ (1 MSEC)
;KEYPAD VARIABLES
NUM_FLAG        EQU $3816                       ; A FLAG THAT GOES TO 1 IF A KEY IS
PRESSED ON THE PAD
CUR_PAD_VAL     EQU $3817                       ; USED TO HOUSE THE VALUE FOR THE
CURRENT KEYPAD ITERATION
CUR_COLUMN      EQU $3819                       ; STORAGE LOCATION FOR VARIABLE OF
CURRENT COLUMN
; CALCULATOR VARIABLES
OPERAND1        EQU $3820                       ; 2 BYTE STORAGE FOR OPERAND 1
(3820-3821)
OPERAND2        EQU $3822                       ; 2 BYTE STORAGE FOR OPERAND 2
(3822-3823)
SOLUTION        EQU $3824                       ; 2 BYTE STORAGE FOR SOLUTION OF
CALCULATION (3824-3825)
SOLUTION2       EQU $3826                       ; 2 BYTE STORAGE FOR SOLUTION2 OF
CALCULATION (3826-2827)
INPUT           EQU $3828                       ; 2 BYTE STORAGE FOR USER INPUT #
(3828-3829)
OP_FLAG         EQU $3830                       ; A FLAG TO INDICATE WHICH OPERAND THE
USER IS ENTERING
OP              EQU $3831                       ; THE OPERATOR +,-,*,/
SOLVED_FLAG     EQU $3832                       ; A FLAG TO INDICATE THE SOLUTION WAS
CALCULATED
DIVISOR         EQU $3833                       ; USED AS A CONTAINER FOR SOLUTION
DISPLAY
CLR_DIGIT       EQU $3835                       ; FLAG TO INDICATE CLEAR WAS ALREADY
PRESSED ONCE
RS      EQU $01         ; REGISTER SELECT (RS) AT PM0 (0=COMMAND, 1=DATA)
```

```
;*******************************************************************************
*******
; The ORG statment below is followed by variable definitions
; THIS IS THE BEGINNING SETUP CODE
;
        ORG     $3800   ; Beginning of RAM for Variables
;
; The main code begins here. Note the START Label
;
        ORG     $4000   ; Beginning of Flash EEPROM
START           LDS     #$3FC0  ; Top of the Stack
        SEI             ; Turn Off Interrupts
                MOVB    #$00, INITRG    ; I/O and Control Registers Start at $0000
        MOVB    #$39, INITRM    ; RAM ends at $3FFF
;
; We Need To Set Up The PLL So that the E-Clock = 24MHz
;
        BCLR CLKSEL,$80                 ; disengage PLL from system
        BSET PLLCTL,$40                 ; turn on PLL
        MOVB #$2,SYNR                   ; set PLL multiplier
        MOVB #$0,REFDV                  ; set  PLL divider
        NOP             ; No OP
        NOP             ; NO OP
PLP     BRCLR CRGFLG,$08,PLP            ; while (!(crg.crgflg.bit.lock==1))
        BSET CLKSEL,$80                 ; engage PLL

        CLI             ; TURN ON ALL INTERRUPTS
;
; End of  setup code. You will always need the above setup code for every experiment


;*******************************************************************************
******
; Begin Code
;*******************************************************************************
******
; Initialize the 68HC11

                LDY  #REGBAS                    ; Initialize register base address
                                                ; Note that Regbas = $0000 so now <Y> =
$0000
                SEI                             ; TURN OFF INTERRUPTS

; INITIALIZE ALL SYSTEM PORTS/INTERRUPTS/DDRS/FLAGS/ETC
                ; SETUP S BIT ON INTERRUPTS
                MOVB #$C0, IRQCR                ; TURN ON IRQ' INTERRUPT AND SET TO
EDGE TRIGGERED
                ANDCC #$BF                      ; SET THE X-BIT TO USE XIRQ' AS A
STANDARD INTERRUPT

                JSR INIT                        ; INITIALIZE ALL OF OUR VARIABLES,
FLAGS, ETC.
                JSR InitLCD                     ; INITIALIZE THE LCD

                ; ALL VARIABLES ARE INITIALIZED SO WE'RE READY FOR INTERRUPTS
                CLI                             ; TURN ON INTERRUPTS
;*********************************
;                               *
;      MAIN PROGRAM CODE IS HERE   *
;                               *
;*********************************
;
                JSR DIRECTIONS                  ; SHOW THE USER THE DIRECTIONS
                JSR DRAW_SCREEN                 ; DRAW SCREEN FOR THE FIRST TIME
```

```
POLL:           MOVB #$00,INVALID_KEY          ; RESET INVALID KEY FLAG
                MOVB #$00,NUM_FLAG             ; CLEAR THE NUM FLAG TO WAIT FOR A NEW
KEY
                JSR GET_KEY                    ; CHECK THE KEYPAD FOR A PRESSED VALUE
                BRCLR NUM_FLAG,$01,NO_KEY      ; IF NO KEY HAS BEEN PRESSED THEN MOVE
ON THE THE NO_KEY LINE
                JSR CALCULATE                  ; CHECK TO SEE IF USER WANTS TO
CALCULATE SOLUTION
                BRSET SOLVED_FLAG,$01,CONTINUE; IF THE SOLVED FLAG IS SET GO TO
CONTINUE
                JSR OPERATION                  ; GO TO OPERATION SUB TO DETERMINE
FUNCTION KEY PRESSED
CONTINUE        BCLR SOLVED_FLAG,$01           ; MAKE SURE THE SOLVED FLAG GETS
CLEARED
                JSR CHECK_KEY                  ; CHECK TO SEE IF THE KEY IS VALID
                BRSET INVALID_KEY,$01,NO_KEY   ; GO BACK AND POLL AGAIN IF WE'VE GOT A
BAD KEY
                JSR INSERT_NUMBER              ; IF A NUMBER KEY HAS BEEN PRESSED THEN
LOAD THE NEW NUMBER
NO_KEY          BRA POLL                       ; GO BACK START PROCESSING AT POLL
AGAIN!



;*********************************
;                                *
;      SUBROUTINES BELOW         *
;                                *
;*********************************
;*******************************************************************************
***********
; PROGRAM INITIALIZATION
INIT:           ; SETUP THE DATA DIRECTON REGISTERS AND INITIALIZE PORT A & PORT T
                MOVB #$F0,DDRT                 ; SET PortT PINS 4-7 TO OUTBOUND AND
PINS 0-3 TO INBOUND
                MOVB #$00,PortT                ; SET ALL PortT PINS TO LOW

                ; SET UP SERIAL PROGRAM INTERFACE SYSTEM
                MOVB #$22,SPIB                 ; SPI CLOCKS A 1/24 OF E-CLOCK
                MOVB #$3B,DDRM                 ; SETUP PortM DATA DIRECTION
                MOVB #$50,SPCR1                ; ENABLE SPI AND SET MODE AS MASTER
                MOVB #$00,SPCR2                ; RESETS SPCR2 TO $00 (ALSO DOES AT
RESET)
                BSET PortM,RCK                 ; SET RCK TO IDLE HIGH
                BCLR PortM, ENABLE             ; ENABLE to Idle LOW

                ; SET UP TIMER COUNT INFORMATION AND PRESCALE INITIALIZE THE COUNTER
                MOVB #$06,TSCR2                ; CONFIGURE PRESCALE FACTOR 64
                MOVB #$01,TIOS                 ; ENABLE OC0 FOR OUTPUT COMPARE
                MOVB #$90,TSCR1                ; ENABLE TCNT & FAST FLAGS CLEAR
                MOVB #$01,TIE                  ; ENABLE TC1 INTERRUPT
                LDD TCNT                       ; FIRST GET CURRENT TCNT
                ADDD #3750                     ; INCREMENT TCNT COUNT BY 3750 AND
STORE INTO TC0
                STD TC0                        ; WE WILL HAVE A SUCCESSFUL COMPARE IN
375 CLICKS
                MOVB #$01,TFLG1                ; OF TCNT. BETTER BE SURE FLAG C0F IS
CLEAR TO START

                ; INITIALIZE PROGRAM DEFINED VARIABLES
                MOVW #$0000,OPERAND1           ; LOAD OPERAND1 WITH 0
                MOVW #$0000,OPERAND2           ; LOAD OPERAND2 WITH 0
```

```
                MOVW #$0000,SOLUTION          ; LOAD SOLUTION1 WITH 0
                MOVW #$0000,SOLUTION2         ; LOAD SOLUTION2 WITH 0
                MOVW #$0000,INPUT             ; LOAD INPUT WITH 0
                MOVB #$00,OP_FLAG             ; SET OUR OP FLAG TO 0
                MOVB #$00,OP                  ; SET THE OP TO 0
                MOVB #$00,SOLVED_FLAG         ; SET THE SOLVED FLAG TO 0

                MOVB #$00,TMR_FLAG            ; INITIALIZE THE TIMER FLAG TO LOW
                MOVB #$00,TIME_COUNT          ; SET TIME_COUNT TO 0
                MOVB #$00,NUM_FLAG            ; SET NUM_FLAG TO 0 TO
                MOVB #$00,INVALID_KEY         ; RESET INVALID KEY FLAG

                ;SET UP INTRO TEXT TO LCD AND PAUSE HERE
                RTS                           ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TO RETRIEVE A PRESSED KEY FROM A MATRIX KEYBOARD, IF THIS ACTION HAPPENS,
SET A FLAG
InitLCD:        JSR delay3              ; WE NEED A SHORT DELAY HERE

        BCLR PortM,RS                 ; SEND A COMMAND
                LDAA #$30               ; Could be $38 too, 2 LINES AND 5X7 MATRIX
        JSR LCD_INPUT                 ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3              ; need extra delay at startup
        LDAA #$30               ; Could be $38 too, 2 LINES AND 5X7 MATRIX
        JSR LCD_INPUT                 ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                    ; WE NEED A SHORT DELAY HERE
        LDAA #$30                     ; Could be $38 too, 2 LINES AND 5X7 MATRIX
        JSR LCD_INPUT                 ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                    ; GIVE US A DELAY AGAIN

                LDAA #$38               ; Use 8 - words (command or data) and
        JSR LCD_INPUT                 ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                    ; NEED SHORT DELAY TO WAIT FOR COMMAND
TO COMPLETE

                LDAA #$0C               ; Turn on the display
        JSR LCD_INPUT                 ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                    ; NEED SHORT DELAY TO WAIT FOR COMMAND
TO COMPLETE

                LDAA #$01               ; clear the display and put the cursor
        JSR LCD_INPUT                 ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay               ; clear command needs more time
        JSR delay               ; to execute
        JSR delay                     ; NEED SHORT DELAY TO WAIT FOR COMMAND TO
COMPLETE
        RTS                           ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: LOAD A BIT INTO THE LCD (RS = 0 for commands OR RS = 1 FOR PRINT)
LCD_INPUT:
SPI_EF: BRCLR SPSR,$20,SPI_EF   ; WAIT FOR REGISTER EMPTY FLAG (SPIEF)
        STAA SPDR               ; OUTPUT COMMAND VIA SPI TO SIPO
CKFLG1          BRCLR SPSR,$80,CKFLG1         ; WAIT FOR SPI FLAG
        LDAA SPDR                     ; LOAD FROM SPI TO CLEAR FLAG
                NOP                           ; WAIT
                BCLR PortM, RCK               ; PULSE RCK
        NOP                           ; WAIT
        NOP                           ; WAIT
                BSET PortM, RCK               ; COMMAND NOW AVAILABEL FOR LCD
        NOP                           ; WAIT
        NOP             ; PROBABLY DON'T NEED TO WAIT
```

```
        NOP             ; BUT WE WILL, JUST IN CASE...
        BSET PortM, ENABLE      ; FIRE ENABLE
        NOP                     ; WE SHOULD WAIT AGAIN
        NOP                             ; UNTIL IT'S FINISHED
        BCLR PortM, ENABLE      ; ENABLE OFF
        JSR delay                       ; GIVE THE LCD TIME TO TAKE COMMAND IN
        RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TO RETRIEVE A PRESSED KEY FROM A MATRIX KEYBOARD, IF THIS ACTION HAPPENS,
SET A FLAG
;          AND STORE THE VALUE
GET_KEY:        LDX #KP_VALUE                   ; LOAD X WITH MEM ADDRESS FOR KP_VALUE

                STX CUR_PAD_VAL                 ; STORE THE ADDRESS OF THE FIRST KEYPAD
VALUE
                LDX #ROW                        ; LOAD X WITH THE INITIAL VALUE AT THE
ROW ADDRESS
                LDY #COLUMN                     ; LOAD Y WITH THE INITIAL VALUE AT THE
COLUMN ADDRESS
                ; NOW WE BEGIN OUR LOOPING
NEXT_ROW        LDAA 1,X+                       ; LOAD ACCUM A WITH CURRENT ROW VALUE
POST INCREMENT
NEXT_COLUMN     LDAB 1,Y+                       ; LOAD ACCUM Y WITH CURRENT COLUMN
VALUE POST INCREMENT
                STAA PortT                      ; SET THE CURRENT ROW TO HIGH VALUE
                STAB CUR_COLUMN                 ; STORE THE CURRENT COLUMN VALUE
                PSHA                            ; PUSH ONTO THE STACK OR IT WILL BE
LOST
                PSHB                            ; PUSH B ONTO THE STACK OR IT WILL BE
LOST
                NOP                             ; WAIT SOME TIME FOR PIN TO GO HI
                NOP                             ; WAIT SOME TIME FOR PIN TO GO HI
                NOP                             ; WAIT SOME TIME FOR PIN TO GO HI
                ABA                             ; ADD B TO A TO GET ALL PINS THAT
SHOULD BE HIGH
                LDAB PortT                      ; LOAD THE VALUE IN PortT INTO ACCUM B
                CBA                             ; CHECK THE CURRENT BIT IN PortT TO OUR
CURRENT COLUMN
                BEQ KEY_PRESSED                 ; IF THE KEY IS PRESSED THEN MAKE IT
SO!
                LDD CUR_PAD_VAL                 ; LOAD THE CUR_PAD_VAL INTO D
                ADDD #1                         ; ADD 1 TO D
                STD CUR_PAD_VAL                 ; STORE D BACK INTO THE PAD VALUE
                PULB                            ; GET B BACK FROM THE STACK FIRST
                PULA                            ; NOW RESTORE A FROM THE STACK
                CPY #COLUMN+4                   ; CHECK TO SEE IF WE'RE AT THE END OF
THE COLUMNS
                BNE NEXT_COLUMN                 ; IF NOT, THEN GO BACK AND TRY NEXT
COLUMN
                LDY #COLUMN                     ; IF WE ARE THEN RESET THE COLUMNS
                CPX #ROW+4                      ; CHECK TO SEE IF WE'RE AT THE END OF
THE ROWS
                BNE NEXT_ROW                    ; IF WE'RE NOT AT END OF ROWS, GO TO
NEXT ROW
                RTS                             ; RETURN FROM THE SUBROUTINE IF WE'VE
PROCESS ALL ROWS AND COLUMNS
KEY_PRESSED     PULB                            ; GET B BACK FROM THE STACK FIRST
                PULA                            ; NOW RESTORE A FROM THE STACK
                MOVB #$01,NUM_FLAG              ; SET NUM_FLAG SINCE A NUMBER WAS
PRESSED
                JSR KEY_RELEASE                 ; NOW WE NEED TO WAIT UNTIL THE KEYS
ARE RELEASED
```

```
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: WAIT UNTIL A PRESSED KEY IS RELEASED TO ELIMINATE BOUNCE AND DOUBLE
PRESSING
KEY_RELEASE:    MOVB #$F0,PortT                 ; SET ROWS 4,5,6,7 OF PortT TO HIGH
                NOP                             ; SHORT TIME WAITING FOR PINS TO GO
HIGH
                BRCLR PortT,$0F,FINISH          ; WHEN COLUMN 1-4 (PM0-PM3) IS CLEAR
THEN ALL KEYS
                                                ; HAVE BEEN RELEASED
                BRA KEY_RELEASE                 ; BRANCH BACK TO KEY RELEASE
FINISH          RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TO CHECK AND MAKE SURE WE HAVE A VALID KEY PRESSED
CHECK_KEY:      LDX CUR_PAD_VAL                 ; GET THE CURRENT KEYPAD VALUE ADDRESS
                LDAA X                          ; LOAD THE KEYPAD VALUE ADDRESS
                CMPA #$09                       ; WAS THIS KEY AN INVALID KEY?
                BGT INVALID                     ; IF IT WAS THEN SET THE FLAG
                RTS                             ; IF NOT RETURN FROM SUBROUTINE
INVALID         MOVB #$01,INVALID_KEY           ; SET THE INVALID KEY FLAG

                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: THIS SUBROUTINE IS USED TO LOAD A NEW DIGIT FROM THE USER INTO THE SYSTEM
INSERT_NUMBER:  BRSET OP_FLAG,$01,U1            ; IF THE FLAG IS SET GO TO U2
                LDD OPERAND1                    ; IF WE GOT HERE, LOAD OPERAND1
                JMP GO                          ; WE'VE GOT OPERAND1 PROCESS NOW
U1              LDD OPERAND2                    ; IF WE'RE HERE, THEN PROCESS OPERAND2
GO              MOVW #$0000,INPUT               ; RESET THE CURRENT VALUE IN INPUT1
                CPD #1000                       ; COMPARE ACMD TO 1000
                LBLO UNDER_1000                 ; IF IT'S LESS THAN 1000 THEN GO TO
UNDER_1000
                LDX #1000                       ; PLACE 1000 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 1000
UNDER_1000      CPD #100                        ; COMPARE ACMD TO 100
                LBLO UNDER_100                  ; IF IT'S LESS THAN 100 THEN GO TO
UNDER_100
                LDX #100                        ; PLACE 100 IN D
                IDIV                            ; DIVIDE THE NUMBER BY 100
                XGDX                            ; PUT THE REMAINDER IN X, QUOTIENT IN D
                LDY #1000                       ; LOAD Y WITH DECIMAL 1000
                EMUL                            ; MULTIPLY ACMD WITH INDEX Y TO MOVE
100S->1000S
                ADDD INPUT                      ; ADD THE VALUE IN INPUT TO ACMD
                STD INPUT                       ; STORE THE NEW VALUE IN INPUT
                XGDX                            ; GET THE REMAINDER BACK IN D
UNDER_100       CPD #10                         ; COMPARE ACMD TO 10
                LBLO UNDER_10                   ; IF IT'S LESS THAN 10 THEN GO TO
UNDER_10
                LDX #10                         ; PLACE 10 IN D
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; PUT THE REMAINDER IN X, QUOTIENT IN D
                LDY #100                        ; LOAD Y WITH 100
                EMUL                            ; MULTIPLY D * 100
                ADDD INPUT                      ; ADD THAT VALUE TO THE VALUE IN INPUT
                STD INPUT                       ; STORE THIS NEW VALUE IN INPUT
                XGDX                            ; MOVE THE REMAINDER BACK INTO D
UNDER_10        LDY #10                         ; LOAD Y WITH 10
                EMUL                            ; MULTIPLY THE REMAINDER WITH 10
                ADDD INPUT                      ; ADD D TO THE VALUE IN INPUT
```

```
                STD INPUT                       ; STORE THE VALUE IN ACMD IN INPUT
                LDD #$0000                      ; RESET D TO 0
                LDY CUR_PAD_VAL                 ; GET THE CURRENT KEYPAD VALUE ADDRESS
                LDAB Y                          ; LOAD B WITH THE ADDRESS IN Y
                ADDD INPUT                      ; ADD OUR KEYPAD VALUE TO THE INPUT
NUMBER
                BRSET OP_FLAG,$01,U2            ; IF THE FLAG IS SET GO TO U2
                STD OPERAND1                    ; STORE ACMD IN OPERAND1
                JMP IN_DONE                     ; WE'RE DONE, GO TO RTS
U2              STD OPERAND2                    ; STORE ACMD IN OPERAND2
IN_DONE         JSR DRAW_SCREEN                 ; LET'S UPDATE THE SCREEN AGAIN

                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
**********
OPERATION:      LDX CUR_PAD_VAL                 ; GET THE KEY PRESSED VALUE ADDRESS
                LDAA X                          ; LOAD THE KEY VALUE
                CMPA #$0F                       ; DID THE USER WANT NEGATIVE?
                BNE NOPE                        ; THEN IT'S NOT THE INVERT KEY
                ;JSR INVERT                     ; LET'S INVERT THE CURRENT OPERAND!
                JMP NOPE3                       ; GO TO NOPE3 TO GET OUT
NOPE            CMPA #$0C                       ; DID THE USER WANT CLEAR?
                BNE NOPE2                       ; IF NOT, TEST NEXT VALUE
                JSR BSPACE                      ; IF SO, THEN GO TO BACKSPACE SUB
                JMP NOPE3                       ; GO TO DRAW SCREEN AND RETURN
NOPE2           BCLR CLR_DIGIT,$01              ; C WASN'T PRESSED TWICE SO CLEAR THE
FLAG
                CMPA #$09                       ; WAS THIS KEY A 9?
                BGT USER_OP                     ; IF IT WAS 9 OR HIGHER, GO TO USER_OP
                RTS                             ; RETURN FROM SUBROUTINE
USER_OP         STAA OP                         ; STORE THE OPERATION FOR USE LATER
                BSET OP_FLAG,$01                ; SET THE OP_FLAG
NOPE3           JSR DRAW_SCREEN                 ; DRAW THE SCREEN
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
**********
CLEAR:
                MOVW #$0000,OPERAND1            ; CLEAR OPERAND 1 TO 0
                MOVW #$0000,OPERAND2            ; CLEAR OPERAND 2 TO 0
                MOVW #$0000,SOLUTION            ; CLEAR SOLUTION TO 0
                MOVW #$0000,SOLUTION2           ; CLEAR SOLUTION2 TO 0
                MOVB #$00,OP                    ; CLEAR THE CURRENT OP ALSO
                BCLR OP_FLAG,$01                ; CLEAR OP_FLAG
                JSR DRAW_SCREEN                 ; UPDATE THE DISPLAY
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
**********
ADDITION:
                LDD OPERAND1                    ; LOAD ACMD WITH OPERAND 1
                ADDD OPERAND2                   ; ADD OPERAND2 TO ACMD (OPERAND1),
STORE IN D
                STD SOLUTION                    ; STORE D (ANSWER) IN SOLUTION VARIABLE
                BSET SOLVED_FLAG,$01            ; SET THE SOLVED FLAG
                JSR DRAW_SCREEN                 ; UPDATE THE DISPLAY
                BCLR OP_FLAG,$01                ; CLEAR OP_FLAG
                MOVW #$0000,OPERAND1            ; CLEAR OPERAND 1 TO 0
                MOVW #$0000,OPERAND2            ; CLEAR OPERAND 2 TO 0
                MOVW #$0000,SOLUTION            ; CLEAR SOLUTION TO 0
                MOVW #$0000,SOLUTION2           ; CLEAR SOLUTION2 TO 0
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
**********
SUBTRACTION:
```

```
                LDD OPERAND1                    ; LOAD ACMD WITH OPERAND 1
                SUBD OPERAND2                   ; SUBTRACT OPERAND 2 FROM ACMD
(OPERAND1), STORE IN D
                STD SOLUTION                    ; STORE D (ANSWER) IN SOLUTION VARIABLE
                BSET SOLVED_FLAG,$01            ; SET THE SOLVED FLAG
                JSR DRAW_SCREEN                 ; UPDATE THE DISPLAY
                BCLR OP_FLAG,$01                ; CLEAR OP_FLAG
                MOVW #$0000,OPERAND1            ; CLEAR OPERAND 1 TO 0
                MOVW #$0000,OPERAND2            ; CLEAR OPERAND 2 TO 0
                MOVW #$0000,SOLUTION            ; CLEAR SOLUTION TO 0
                MOVW #$0000,SOLUTION2           ; CLEAR SOLUTION2 TO 0
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
MULTIPLICATION:
                LDD OPERAND1                    ; LOAD ACMD WITH OPERAND 1
                LDY OPERAND2                    ; LOAD INDEX Y WITH OPERAND 2
                ; USE EMULS (SIGNED 16 BY 16 MULTIPLY)  (D) X (Y) -> Y:D
                EMULS                           ; PERFORM MULTIPLICATION
                STD SOLUTION                    ; STORE THE LOWER WORD IN SOLUTION
                STY SOLUTION2                   ; STORE THE UPPER WORD IN SOLUTION 2
                BSET SOLVED_FLAG,$01            ; SET THE SOLVED FLAG
                JSR DRAW_SCREEN                 ; UPDATE THE DISPLAY
                BCLR OP_FLAG,$01                ; CLEAR OP_FLAG
                MOVW #$0000,OPERAND1            ; CLEAR OPERAND 1 TO 0
                MOVW #$0000,OPERAND2            ; CLEAR OPERAND 2 TO 0
                MOVW #$0000,SOLUTION            ; CLEAR SOLUTION TO 0
                MOVW #$0000,SOLUTION2           ; CLEAR SOLUTION2 TO 0
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DIVISION:
                ; USE IDIVS (SIGNED 16 X 16 INTEGER DIVIDE (D) / (X) -> X [REMAINDER
-> D]
                JSR DRAW_SCREEN                 ; UPDATE THE DISPLAY
                BCLR OP_FLAG,$01                ; CLEAR OP_FLAG
                MOVW #$0000,OPERAND1            ; CLEAR OPERAND 1 TO 0
                MOVW #$0000,OPERAND2            ; CLEAR OPERAND 2 TO 0
                MOVW #$0000,SOLUTION            ; CLEAR SOLUTION TO 0
                MOVW #$0000,SOLUTION2           ; CLEAR SOLUTION2 TO 0
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
BSPACE:         LDX CUR_PAD_VAL                 ; GET THE KEY PRESSED VALUE ADDRESS
                LDAA X                          ; LOAD THE KEY VALUE
                CMPA #$0C                       ; WAS THIS KEY A 'C'?
                BNE NO_BSPC                     ; IF IT WAS THEN SET THE FLAG
                BRSET CLR_DIGIT,$01,CLR_ALL     ; IF CLR FLAG IS SET THEN CLEAR ALL

                BRSET OP_FLAG,$01,B0            ; IF THE FLAG IS SET GO TO U2
                LDD OPERAND1                    ; NO FLAG? LOAD OPERAND1
                JMP B1                          ; SKIP LOADING OPERAND 2 SINCE WE
LOADED OPERAND1
B0              LDD OPERAND2                    ; FLAG? LOAD OPERAND 2

B1              LDX #0010                       ; PLACE TEN IN D
                IDIV                            ; DIVIDE OUR NUMBER BY 10
                BRSET OP_FLAG,$01,B3            ; IF THE FLAG IS SET GO TO U2
                STX OPERAND1                    ; NO FLAG? LOAD OPERAND1
                JMP B4                          ; SKIP LOADING OPERAND 2 SINCE WE
LOADED OPERAND1
B3              STX OPERAND2                    ; FLAG? LOAD OPERAND 2
```

```
B4              BCLR PortM,RS                   ; SEND A COMMAND TO LCD
                LDAA #$10                       ; SEND BACKSPACE CHARACTER TO DISPLAY
                JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                BSET PortM,RS                   ; SEND A COMMAND TO LCD
                LDAA #$20                       ; SEND BACKSPACE CHARACTER TO DISPLAY
                JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                BCLR PortM,RS                   ; SEND A COMMAND TO LCD
                LDAA #$10                       ; SEND BACKSPACE CHARACTER TO DISPLAY
                JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                BSET CLR_DIGIT,$01              ; SET THE CLEAR ALL FLAG
NO_BSPC         RTS                             ; RETURN FROM SUBROUTINE
CLR_ALL         JSR CLEAR                       ; SET CLEAR FLAG
                BCLR CLR_DIGIT,$01              ; CLEAR THE CLEAR_DIGIT FLAG
                RTS                             ; RETURN FROM SUBROUTINE

;*******************************************************************************
***********
INVERT:
                BRSET OP_FLAG,$01,I0            ; IF THE FLAG IS SET GO TO U2
                COM OPERAND1                    ; NO FLAG? LOAD OPERAND1
                JMP I1                          ; SKIP LOADING OPERAND 2 SINCE WE
LOADED OPERAND1
I0              COM OPERAND2                    ; FLAG? LOAD OPERAND 2
I1              RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CALCULATE:      BRCLR OP_FLAG,$01,CALC_DONE     ; IF THE OP FLAG ISN'T SET, GO TO
CALC_DONE
                LDX CUR_PAD_VAL                 ; GET THE CUR_PAD_VAL
                LDAA X                          ; LOAD THE ADDRESS INTO A
                CMPA #$0E                       ; COMPARE USER KEY TO E
                BNE CALC_DONE                   ; IF USER DIDN'T PRESS E, THEN LEAVE
SUB
                LDAB OP                         ; LOAD THE OPERATION VALUE
                CMPB #$0A                       ; WAS IT ADDITION?
                BNE SUBT                        ; NO? THEN TRY NEXT VALUE
                JSR ADDITION                    ; YES? THEN LETS ADD!
                JMP CALC_DONE                   ; GO TO CALC_DONE
SUBT            CMPB #$0B                       ; WAS IT SUBTRACTION?
                BNE DIV                         ; NO? THEN TRY NEXT VALUE
                JSR SUBTRACTION                 ; YES? THEN LETS SUBTRACT!
                JMP CALC_DONE                   ; GO TO CALC_DONE
DIV             CMPB #$0D                       ; WAS IT DIVISION?
                BNE MULT                        ; NO? THEN TRY NEXT VALUE
                JSR DIVISION                    ; YES? THEN LETS DIVIDE!
                JMP CALC_DONE                   ; GO TO CALC_DONE
MULT            CMPB #$0E                       ; WAS IT MULTIPLICATION?
                BNE CALC_DONE                   ; NO? THEN LEAVE SUB
                JSR MULTIPLICATION              ; YES? LET'S MULTIPLY!
CALC_DONE       RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: PRINT A STRING TO THE LCD (USES LCD_INPUT)
PRINT_STRING:
Loop1   LDAA 0,X                ; LOAD A CHARACTER INTO ACMA
        BEQ Done1               ; QUIT IF WE REACH A $00
        JSR LCD_INPUT           ; AND OUTPUT THE CHARACTER
        INX             ; GO TO NEXT CHARACTER
        BRA Loop1                       ; PROCESS NEXT CHARACTER
Done1           RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DRAW_SCREEN:    BCLR PortM, RS                  ; SEND A COMMAND TO LCD
```

```
                LDAA #$01                       ; CLEAR SCREEN COMMAND
                JSR LCD_INPUT                   ; SEND TO LCD
                LDAA #$02                       ; RETURN TO HOME COMMAND
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING1                    ; LOAD "1:" INTO DISPLAY
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB
                LDD OPERAND1                    ; LOAD OPERAND1 INTO ACMD
                JSR DISPLAY_NUMBER              ; DISPLAY OPERAND1
                LDX #STRING2                    ; PRINT A COUPLE SPACES
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB

                BRSET OP_FLAG,$01,SPCE          ; IF OP FLAG IS SET, GO TO SPCE
                LDAA #$20                       ; LOAD ACMA WITH ASCII BLANK
                JMP O_OP                        ; GO TO O_OP
SPCE            LDAA OP                         ; LOAD ACMA WITH THE OP VALUE
                JSR TO_ASCII                    ; CONVERT THE VALUE TO ASCII
O_OP            JSR LCD_INPUT                   ; PRINT IT TO LCD

                LDX #STRING3                    ; PRINT ANOTHER SPACE
                JSR PRINT_STRING                ; SEND SPACE TO LCD
                LDX #STRING4                    ; PRINT "2:" TO SCREEN
                JSR PRINT_STRING                ; SEND IT TO LCD
                LDD OPERAND2                    ; LOAD ACMD WITH OPERAND2
                JSR DISPLAY_NUMBER              ; PRINT IT TO SCREEN

                BCLR PortM, RS                  ; SENT A COMMAND TO LCD
                LDAA #$C0                       ; GO TO SECOND LINE TO PRINT
                JSR LCD_INPUT                   ; SEND COMMAND

                ; PRINT THE CURRENT COUNT
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING5                    ; PRINT 'ANS: '
                JSR PRINT_STRING                ; LET'S PRINT THE STRING NOW

                JSR DISPLAY_SOLUTION            ; PRINT THE SOLUTION TO THE SCREEN

                BCLR PortM, RS                  ; SEND A COMMAND TO LCD
                LDAA #$0E                       ; LCD DISPLAY ON, CURSOR BLINKING
                JSR LCD_INPUT                   ; PRINT COMMAND TO LCD

                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DIRECTIONS:     BCLR PortM, RS                  ; SEND A COMMAND TO LCD
                LDAA #$01                       ; CLEAR SCREEN COMMAND
                JSR LCD_INPUT                   ; SEND TO LCD
                LDAA #$02                       ; RETURN TO HOME COMMAND
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING6                    ; PRINT DIRECTION
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB
                BCLR PortM, RS                  ; SENT A COMMAND TO LCD
                LDAA #$C0                       ; GO TO SECOND LINE TO PRINT
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING7                    ; PRINT DIRECTION
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB TO PPRINT
                JSR delay2                      ; DELAY A BIT
                JSR delay2                      ; DELAY A BIT

                BCLR PortM, RS                  ; SEND A COMMAND TO LCD
                LDAA #$01                       ; CLEAR SCREEN COMMAND
```

```
JSR LCD_INPUT           ; SEND TO LCD
LDAA #$02               ; RETURN TO HOME COMMAND
JSR LCD_INPUT           ; SEND COMMAND
BSET PortM, RS          ; LET'S PRINT TO LCD
LDX #STRING8            ; PRINT DIRECTION
JSR PRINT_STRING        ; GO TO PRINT_STRING SUB
BCLR PortM, RS          ; SENT A COMMAND TO LCD
LDAA #$C0               ; GO TO SECOND LINE TO PRINT
JSR LCD_INPUT           ; SEND COMMAND
BSET PortM, RS          ; LET'S PRINT TO LCD
LDX #STRING9            ; PRINT DIRECTION
JSR PRINT_STRING        ; GO TO PRINT_STRING SUB TO PPRINT
JSR delay2              ; DELAY A BIT
JSR delay2              ; DELAY A BIT

BCLR PortM, RS          ; SEND A COMMAND TO LCD
LDAA #$01               ; CLEAR SCREEN COMMAND
JSR LCD_INPUT           ; SEND TO LCD
LDAA #$02               ; RETURN TO HOME COMMAND
JSR LCD_INPUT           ; SEND COMMAND
BSET PortM, RS          ; LET'S PRINT TO LCD
LDX #STRING10           ; PRINT DIRECTION
JSR PRINT_STRING        ; GO TO PRINT_STRING SUB
BCLR PortM, RS          ; SENT A COMMAND TO LCD
LDAA #$C0               ; GO TO SECOND LINE TO PRINT
JSR LCD_INPUT           ; SEND COMMAND
BSET PortM, RS          ; LET'S PRINT TO LCD
LDX #STRING11           ; PRINT DIRECTION
JSR PRINT_STRING        ; GO TO PRINT_STRING SUB TO PPRINT
JSR delay2              ; DELAY A BIT
JSR delay2              ; DELAY A BIT

BCLR PortM, RS          ; SEND A COMMAND TO LCD
LDAA #$01               ; CLEAR SCREEN COMMAND
JSR LCD_INPUT           ; SEND TO LCD
LDAA #$02               ; RETURN TO HOME COMMAND
JSR LCD_INPUT           ; SEND COMMAND
BSET PortM, RS          ; LET'S PRINT TO LCD
LDX #STRING12           ; PRINT DIRECTION
JSR PRINT_STRING        ; GO TO PRINT_STRING SUB
BCLR PortM, RS          ; SENT A COMMAND TO LCD
LDAA #$C0               ; GO TO SECOND LINE TO PRINT
JSR LCD_INPUT           ; SEND COMMAND
BSET PortM, RS          ; LET'S PRINT TO LCD
LDX #STRING13           ; PRINT DIRECTION
JSR PRINT_STRING        ; GO TO PRINT_STRING SUB TO PPRINT
JSR delay2              ; DELAY A BIT
JSR delay2              ; DELAY A BIT

BCLR PortM, RS          ; SEND A COMMAND TO LCD
LDAA #$01               ; CLEAR SCREEN COMMAND
JSR LCD_INPUT           ; SEND TO LCD
LDAA #$02               ; RETURN TO HOME COMMAND
JSR LCD_INPUT           ; SEND COMMAND
BSET PortM, RS          ; LET'S PRINT TO LCD
LDX #STRING14           ; PRINT DIRECTION
JSR PRINT_STRING        ; GO TO PRINT_STRING SUB
BCLR PortM, RS          ; SENT A COMMAND TO LCD
LDAA #$C0               ; GO TO SECOND LINE TO PRINT
JSR LCD_INPUT           ; SEND COMMAND
BSET PortM, RS          ; LET'S PRINT TO LCD
LDX #STRING15           ; PRINT DIRECTION
JSR PRINT_STRING        ; GO TO PRINT_STRING SUB TO PPRINT
```

```
                JSR delay2                      ; DELAY A BIT
                JSR delay2                      ; DELAY A BIT

                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TO TAKE A VALUE IN ACMD AND CONVERT IT TO ITS CORRESPONDING ASCII DECIMAL
VALUE TO DISPLAY
; NOTE: THIS SUB ASSUMES A VALID NUMBER IS LOADED INTO ACMD AND IS TO BE OUTPUT VIA
LCD ASCII
DISPLAY_NUMBER:
                BSET PortM, RS                  ; PRINT CHARACTER TO LCD
                LDX #1000                       ; PLACE 1000 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 1000
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ONTO STACK
                JSR TO_ASCII                    ; CONVERT A TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #100                        ; PLACE 100 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 100
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ONTO THE STACK
                JSR TO_ASCII                    ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X FROM STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #10                         ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR TO_ASCII                    ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                TBA                             ; TRANSFER B INTO A
                JSR TO_ASCII                    ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TO TAKE A VALUE IN ACMD AND CONVERT IT TO ITS CORRESPONDING ASCII DECIMAL
VALUE TO DISPLAY
; NOTE: THIS SUB ASSUMES A VALID NUMBER IS LOADED INTO ACMD AND IS TO BE OUTPUT VIA
LCD ASCII
DISPLAY_SOLUTION:
                BSET PortM, RS                  ; PRINT CHARACTER TO LCD
                LDY SOLUTION2                   ; GET THE UPPER 2 BYTES
                LDD SOLUTION                    ; GET THE LOWER 2 BYTES
                LDX #10000                      ; PLACE 10000 IN X
                STX DIVISOR                     ; STORE THE DIVISOR
                EDIVS                           ; DIVIDE THE NUMBER BY 10000
                PSHD                            ; PUSH THE REMAINDER FOR USE LATER
                XGDY                            ; MOVE THE REMAINDER TO Y AND THE
QUOTIENT TO D
                LDX #10                         ; LOAD INDEX X WITH 10
```

```
                IDIV                            ; DIVIDE D BY 10 (PUT QUOTIENT IN X,
REMAINDER IN D)
                PSHD                            ; PUSH D ONTO THE STACK
                XGDX                            ; SWICH X AND D
                TBA                             ; TRANSFER B TO A
                JSR TO_ASCII                    ; CONVERT ACMA TO ASCII
                JSR LCD_INPUT                   ; OUTPUT TO LCD
                PULD                            ; PULL THE REMAINDER BACK OFF STACK
                TBA                             ; MOVE ACMB TO ACMA
                JSR TO_ASCII                    ; CONVERT TO ASCII
                JSR LCD_INPUT                   ; OUTPUT TO LCD
                PULD                            ; PULL THE ORIGINAL REMAINDER BACK NOW
                LDX #1000                       ; PLACE 1000 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 1000
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ONTO THE STACK
                JSR TO_ASCII                    ; CONVERT ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF THE STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #100                        ; PLACE 100 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 100
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ONTO THE STACK
                JSR TO_ASCII                    ; CONVERT ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF THE STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #10                         ; PLACE 10 IN D
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ONTO THE STACK
                JSR TO_ASCII                    ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF THE STACK
                XGDX                            ; EXCHANGE D WITH X
                TBA                             ; TRANSFER B TO A
                JSR TO_ASCII                    ; CONVERT ACMA TO ASCII VALUE
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TAKE THE VALUE ACMA AND UPDATE IT TO THE ASCII VALUE OF THAT CHARACTER
TO_ASCII:       TAB
                LDY #ASCII                      ; LOAD THE BEGINNING ADDRESS OF TABLE
INTO X
                ABY                             ; ADD B TO THE X INDEX
                LDAA Y                          ; LOAD THE ADDRESS OF INDEX X INTO
ACCUM A
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
delay           LDY #8000                         ; COMMAND DELAY ROUTINE.  WAY TO
LONG. OVERKILL!
A2:     DEY             ; BUT WE DO NEED TO WAIT FOR THE LCD CONTROLLER
        BNE A2                  ; TO DO IT'S THING.  HOW MUCH TIME?
        RTS                             ; RETURN FROM SUBROUTINE
```

```
delay2          LDY #$F000                      ; LONG DELAY ROUTINE.  ADJUST AS
NEEDED.
        PSHA            ; SAVE ACMA
A3:     LDAA #$8F                       ; LONG DELAY LOAD ACMA WITH 8F (NESTED LOOP)
AB:     DECA                            ; DECREMENT A
        BNE AB                  ; BRANCH TO AB IF NOT EQUAL
        DEY                             ; DECREMENT Y
        BNE A3                  ; BRANCH TO A3 IF NOT EQUAL
        PULA            ; GET ACMA BACK
        RTS                             ; RETURN FROM SUBROUTINE

delay3          LDAA #$0F                       ; LOAD 15 (F) INTO ACMA
AA6:    LDY #$FFFF                       ; LOAD Y WITH FFFF (Blink Delay routine.)
A6:     DEY             ; DECREMENT Y
        BNE A6                          ; BRANCH TO A6 IF NOT EQUAL
        DECA                            ; DECREMENT A
        BNE AA6                 ; BRANCH TO AA6 IF NOT EQUAL
        RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; TC0 INTERRUPT SUBROUTINE
ISR_TC0:        LDD TC0                         ; INTERRUPT READS THE FLAG SO THIS
WRITE CLEARS THE FLAG
                ADDD #3750                      ; ADD THE EQUIVALENT .1 SECOND CNT TO
REGISTER D
                STD TC0                         ; UPDATE TC0 MEMORY TO NEW VALUE
                PSHA                            ; SAVE A ON THE STACK
                LDAA TIME_COUNT                 ; LOAD THE VALUE OF TIME_COUNT INTO A
                CMPA #100                       ; IF TIME_COUNT = 100 THEN WE HAVE 1
SECOND
                BNE TMR_UPDATE                  ; IF WE'RE NOT AT 100 YET, GOTO
TMR_UPDATE LINE
                MOVB #$01,TMR_FLAG              ; TURN ON OUR TIMER FLAG
                MOVB #$00,TIME_COUNT            ; RESET OUR TIMER COUNT BACK TO ZERO
                PULA                            ; PUL A BACK OFF THE STACK
PAUSED          RTI                             ; RETURN FROM THE INTERRUPT
TMR_UPDATE      ADDA #01                        ; INCREMENT THE VALUE IN A
                STAA TIME_COUNT                 ; STORE A BACK INTO TIME_COUNT
                PULA                            ; PULL A BACK OFF THE STACK
                RTI                             ; RETURN FROM THE INTERRUPT
;*******************************************************************************
*******
                ORG $FFEE                       ; VECTOR ADDRESS FOR TC0 INTERRUPT
                FDB ISR_TC0                     ; ISR_TIMER IS A LABEL FOR THE
INTERRUPT SUBROUTINE

;*******************************************************************************
******
; Have the Assembler put the solution data in the look-up table

                ORG $5500                       ; The look-up table is at $5500

TABLE:          DC.B $00, $01, $02, $03, $04  ; Define data table of mappings to each
of the
                DC.B $05, $06, $07, $08, $09  ; matrix keypad values.
                DC.B $0A, $0B, $0C, $0D, $0E  ; Memory locations correspond to their
values
                DC.B $0F                        ; i.e. $5500 = 0, $5501 = 1, etc

ASCII:          DC.B $30, $31, $32, $33, $34  ; Define data table of mappings to each
of the
                DC.B $35, $36, $37, $38, $39  ; ascii values for the keypad
```

```
                DC.B $2B, $2D, $20, $2F, $2A  ; Memory locations correspond to their
values
                DC.B $20                      ; i.e. $5500 = 0, $5501 = 1, etc


ROW:            DC.B $10, $20, $40, $80       ; PortT OUTPUT VALUES FOR MATRIX KEYPAD
ROWS
COLUMN:         DC.B $01, $02, $04, $08       ; PortM INPUT VALUES FOR MATRIX KEYPDA
COLUMNS

KP_VALUE:       DC.B $01, $02, $03, $0A       ; KEY VALUES FROM KEYPAD FOR ITERATING
THROUGH
                DC.B $04, $05, $06, $0B
                DC.B $07, $08, $09, $0C
                DC.B $00, $0F, $0E, $0D

STRING1         FCC "1:"                      ; CREATE A STRING FOR PAUSED
                DC.B $00
STRING2         FCC ""                        ; CREATE A STRING WITH THE RUN
                DC.B $00
STRING3         FCC " "                       ; CREATE A STRING WITH THE UP
                DC.B $00
STRING4         FCC "2:"                      ; CREATE A STRING WITH THE DOWN
                DC.B $00
STRING5         FCC "ANS: "                   ; CREATE A STRING FOR THE TIME LINE
                DC.B $00
STRING6         FCC "    ECE 367     "        ; CREATE A STRING
                DC.B $00
STRING7         FCC "  SPRING 2013   "        ; CREATE A STRING
                DC.B $00
STRING8         FCC "   LAB EXP 8    "        ; CREATE A STRING
                DC.B $00
STRING9         FCC "   CALCULATOR   "        ; CREATE A STRING
                DC.B $00
STRING10        FCC "A - ADDITION    "        ; CREATE A STRING
                DC.B $00
STRING11        FCC "B - SUBTRACTION "        ; CREATE A STRING
                DC.B $00
STRING12        FCC "C - CLEAR       "        ; CREATE A STRING
                DC.B $00
STRING13        FCC "D - DIVISION    "        ; CREATE A STRING
                DC.B $00
STRING14        FCC "E - MULTIPLY    "        ; CREATE A STRING
                DC.B $00
STRING15        FCC "F - NEGATIVE    "        ; CREATE A STRING
                DC.B $00

; End of code

; Define Power-On Reset Interrupt Vector - Required for all programs!

; AGAIN - OP CODES are at column 9
                ORG $FFFE                     ; $FFFE, $FFFF = Power-On Reset Int.
Vector Location
                FDB START                     ; Specify instruction to execute on
power up

                END                           ; (Optional) End of source code

; Labels start in the first column  (left most column = colunm 1)
; OP CODES are at column 9
; COMMENTS follow a ";" symbol
; Blank lines are allowed (Makes the code more readable)
```