```
; Author: Shuaib Shameem
; ECE 367 - Microprocessor-Based Design
; Experiment 9  - Math Flash Card System
; March 28, 2012

; Define symbolic constants
PortT    EQU    $0240
PORTT   EQU   $0240
DDRT   EQU    $0242
PortM   EQU   $0250
DDRM   EQU    $0252
RS     EQU    $01    ; Register Select (RS) at PT0 (0 = command, 1= Data)
ENABLE EQU    $02     ; LCD ENABLE at PT1
RCK    EQU    $08    ; RCK connect to PT2
SPCR1   EQU   $00D8
SPCR2  EQU    $00D9
SPIB    EQU   $00DA
SPSR    EQU   $00DB
SPDR    EQU   $00DD
INITRG EQU    $0011
INITRM EQU    $0010
INITEE EQU    $0012
CLKSEL EQU    $39
PLLCTL EQU    $3A
CRGFLG EQU    $37
SYNR   EQU    $34
REFDV    EQU    $35
TCNT    EQU   $44
TSCR1   EQU   $46
;
; RAM Variables
;
        ORG   $3800
COUNT    EQU    $3800
PRINT   EQU   $3801
KEY     EQU   $3802    ; - Upper 8-bits of the Variable \
KEY_1   EQU   $3803    ; - Lower 8-bits of the Variable / 16-bits
GEN     EQU   $3804
NEGCH   EQU   $3805
PORTMSK EQU   $3806
KCOUNT  EQU   $3807
DIGIT1  EQU   $3808
DIGIT2  EQU   $3809
SOLU    EQU   $380A    ; - Upper 8-bits of the Variable \
SOLU_L  EQU   $380B    ; - Lower 8-bits of the Variable / 16-bits
FANS    EQU   $380C
SKIPMSK EQU   $3810


; Initialize the NanoCore12:
; The main code begins here. Note the START Label
;
      ORG   $4000         ; Beginning of Flash EEPROM
START  LDS   #$3FCE     ; 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 = 24 MHz
;
      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 Interrupts
;
      LDAA  #$0F           ; Make PortT Bits 7-4 output
```

```
      STAA  DDRT
      LDAA  #$22
      STAA  SPIB           ; SPI clocks a 1/24 of E-Clock
        MOVB  #$3F, DDRM      ; Setup PortM data direction
;
; Setup for Master, enable, high speed SPI, and Built-in Timer
;
      LDAA  #$50
        STAA  SPCR1
      LDAA  #$00
      STAA  SPCR2
      LDAA  #$80
      STAA  TSCR1
;
; Initialize Variables to $00
;
        BCLR  PRINT, $FF
        BCLR  KEY,   $FF
        BCLR  KEY_1, $FF
      BCLR  GEN,   $FF
      BCLR  FANS,  $FF
      BCLR  SOLU,  $FF
      BCLR  SOLU_L,$FF
      BCLR  SKIPMSK,$FF
;
; Initialize the LCD Display
;
        LDAA  #00
        BSET  PortM, RCK    ; Set RCK to Idle HIGH

        JSR   InitLCD     ; Initialize the LCD

;
; User Interface
;
Loop0    LDX   #String1        ; Load base address of String1
        JSR   PrintString

      LDAA  #$C0            ; First line is done jump to line 2
        JSR    Command

      LDX   #String2          ; Load base address of String2
        JSR   PrintString

        JSR    delay2         ; Let's display the message a while

        JSR   BlinkDisp         ; Blink the display 4 times

Begin   JSR   ClearDisp         ; Clear the display

      LDX   #String3          ; Load base address of String3
        JSR   PrintString

      LDAA  #$C0          ; First line is done jump to line 2
        JSR    Command

      LDX   #String4          ; Load base address of String4
        JSR   PrintString

        JSR    delay2         ; Let's display the message a while

        JSR   ShiftSecondLine   ; Shift the second line to left

        LDX   #String5        ; Load base address of String5
        JSR   PrintString

        JSR    delay2         ; Let's display the message a while

        JSR   ShiftSecondLine

        LDX   #String6        ; Load base address of String6
        JSR   PrintString
```

```
         JSR    delay2         ; Let's display the message a while


        JSR   ShiftSecondLine

        LDX   #String7         ; Load base address of String7
        JSR   PrintString

         JSR    delay2         ; Let's display the message a while


        JSR   ShiftSecondLine

        LDX   #String8         ; Load base address of String8
        JSR   PrintString

      JSR    delay2         ; Let's display the message a while


Select   BCLR  GEN, $FF          ; Clear the GEN variable
        JSR   GetKey            ; Get the Keypad input

        JSR   ClearDisp         ; Clear the Display

        LDX   #Strin13         ; Load base address of String13
        JSR   PrintString

         LDAA  #$C0          ; Jump to line 2
        JSR    Command

        JSR   Method            ; Determine what arithmetic to use
        nop
        nop
        BRSET GEN, $FF, Select  ; If a wrong key is pressed, try again

        LDAA  #$3D              ; Print the "=" sign
      JSR   Print

      JSR   ReleaseWait       ; Wait for key release

      JSR   CheckAns          ; Chech if the Answer is correct or not

        JMP   Begin             ; Start Over
;
; ====================================================================================
;
; SubRoutines
;
; Print Digit1
;
PDig1   LDAA  DIGIT1            ; Load Digit1 on Accl A
        BMI   PrintNeg1         ; Branch if the MSB is set, hence the number is negative
        JSR   AconvP            ; Convert the value of Accl A to Ascii and Print
        JMP   con1
PrintNeg1
        LDAA  #$2D
        JSR   Print             ; Print the "-" sign if negative
        LDAA  DIGIT1            ; Load Digit1 on Accl A
        COMA                    ; Get Two's compliment
        INCA
        JSR   AconvP            ; Convert the value of Accl A to Ascii and Print
con1    RTS
;
; Print Digit2
;
PDig2   LDAA  DIGIT2            ; Load Digit2 on Accl A
      BMI   PrintNeg2         ; Branch if the MSB is set, hence the number is negative
        JSR   AconvP            ; Convert the value of Accl A to Ascii and Print
        JMP   con2
PrintNeg2
        LDAA  #$2D              ; Print the "-" sign if negative
```

```
        JSR   Print             ; Load Digit2 on Accl A
        LDAA  DIGIT2            ; Get Two's compliment
        COMA
        INCA
        JSR   AconvP            ; Convert the value of Accl A to Ascii and Print
con2    RTS
;
; Choose and do the Arithmetic Method
;
Method: JSR   RanNum            ; Generate Random Numbers
        LDAA  KEY_1             ; Load KEY_1 on Accl A
        CMPA  #$01              ; Compare Accl A to hex $01
        BEQ   Add               ; Branch to Add if Key 1 is pressed
        CMPA  #$02
        BEQ   Sub               ; Branch to Sub if Key 2 is pressed
        CMPA  #$03
        BEQ   Mult              ; Branch to Mult if Key 3 is pressed
        CMPA  #$04
        BEQ   Divi              ; Branch to Divi if Key 4 is pressed

        BSET  GEN, $FF          ; Set GEN if some other key has been pressed
        JSR   ShiftSecondLine   ; Shift line to to the left
        LDX   #Strin12          ; Load base address of Strin12
        JSR   PrintString       ; Print the String
        RTS
;
; Add the Numbers
;
Add:    LDAA  DIGIT1            ; Load Digit1 on Accl A
        ADDA  DIGIT2            ; Add Accl A and Digit2
        STAA  SOLU_L            ; Store Accl A to Low bits of SOL
        JSR   PDig1             ; Print Digit1
         LDAA  #$2B              ; Print the "+" sign
      JSR   Print
        JSR   PDig2             ; Print Digit2
        RTS
;
; Subtract the Numbers
;
Sub:    LDAA  DIGIT1            ; Load Digit1 on Accl A
        SUBA  DIGIT2            ; Sub Digit2 from Accl A
        STAA  SOLU_L            ; Store Accl A to Low bits of SOL
        JSR   PDig1             ; Print Digit1
      LDAA  #$2D              ; Print the "-" sign
      JSR   Print
      JSR   PDig2             ; Print Digit2
        RTS
;
; Multiply the Numbers
;
Mult:   LDAA  DIGIT1            ; Load Digit1 on Accl A
        LDAB  DIGIT2            ; Load Digit2 on Accl B
        MUL                     ; Multiply A and B
        STD   SOLU              ; Store D to SOLU
        JSR   PDig1             ; Print Digit1
        LDAA  #$2A              ; Print the "*" sign
      JSR   Print
      JSR   PDig2             ; Print Digit2
        RTS
;
; Divide the Numbers
;
Divi:   LDAA  DIGIT1            ; Load Digit1 on Accl A
        LDAB  DIGIT2            ; Load Digit2 on Accl B
        IDIV                    ; Divide
        STD   SOLU              ; Store D to SOLU
        JSR   PDig2             ; Print Digit1
         LDAA  #$2F              ; Print the "/" sign
      JSR   Print
      JSR   PDig2             ; Print Digit2
        RTS
;
```

```
; Check if the Answer is Correct of not
;
CheckAns:
        MOVB  #$00, FANS        ; Clear FANS
        MOVB  #$00, NEGCH       ; Clear NEGCH
V0      JSR   ReleaseWait       ; Wait for the Key Release
        JSR   GetKey            ; Get the value of Key pressed

        LDAA  KEY_1             ; Load KEY_1 on Accl A
        CMPA  #$0C              ; Branch to check Answer if A = $0C
        BEQ   V1
        CMPA  #$0B              ; Branch to negative input if A = $0B
        BEQ   V2
        CMPA  #$0A              ; If Any other key than Numbers, E and F,
        BGE   V0                ; try again

        LDAA  FANS              ; Load FANS to Accl A
        LDAB  #10               ; Load #10 on Accl B
        MUL                     ; Multiply A and B
        STAB  FANS              ; Store B back to FANS to get the tenth place
                                ; of the original value in FANS

        LDAA  KEY_1             ; Load KEY_1 on Accl A
        ADDA  FANS              ; Add Accl A and FANS
        STAA  FANS              ; Store A back to FANS

        LDAA  KEY_1             ; Load KEY_1 on Accl A
        JSR   AconvP            ; Convert the value to Ascii and Print
        JMP   V0                ; Go back to V0

V2      MOVB  #$FF, NEGCH       ; Set NEGCH
        LDAA  #$2D              ; Print the "-" Sign
        JSR   Print
        JMP   V0                ; Go Back to V0

V1      LDAA  FANS              ; Load FANS on Accl A
        BRCLR NEGCH, $FF, PosSol; Branch if NEGCH is Clear, the solution is Positive,
        COMA                    ; Two's Compliment A
        INCA
PosSol  CMPA  SOLU_L            ; Compare A to the Low bits of SOLU
        BNE   IncAns            ; If not equal, the answer is incorrect

         LDAA  #$C0          ; Jump to line 2
        JSR    Command
      LDX   #Strin10          ; Load base address of Strin10
        JSR   PrintString      ; Print the String
        JSR   delay2
        JSR   BlinkDisp         ; Blink the Display
        RTS

IncAns  LDAA  #$C0          ; Jump to line 2
        JSR    Command
       LDX   #Strin11        ; Load base address of Strin11
        JSR   PrintString       ; Print the String
        JSR   delay2
        JSR   BlinkDisp         ; Blink the Display
        RTS
;
; Random Number Generator
;
RanNum  LDD TCNT
Loop1   CMPB #10
        BLO  Save1
        SUBB #10
        BRA  Loop1
Save1   STAB DIGIT1
Loop2   CMPA #10
        BLO  Save2
        SUBA #10
        BRA  Loop2
Save2   STAA DIGIT2
;
```

```
; Initialize the LCD
;
InitLCD  JSR   delay3
      LDAA  #$30       ; Could be $38 too.
      JSR   Command
      JSR   delay3   ; need extra delay at startup
      LDAA  #$30     ; see data sheet. This is way
      JSR   Command    ; too much delay
      JSR   delay3
      LDAA  #$30
      JSR   Command
      LDAA  #$38       ; Use 8 - words (command or data) and
      JSR   Command      ; and both lines of the LCD
      LDAA  #$0C       ; Turn on the display
      JSR   Command
      LDAA  #$01       ; clear the display and put the cursor
      JSR   Command      ; in home position (DD RAM address 00)
      JSR   delay    ; clear command needs more time
      JSR   delay      ; to execute
      JSR   delay
      RTS
;
; Convert a hex to Ascii and Print the number
;
AconvP  LDAB  #$30          ; Load $30 on Accl B
        ABA                 ; Add A and B
      JSR   Print         ; Print Accl A
      RTS
;
; Print or Command
;
Print   BSET  PRINT, $FF
        JMP   spi_a
Command  BCLR  PRINT, $FF
spi_a: BRCLR SPSR,  $20,spi_a    ; Wait for register empty flag (SPIEF)
;   LDAB  SPDR          ; Read the SPI data register. This clears the flag automatically
        STAA  SPDR          ; Output command  via SPI to SIPO
CKFLG1  BRCLR SPSR,  $80, CKFLG1  ; Wait for SPI Flag
        LDAA  SPDR
        NOP                        ; Wait
        BCLR  PortM, RCK          ; Pulse RCK
        NOP
      NOP
      NOP
        BSET  PortM, RCK          ; Command now available for LCD
        BRCLR PRINT, $FF, ComL
        BSET  PortM, RS
        JMP   F1
ComL  BCLR  PortM, RS         ; RS = 0 for commands
F1     NOP
      NOP                     ; Probably do not need to wait
      NOP                     ; but we will, just in case ...
      BSET  PortM, ENABLE       ; Fire ENABLE
      NOP                     ; Maybe we will wait here too ...
      NOP
      NOP
      NOP
      BCLR  PortM, ENABLE     ; ENABLE off
      JSR   delay
      RTS
;
;  Blink the Display 4 times
;
BlinkDisp
        MOVB  #$04, COUNT       ; Initialize a counter
A4    LDAA  #$08             ; Turn off display but keep memory values
      JSR   Command
      JSR   delay3
      LDAA  #$0C             ; Turn on display. So, we Blinked!
      JSR   Command
      JSR   delay3
      DEC   COUNT
```

```
      BNE    A4            ; Blink 4 times
      RTS
;
; Clear the Display
;
ClearDisp
        LDAA  #$01               ; Clear the display and send cursor home
      JSR    Command
      JSR    delay              ; Clear needs more time so 3 delays
      JSR    delay
      JSR    delay
        RTS
;
; Print the String at the address loaded at X
;
PrintString
Loop7   LDAA  0,X         ; Load a character into ACMA
      BEQ    Done7              ; quit when if last character is $00
      JSR   Print             ; and output the character
      INX                      ; let's go get the next character
      BRA    Loop7
Done7   RTS
;
; Shift the second line to the left
;
ShiftSecondLine
        LDAA  #$C0          ; Jump to line 2
        JSR    Command
        LDAA  #$0C              ; Shift the Line to the left
        JSR   Command
        JSR   delay2            ; Delay it by some
        RTS
;
; Get the Value of the Key pressed
;
ReleaseWait
        BCLR SKIPMSK, $FF
        BSET SKIPMSK, $F0
GetKey: BCLR KEY, $FF         ; Clear variable KEY contents
        BCLR KEY_1, $FF
        LDAA #$0F          ; Load Acc. A with $F0
        STAA PORTT         ; Output high on all rows
        BRSET PORTT, $80, *; Check Column 1 for pressed key
        NOP
        BRSET PORTT, $40, *; Check Column 2 for pressed key
        NOP
        BRSET PORTT, $20, *; Check Column 3 for pressed key
        NOP
        BRSET PORTT, $10, *; Check Column 4 for pressed key
        BRSET SKIPMSK, $F0, RT
GKEY:   LDAA #$08          ; Once all keys are released, load Acc. A with $80
        STAA PORTT         ; Output high on row 1
        JSR sdelay
        BRSET PORTT, $80, KEY1    ;If high, key 1 was pressed
        NOP
        BRSET PORTT, $40, KEY2    ;If high, key 2 was pressed
        NOP
        BRSET PORTT, $20, KEY3    ;If high, key 3 was pressed
        NOP
;        BRSET SKIPMSK, $0F, Skip1
;        BRSET PORTT, $10, KEYA    ;If high, key A was pressed
Skip1   LDAA #$04          ; No key press yet, load Acc. A with $40
        STAA PORTT         ; Output high on row 2
        JSR sdelay
        BRSET PORTT, $80, KEY4    ;If high, key 4 was pressed
        NOP
        BRSET PORTT, $40, KEY5    ;If high, key 5 was pressed
        NOP
        BRSET PORTT, $20, KEY6    ;If high, key 6 was pressed
        NOP
;        BRSET SKIPMSK, $0F, Skip2
        BRSET PORTT, $10, KEYB    ;If high, key B was pressed
```

```
Skip2   LDAA #$02          ; No key press yet, load Acc. A with $20
        STAA PORTT         ; Output high on row 3
        JSR sdelay
        BRSET PORTT, $80, KEY7    ;If high, key 7 was pressed
        NOP
        BRSET PORTT, $40, KEY8    ;If high, key 8 was pressed
        NOP
        BRSET PORTT, $20, KEY9    ;If high, key 9 was pressed
        NOP
;        BRSET SKIPMSK, $0F, Skip3
        BRSET PORTT, $10, KEYC    ;If high, key C was pressed
Skip3   LDAA #$01          ; No key press yet, load Acc. A with $10
        STAA PORTT         ; Output high on row 4
        JSR sdelay
        BRSET PORTT, $80, KEY0    ;If high, key 0 was pressed
;        BRSET SKIPMSK, $0F, Skip4
;        BRSET PORTT, $40, KEYF    ;If high, key F was pressed
;        BRSET PORTT, $20, KEYE    ;If high, key E was pressed
;        BRSET PORTT, $10, KEYD    ;If high, key D was pressed
Skip4   LBRA GKEY         ; No key press, check again
RT:     BCLR SKIPMSK, $FF
        RTS

;
; Set of labels to set KEY to the pressed key's value
;            OR to branch to a relevant routine
;
KEY1:   BSET KEY_1, $01       ; Set KEY to 1
        RTS                  ; Return to GETKEY's calling routine

KEY2:   BSET KEY_1, $02       ; Set KEY to 2
        RTS                  ; Return to GETKEY's calling routine

KEY3:   BSET KEY_1, $03       ; Set KEY to 3
        RTS                  ; Return to GETKEY's calling routine

;KEYA:   BSET KEY_1, $0A       ; Set KEY to A
;        RTS

KEY4:   BSET KEY_1, $04       ; Set KEY to 4
        RTS                  ; Return to GETKEY's calling routine

KEY5:   BSET KEY_1, $05       ; Set KEY to 5
        RTS                  ; Return to GETKEY's calling routine

KEY6:   BSET KEY_1, $06       ; Set KEY to 6
        RTS                  ; Return to GETKEY's calling routine

KEYB:   BSET KEY_1, $0B       ; Set KEY to B
        RTS

KEY7:   BSET KEY_1, $07       ; Set KEY to 7
        RTS                  ; Return to GETKEY's calling routine

KEY8:   BSET KEY_1, $08       ; Set KEY to 8
        RTS                  ; Return to GETKEY's calling routine

KEY9:   BSET KEY_1, $09       ; Set KEY to 9
        RTS                  ; Return to GETKEY's calling routine

KEYC:   BSET KEY_1, $0C       ; Set KEY to C
        RTS

KEY0:   BSET KEY_1, $00       ; Set KEY to 0
        RTS                  ; Return to GETKEY's calling routine

;KEYD:   BSET KEY, $0D       ; Set KEY to D
;        RTS

;KEYE:   BSET KEY, $0E       ; Set KEY to E
;        RTS                 ; Return to GETKEY's calling routine
```

```
;KEYF:   BSET KEY, $0F       ; Set KEY to F
;        RTS                 ; Return to GETKEY's calling routine
;
;

String1  FCC   "Welcome to Math "
        DC.B  $00
String2  FCC   "Flash Cards     "
        DC.B  $00
String3 FCC   "Options:        "
        DC.B  $00
String4 FCC   "Press the Key   "
        DC.B  $00
String5 FCC   "1.Addition      "
        DC.B  $00
String6 FCC   "2.Subtraction   "
        DC.B  $00
String7 FCC   "3.Multiplication"
        DC.B  $00
String8 FCC   "4.Division      "
        DC.B  $00
Strin10 FCC   "That is correct!"
        DC.B  $00
Strin11 FCC   "Incorrect :/    "
        DC.B  $00
Strin12 FCC   "Invalid Option. "
        DC.B  $00
Strin13 FCC   "Question:       "
        DC.B  $00

; Subroutine to delay the controller

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 is this
        RTS                ; anyway? 2.5 msec

delay2 LDY    #$F000        ; Long Delay routine.  Adjust as needed.
      PSHA           ; Save ACMA (do we need to?)
A3:     LDAA  #$4A          ; Makes the delay even longer! (Nested loop.)
AB:   DECA
      BNE    AB           ;
      DEY
      BNE    A3           ;
      PULA          ; Get ACMA back
      RTS

delay3  LDAA  #$0F
AA6:  LDY    #$FFFF       ; Blink Delay routine.
A6:     DEY               ;
        BNE    A6
        DECA
        BNE    AA6          ;
        RTS

sdelay: PSHY
        LDY #15000   ; Loop counter = 15000 - 2 clock cycles
A0:     LBRN A0      ; 3 clock cycles \
        DEY          ; 1 clock cycles | 8 clock cycles in loop
        LBNE A0      ; 4 clock cycles / Time = 8*<Y>/(24*10**6) + 2 =
;                    ; [8X15000 + 2]/24000000 ~= 5msec
        PULY
        rts

; End of code
; Define Power-On Reset Interrupt Vector

      ORG    $FFFE    ; $FFFE, $FFFF = Power-On Reset Int. Vector Location
      FDB   START   ; Specify instruction to execute on power up
```