```
; Author: Shuaib Shameem
; ECE 367 - Spring 2012
; A/D Converter
; Takes a 8-bit Digital conversion
; Passes it through a PISO
; Into the SPI subsystem
; Passes information back out of the SPI
; Into a SIPO
; Into an LCD which displays the voltage level
; Define symbolic constants
;
Regbase  EQU   $0000   ; Register block starts at $0000
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
TSCR2   EQU   $4D
TFLG2   EQU   $4F
;
; RAM Variables
;
        ORG   $3800
COUNT  EQU   $3800
PRINT   EQU   $3801
CONVT   EQU   $3802
BIT0    EQU   $3803
BIT1    EQU   $3804
BIT2    EQU   $3805
BIT3    EQU   $3806
BIT4    EQU   $3807
BIT5    EQU   $3808
BIT6    EQU   $3809
BIT7    EQU   $380A
BITLO   EQU   $380B
BITLW   EQU   $380C
HEX1    EQU   $380D
HEX2    EQU   $380E
DEC1    EQU   $380F
DEC2    EQU   $3810
DEC3    EQU   $3811
;
; 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
       MOVB  #$80, TSCR2     ; Enable Timer Overflow Interrupt
 ;
      CLI                     ; Turn ON Interrupts
 ;
     LDAA  #$FF           ; 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  #$02
     STAA  SPCR2
 ;
 ; Initialize Variables to $00
 ;
        BCLR  PRINT, $FF
        BCLR  CONVT, $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

 repeat  JSR   Capture           ; Capture the input from PISO Chip
        JSR   InitialVals       ; Initialize variable values

        LDX   #String3        ; Load base address of String2
        JSR   PrintString

        JSR   BinaryConv        ; Convert the input to binary and display

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

        LDX   #String4        ; Load base address of String2
        JSR   PrintString

        JSR   HexConv           ; Convert the input to hex and display

        LDX   #String5        ; Load base address of String2
        JSR   PrintString

        JSR   DecimalConv       ; Convert the input to decimal and display

        JSR   delay3
```

```
      JSR   ClearDisp        ; Clear the display
        JMP   repeat            ; Repeat the process again
;
; ===================================================================================
;
; SubRoutines
;
; 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
;
; Initialize all the variables
;
InitialVals
        LDAA  #$30          ; Load Hex $30 on Accl A
        STAA  BIT0          ; Initialize all the BIT variables with hex $30
        STAA  BIT1
        STAA  BIT2
        STAA  BIT3
        STAA  BIT4
        STAA  BIT5
        STAA  BIT6
        STAA  BIT7
        LDAA  #$00          ; Load Hex $00 on Accl A
        STAA  HEX1          ; Initialize all HEX and DEC variables to hex $00
        STAA  HEX2
        STAA  DEC1
        STAA  DEC2
        STAA  DEC3
        RTS
;
; Convert the input to a Binary value
;
BinaryConv
        LDAA  CONVT         ; Load CONVT on Accl A
        PSHA                ; Push it on the stack to save the value
        ASL   CONVT         ; Shift CONVT to the left
        BCS   Out1          ; If Carry set to 1, Branch
        JMP   R1            ; Jump to keep the BIT value to $30
Out1    MOVB  #$31, BIT0    ; Move $31 to the BIT variable to display 1
R1      LDAA  BIT0          ; Load BIT on Accl A
        JSR   Print         ; Print the value
        ASL   CONVT         ; Shift CONVT to the left
        BCS   Out2          ; If Carry set to 1, Branch
        JMP   R2            ; Jump to keep the BIT value to $30
Out2    MOVB  #$31, BIT1    ; Move $31 to the BIT variable to display 1
R2      LDAA  BIT1          ; Load BIT on Accl A
```

```
        JSR   Print         ; Print the value
        ASL   CONVT         ; Shift CONVT to the left
        BCS   Out3          ; If Carry set to 1, Branch
        JMP   R3            ; Jump to keep the BIT value to $30
Out3    MOVB  #$31, BIT2    ; Move $31 to the BIT variable to display 1
R3      LDAA  BIT2          ; Load BIT on Accl A
        JSR   Print         ; Print the value
        ASL   CONVT         ; Shift CONVT to the left
        BCS   Out4          ; If Carry set to 1, Branch
        JMP   R4            ; Jump to keep the BIT value to $30
Out4    MOVB  #$31, BIT3    ; Move $31 to the BIT variable to display 1
R4      LDAA  BIT3          ; Load BIT on Accl A
        JSR   Print         ; Print the value
        ASL   CONVT         ; Shift CONVT to the left
        BCS   Out5          ; If Carry set to 1, Branch
        JMP   R5            ; Jump to keep the BIT value to $30
Out5    MOVB  #$31, BIT4    ; Move $31 to the BIT variable to display 1
R5      LDAA  BIT4          ; Load BIT on Accl A
        JSR   Print         ; Print the value
        ASL   CONVT         ; Shift CONVT to the left
        BCS   Out6          ; If Carry set to 1, Branch
        JMP   R6            ; Jump to keep the BIT value to $30
Out6    MOVB  #$31, BIT5    ; Move $31 to the BIT variable to display 1
R6      LDAA  BIT5          ; Load BIT on Accl A
        JSR   Print         ; Print the value
        ASL   CONVT         ; Shift CONVT to the left
        BCS   Out7          ; If Carry set to 1, Branch
        JMP   R7            ; Jump to keep the BIT value to $30
Out7    MOVB  #$31, BIT6    ; Move $31 to the BIT variable to display 1
R7      LDAA  BIT6          ; Load BIT on Accl A
        JSR   Print         ; Print the value
        ASL   CONVT         ; Shift CONVT to the left
        BCS   Out8          ; If Carry set to 1, Branch
        JMP   R8            ; Jump to keep the BIT value to $30
Out8    MOVB  #$31, BIT7    ; Move $31 to the BIT variable to display 1
R8      LDAA  BIT7          ; Load BIT on Accl A
        JSR   Print         ; Print the value

        PULA                ; Pull the Stack value out to Accl A
        STAA  CONVT         ; Store Accl A value on CONVT
        RTS
;
; Convert the value to a Hex Value
;
HexConv                     ; Get the first four bits of CONVT to form a hex output

        LDAA  CONVT         ; Load CONVT on Accl A
        PSHA                ; Push it on the stack to save the value



Ova1    LDAA  HEX2          ; Load HEX on Accl A
        ASR   CONVT         ; Shift CONVT to the right
        BCS   Het1          ; If Carry set to 1, Branch
        LDAA  #$00          ; Load $00 on Accl A
        JMP   H1            ; Jump to add the value to HEX
Het1    LDAA  #$01          ; Load $01 on Accl A
        ADDA  HEX2          ; Add the Value to HEX
        STAA  HEX2          ; Store Accl A back on HEX
H1      ASR   CONVT         ; Shift CONVT to the right
        BCS   Het2          ; If Carry set to 1, Branch
        LDAA  #$00          ; Load $00 on Accl A
        JMP   H2            ; Jump to add the value to HEX
Het2    LDAA  #$02          ; Load $01 on Accl A
        ADDA  HEX2          ; Add the Value to HEX
        STAA  HEX2          ; Store Accl A back on HEX
H2      ASR   CONVT         ; Shift CONVT to the right
        BCS   Het3          ; If Carry set to 1, Branch
        LDAA  #$00          ; Load $00 on Accl A
        JMP   H3            ; Jump to add the value to HEX
Het3    LDAA  #$04          ; Load $01 on Accl A
        ADDA  HEX2          ; Add the Value to HEX
        STAA  HEX2          ; Store Accl A back on HEX
```

```
H3      ASR   CONVT         ; Shift CONVT to the right
        BCS   Het4          ; If Carry set to 1, Branch
        LDAA  #$00          ; Load $00 on Accl A
        JMP   H4            ; Jump to add the value to HEX
Het4    LDAA  #$08          ; Load $01 on Accl A
        ADDA  HEX2          ; Add the Value to HEX
        STAA  HEX2          ; Store Accl A back on HEX
H4      CMPA  #$09          ; Compare Accl A to $09
        BHI   High1         ; Branch if Accl A > $09
        LDAA  HEX2          ; Load HEX on Accl A
        ADDA  #$30          ; Add $30 to Accl A
        STAA  HEX2          ; Store Accl A on HEX
        JMP   Pri1          ; Jump to Pri1
High1   LDAA  HEX2          ; Load HEX on Accl A
        ADDA  #$37          ; Add $41 to Accl A
        STAA  HEX2          ; Store Accl A to HEX

Pri1    LDAA  HEX1          ; Same function as the above code with HEX1
        ASR   CONVT         ; to get the last 4 bits of CONVT to form another
        BCS   Het5          ; hex output
        LDAA  #$00
        JMP   H5
Het5    LDAA  #$01
        ADDA  HEX1
        STAA  HEX1
H5      ASR   CONVT
        BCS   Het6
        LDAA  #$00
        JMP   H6
Het6    LDAA  #$02
        ADDA  HEX1
        STAA  HEX1
H6      ASR   CONVT
        BCS   Het7
        LDAA  #$00
        JMP   H7
Het7    LDAA  #$04
        ADDA  HEX1
        STAA  HEX1
H7      ASR   CONVT
        BCS   Het8
        LDAA  #$00
        JMP   H8
Het8    LDAA  #$08
        ADDA  HEX1
        STAA  HEX1
H8      CMPA  #$09
        BHI   High2
        LDAA  HEX1
        ADDA  #$30
        STAA  HEX1
        JMP   Pri2
High2   LDAA  HEX1
        ADDA  #$37
        STAA  HEX1

Pri2    LDAA  HEX1          ; Load HEX1 on Accl A
        JSR   Print         ; Print HEX1
        LDAA  HEX2          ; Load HEX2 on Accl A
        JSR   Print         ; Print HEX2

        PULA                ; Pull the Stack value out to Accl A
        STAA  CONVT         ; Store Accl A value on CONVT
        RTS
;
; Convert the value to a 3 digit decimal Value
;
DecimalConv
        LDAA CONVT          ; Load CONVT on Accl A

Check2  CMPA #100           ; Compare D to #100
        BLO  Check3         ; Branch if Less than 100
```

```
        INC  DEC3           ; Increment DEC3
        SUBA #100           ; Subtract D by 100
        JMP  Check2         ; Jump to Check2
Check3  CMPA  #10           ; Compare D to #10
        BLO  PutIn          ; Branch if less than 10
        INC  DEC2           ; Increment DEC2
        SUBA #10            ; Subtract D by 10
        JMP  Check3         ; Jump to Check3
PutIn   STAA DEC1           ; Store D on UNUMBA1
        LDAA DEC3           ; Load DEC3 on A
        JSR  AconvP         ; Convert to Ascii and Print
        LDAA DEC2           ; Load DEC2 on A
        JSR  AconvP         ; Convert to Ascii and Print
        LDAA DEC1           ; Load DEC1 on A
        JSR  AconvP         ; Convert to Ascii and Print
        RTS
;
; Capture the Input from PISO
;
Capture LDAA #$01
        STAA PortT
        LDAB  #$FF                ; Load $FF on Accl B
        STAB  SPDR                ; Store B on SPDR
CKFLG2  BRCLR SPSR,  $80, CKFLG2  ; Wait for SPI Flag
      LDAA  SPDR                ; Load SPDR value on Accl A
      STAA  CONVT               ; Store A on CONVT
    LDAA #$00
    STAA PortT
    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
;
; Strings
;
String1  FCC   "The AD Convertor"
        DC.B  $00
String2  FCC   "By: Shuaib      "
        DC.B  $00
String3  FCC   "Bin: "
        DC.B  $00
String4  FCC   "Hex: "
        DC.B  $00
String5  FCC   " Dec: "
        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
```