LAMPIRAN B
Listing Program

Listing Program *Hardware Secondary Controller* B-1

```
;#########################################################
;##                                                     ##
;##  Project Name : C.R.S [SMS] - Secondary Controller  ##
;##        Author : Jonathan Chandra a.k.a JoC          ##
;##   Create Date : 15:21:03 06/07/2011                 ##
;##                                                     ##
;#########################################################

;-------------------- Program embedded variables --------------------------------
; # Authorized phone number
; If you enter a number please make sure there is 0 value after that. And if there
is still empty slots, fill them with char ' '
NUM1 equ       '6'
NUM2 equ       '2'
NUM3 equ       '8'
NUM4 equ       '1'
NUM5 equ       '3'
NUM6 equ       '2'
NUM7 equ       '2'
NUM8 equ       '2'
NUM9 equ       '8'
NUM10 equ      '8'
NUM11 equ      '7'
NUM12 equ      '7'
NUM13 equ      '9'
NUM14 equ      0

AUTH_NUM_LENGTH equ 13 ; <- Authorized number length

; # HEARTBEAT timeout
STARTUP_TIMEOUT equ 15 ; in minutes
NORMAL_TIMEOUT equ 5 ; in minutes (and this also applied to wait until computer
completely off)

;---------------------------------- Constant ----------------------------------
NUMBER_ADDRESS_OFFSET equ 30h ; 0x030
MESSAGE_ADDRESS_OFFSET equ 3Eh ; 0x30 + 0x0E
MESSAGE_MAX equ 50 ; 50-chars ASCII
NUMBER_MAX equ 14 ; 14-chars ASCII Numbers

; ------------------------- Ports and Pins aliases -----------------------------
; ### M.P.D Interface Lines (Data_PORT, Status_Bits and Control_Bits) ###
COMM_Lines equ P2
COMM_Port equ P2

MPD_Clock equ P1.4 ; Output pin, clock signaling to M.P.D Translator
MPD_Busy_flag equ P1.5 ; Input pin, Output signal from M.P.D Translator
MPD_Ready equ P1.6 ; Input pin, Output signal from M.P.D Translator

; ### LCD Connections ###
LCD_Lines equ P0
LCD_Lines_bit7 equ P0.7
RS equ P3.5
RW equ P3.6
EN equ P3.7

; ### 2nd Controller Outputs & Inputs ###
RELAY equ P1.0 ; - (OUT) Control Computer PWR_SW [Active-Low]

; 74LS373 Chips
OE equ P1.1 ; - (OUT) Control Computer COMM. Lines + Control Computer CONTROL_BITS
[Active-Low]

nPWR_Sense equ P1.3 ; (IN) Power sensor status (HIGH=Computer_OFF,
LOW=Computer_ON)

Heartbeat_Sense equ P3.2 ; (Reserved) External Interrupt Program

; ------------------- Variables/RAM_Addrs/Registers aliases ---------------------
Number_Index equ 70h
```

```
Number_Length equ 71h

Message_Index equ 72h
Message_Length equ 73h

vTimer_Hours equ 74h
vTimer_Minutes equ 75h
vTimer_Seconds equ 76h
vTimer_Milliseconds equ 77h
vTimer_Reserved equ 78h ; Used by Timer1 to produce 1 ms
vTimer_Reserved2 equ 79h ; Used by Timer1 to produce 1 second

HEARTBEAT_COUNT equ R4
FLAG equ R5
; XXXX.XXXa
; a = Not_Responding flag. This bit is set by timeout program (if heartbeat from
computer lost and pass the timeout interval).

TMP_A equ R6
TMP_B equ R7

;===============================================================================
;-------------------------- Reset Interrupt Vector Address ----------------------
org 0000h
jmp program_init

;------------------------- External 0 Interrupt Vector Address ------------------
org 0003h
jmp external0_interrupt

;------------------------- Timer 1 Interrupt Vector Address ---------------------
org 001Bh
jmp timer1_interrupt

;------------------------ Program Vector Address --------------------------------
org 0030h
program_init:

   ; > Setup ports / pins
      setb MPD_Ready ; set as Input
      setb nPWR_Sense ; set as Input
      setb MPD_Clock ; default HIGH
      setb MPD_Busy_flag ; set as Input

      setb RELAY ; default = disabled
      setb OE ; default = disabled

      ;[IMPORTANT for LCD Initalization]
      clr EN ; LCD enable false
      clr RW ; LCD mode write
      setb RS ; LCD comm for data
      mov LCD_Lines, #00h

   ; > Initialize variables
      mov HEARTBEAT_COUNT, #0
      mov FLAG, #0
      mov Number_Index, #0
      mov Number_Length, #0
      mov Message_Index, #0
      mov Message_Length, #0

   ; > Setup and initialize timer
      ; Set Timer0 to 16-bit counter
      ; Set Timer1 to 8-bit autoload counter
      mov TMOD, #21h
      ; Setup Timer1 values
      mov TH1, #06h ; reset to 0x06 | 250us = 256us - x | x = 6
      mov TL1, #05h
      clr TR1 ; Timer1 off.
```

```
   ; > LCD Initialization
      call delay_50ms
      call LCD_init
      call LCD_Clear

   ; > Initialize interrupt
      setb IT0 ; (falling-edge) external interrupt type
      setb ET1 ; Timer1 interrupt enabled
      setb EA ; Global interrupt enabled
      clr EX0 ; Disable heartbeat sensor

   ; [Startup] > Check for M.P.D Translator ready state
      call display_wait_mpd
      call display_please_wait
      jb MPD_Ready, $ ; wait until Ready pin goes LOW.

   ; [Startup] > Start on command mode / step no 2.
      ; delay 1 second
      call delay_1sec
      ; jump
      jmp step_2

program_begin:

   ; 1 > Check for "Computer-Power-Sensor"
   ;   if the sensor return value is HIGH (which mean the computer is OFF) then
proceed to step 2
   ;   if the sensor return value is LOW (which mean the computer is ON) then go
to step 3, skip step no 2 (automatically set priority to computer)
   ;   if the sensor return value is LOW and heartbeat_failure flag is set, then
inform the user what happened, and goto step no 2. (waiting next instruction)

      power_check:

         jb nPWR_Sense, state_off
         state_on:

            ; > Check for NOT-RESPONDING flag
            ;  if NOT-RESPONDING is true then goto step no 2. Leave the program
flow.
            ;  if responding then <error> [why the program goes here? by design,
it shouldn't goes here!]

            mov A, FLAG
            jb Acc.0, not_responding
               ; Responding - normal
               ; Boot time? or Error?

               call LCD_Clear
               call display_error
               call smsg_error
               jmp end_state

            not_responding:

               call LCD_Clear
               call display_not_responding
               call smsg_not_responding
               jmp end_state

         state_off:
            ; Nothing to do, proceed to step no 2

         end_state:

         ; delay 1 second
         call delay_1sec

   ; 2 > Set priority to 2nd_Controller and get new message from M.P.D Translator.
      step_2:
```

```
;    - Set priority to 2nd Controller
       setb OE ; disabled

;     - Tell author, 2nd controller is ready
       call LCD_Clear
       call display_please_wait
       call smsg_ready

   check_message:

;    - Display text
       call LCD_Clear
       call display_checking_for_message

;    - Get message from M.P.D Translator
       ; Save the data and number into RAM.
       ; After save, delete the new message

       ;[Get M.P.D Translator Status]
          ; > Send instruction number
          mov COMM_Lines, #09h
          call func_MPD_Clock

          ; > Set COMM Port into input mode
          mov COMM_Port, #0FFh

          ; > Execute instruction
          call func_MPD_Clock

          ; > Wait for busy flag transition (falling-edge)
          call delay_1ms ; (delay 1 ms)
          jb MPD_Busy_Flag, $

          ; > Get returned data
          mov TMP_A, COMM_Port ; save into TMP

          ; > Confirm, the data has been received
          call func_MPD_Clock

          ; > Exit from this instruction
          call func_MPD_Clock

          ; Process_No Done
          mov A, #' '
          call LCD_Write_Data
          mov A, #'#'
          call LCD_Write_Data

       cjne TMP_A, #01h, jhgo0

       ; *jump helper begin*
       jmp jh0
          jhgo0:

          ; Keep connection alive
          call func_MPD_MP_Attention

          ; Process_No Done
          mov A, #'%'
          call LCD_WRITE_Data

          jmp no_message
       jh0:
       ; *jump helper end*

          ; Process_No
          call LCD_CLear

          ;[Clear Storages]
```

```
                  call func_MPD_Clear_Storages

                  ; > Reset RAM pointers
                  mov NUMBER_LENGTH, #0
                  mov NUMBER_INDEX, #0
                  mov MESSAGE_LENGTH, #0
                  mov MESSAGE_INDEX, #0

               ; Process_No Done
               mov A, #'1'
               call LCD_Write_Data

               ;[Decode Message]
                  call func_MPD_Decode

               ; Process_No Done
               mov A, #'2'
               call LCD_Write_Data

               ;[Delete last SMS]
                  call func_MPD_Delete_last_msg

               ; Process_No Done
               mov A, #'3'
               call LCD_Write_Data

               ;[Read Message]

                  ; > Send instruction number
                  mov COMM_Lines, #07h
                  call func_MPD_Clock

                  ; > Set COMM Port into INPUT mode
                  mov COMM_Lines, #0FFh

                  ; > Execute instruction
                  call func_MPD_Clock

                  ; ! No wait for busy, because this process included into M.P.D
program directly

                  ; > Get returned data into pattern and save them.

                     ; 1. Number length
                     mov NUMBER_LENGTH, COMM_Lines
                     call func_MPD_Clock

                     ; 2. Number data, digit by digit
                     mov TMP_A, NUMBER_LENGTH
                     ;(Loop-begin)

                        save_number:

                           mov B, COMM_Lines
                           call func_MPD_Clock

                           ; indirect-addressing
                           mov A, #NUMBER_ADDRESS_OFFSET
                           add A, NUMBER_INDEX
                           mov R0, A

                           ; save on RAM
                           mov @R0, B

                           ; next digit address
                           inc NUMBER_INDEX

                        djnz TMP_A, save_number

                     ;(Loop-end)
```

```
      ; Process_No Done
      mov A, #'4'
      call LCD_Write_Data

      ; 3. Message length
      mov MESSAGE_LENGTH, COMM_Lines
      call func_MPD_Clock

      ; 4. Message data, char by char
      mov TMP_A, MESSAGE_LENGTH
      ;(Loop-begin)

         save_message:

            mov B, COMM_Lines
            call func_MPD_Clock

            ; indirect-addressing
            mov A, #MESSAGE_ADDRESS_OFFSET
            add A, MESSAGE_INDEX
            mov R0, A

            ; save on RAM
            mov @R0, B

            ; next digit address
            inc MESSAGE_INDEX

         djnz TMP_A, save_message

      ;(Loop-end)

      ; Process_No Done
      mov A, #'5'
      call LCD_Write_Data

   ; > Exit instruction
   call func_MPD_Clock

; Process_No Done
mov A, #'6'
call LCD_Write_Data

call delay_1sec

; > Check for authorized number!

   call ResetDigitIndex

   call ReadDigitAndNext
   cjne A, #NUM1, wrong_number1
   call ReadDigitAndNext
   cjne A, #NUM2, wrong_number2
   call ReadDigitAndNext
   cjne A, #NUM3, wrong_number3
   call ReadDigitAndNext
   cjne A, #NUM4, wrong_number4
   call ReadDigitAndNext
   cjne A, #NUM5, wrong_number5
   call ReadDigitAndNext
   cjne A, #NUM6, wrong_number6
   call ReadDigitAndNext
   cjne A, #NUM7, wrong_number7
   call ReadDigitAndNext
   cjne A, #NUM8, wrong_number8
   call ReadDigitAndNext
   cjne A, #NUM9, wrong_number9
   call ReadDigitAndNext
   cjne A, #NUM10, wrong_number10
```

```
                call ReadDigitAndNext
                cjne A, #NUM11, wrong_number11
                call ReadDigitAndNext
                cjne A, #NUM12, wrong_number12
                call ReadDigitAndNext
                cjne A, #NUM13, wrong_number13
                call ReadDigitAndNext
                cjne A, #NUM14, wrong_number14
                   ; [PROCESS]

                   jmp authorized_number

                wrong_number14:
                wrong_number13:
                wrong_number12:
                wrong_number11:
                wrong_number10:
                wrong_number9:
                wrong_number8:
                wrong_number7:
                wrong_number6:
                wrong_number5:
                wrong_number4:
                wrong_number3:
                wrong_number2:
                wrong_number1:

                   ; INTRUDER NUMBER, IGNORE IT

                   call LCD_Clear
                   call display_intruder_number
                   jmp no_message

                authorized_number:

             ; [Display]
                ; Show on the LCD, the command name

                call LCD_Clear

                call display_command_is
                call LCD_goto_2nd

                call ResetCharIndex
                mov TMP_A, MESSAGE_LENGTH
                next_char:
                   call ReadCharAndNext
                   call LCD_Write_Data
                djnz TMP_A, next_char

                call delay_1sec
                call LCD_Clear

             ;     - If authorized then save the message into RAM
             ;             - Call RAM addresses for values then compare into switches

             ;     [Supported-Commands] | Instruction switches

             ;     `Power on`
             ;             Turn on computer by pushing PWR_BUTTON in 1 sec, then
release it off.

                ; call Refresh position (reset address)
                call ResetCharIndex

                ; call Get data, and next address
                call ReadCharAndNext
                cjne A, #'P', no_power_on1
                call ReadCharAndNext
                cjne A, #'o', no_power_on2
```

```
                call ReadCharAndNext
                cjne A, #'w', no_power_on3
                call ReadCharAndNext
                cjne A, #'e', no_power_on4
                call ReadCharAndNext
                cjne A, #'r', no_power_on5
                call ReadCharAndNext
                cjne A, #' ', no_power_on6
                call ReadCharAndNext
                cjne A, #'o', no_power_on7
                call ReadCharAndNext
                cjne A, #'n', no_power_on_8
                call ReadCharAndNext
                cjne A, #0, no_power_on_9
                   ; [PROCESS]

                      ; check, is it already on? If yes, then skip pushing the
switch
                      jnb nPWR_SENSE, already_powered_on

                         call display_acknowledged

                         clr RELAY ; activate
                         call delay_1sec
                         setb RELAY ; deactivate

                         call smsg_acknowledged

                      already_powered_on:

                      jmp priority_computer

                   jmp instruction_done
                no_power_on_9:
                no_power_on_8:
                no_power_on7:
                no_power_on6:
                no_power_on5:
                no_power_on4:
                no_power_on3:
                no_power_on2:
                no_power_on1:

        ;     `Soft off`
        ;            Turn off computer by pushing PWR_BUTTON less than 4 sec. ( 1
sec )

                ; call Refresh position (reset address)
                call ResetCharIndex

                ; call Get data, and next address
                call ReadCharAndNext
                cjne A, #'S', no_soft_off1
                call ReadCharAndNext
                cjne A, #'o', no_soft_off2
                call ReadCharAndNext
                cjne A, #'f', no_soft_off3
                call ReadCharAndNext
                cjne A, #'t', no_soft_off4
                call ReadCharAndNext
                cjne A, #' ', no_soft_off5
                call ReadCharAndNext
                cjne A, #'o', no_soft_off6
                call ReadCharAndNext
                cjne A, #'f', no_soft_off7
                call ReadCharAndNext
                cjne A, #'f', no_soft_off8
                call ReadCharAndNext
                cjne A, #0, no_soft_off8
                   ; [PROCESS]
```

```
                        ; skip pushing the switch, if it's already off
                        jb nPWR_Sense, already_off_1

                           call display_acknowledged

                           clr RELAY ; activate
                           call delay_1sec
                           setb RELAY ; deactivate

                           call smsg_acknowledged

                        already_off_1:

                     jmp instruction_done
                  no_soft_off9:
                  no_soft_off8:
                  no_soft_off7:
                  no_soft_off6:
                  no_soft_off5:
                  no_soft_off4:
                  no_soft_off3:
                  no_soft_off2:
                  no_soft_off1:

            ;     `Hard off`
            ;            Turn off computer by pushing PWR_BUTTON more than 4 sec. ( 5
sec )

                  ; call Refresh position (reset address)
                  call ResetCharIndex

                  ; call Get data, and next address
                  call ReadCharAndNext
                  cjne A, #'H', no_hard_off1
                  call ReadCharAndNext
                  cjne A, #'a', no_hard_off2
                  call ReadCharAndNext
                  cjne A, #'r', no_hard_off3
                  call ReadCharAndNext
                  cjne A, #'d', no_hard_off4
                  call ReadCharAndNext
                  cjne A, #' ', no_hard_off5
                  call ReadCharAndNext
                  cjne A, #'o', no_hard_off6
                  call ReadCharAndNext
                  cjne A, #'f', no_hard_off7
                  call ReadCharAndNext
                  cjne A, #'f', no_hard_off8
                  call ReadCharAndNext
                  cjne A, #0, no_hard_off9
                     ; [PROCESS]

                        ; skip pushing the switch, if it's already off
                        jb nPWR_Sense, already_off_2

                           call display_acknowledged

                           clr RELAY ; activate
                           call delay_1sec
                           call delay_1sec
                           call delay_1sec
                           call delay_1sec
                           call delay_1sec
                           setb RELAY ; deactivate

                           call smsg_acknowledged

                        already_off_2:
```

```
                jmp instruction_done
            no_hard_off9:
            no_hard_off8:
            no_hard_off7:
            no_hard_off6:
            no_hard_off5:
            no_hard_off4:
            no_hard_off3:
            no_hard_off2:
            no_hard_off1:

        ;     `Attention`
        ;            Get current status from computer (ON/OFF/NOT-RESPONDING
STATE?)

            ; call Refresh position (reset address)
            call ResetCharIndex

            ; call Get data, and next address
            call ReadCharAndNext
            cjne A, #'A', no_attention1
            call ReadCharAndNext
            cjne A, #'t', no_attention2
            call ReadCharAndNext
            cjne A, #'t', no_attention3
            call ReadCharAndNext
            cjne A, #'e', no_attention4
            call ReadCharAndNext
            cjne A, #'n', no_attention5
            call ReadCharAndNext
            cjne A, #'t', no_attention6
            call ReadCharAndNext
            cjne A, #'i', no_attention7
            call ReadCharAndNext
            cjne A, #'o', no_attention8
            call ReadCharAndNext
            cjne A, #'n', no_attention9
            call ReadCharAndNext
            cjne A, #0, no_attention10
               ; [PROCESS]

                  call display_acknowledged
                  call smsg_attention

               jmp instruction_done
            no_attention10:
            no_attention9:
            no_attention8:
            no_attention7:
            no_attention6:
            no_attention5:
            no_attention4:
            no_attention3:
            no_attention2:
            no_attention1:

        ;     `Priority computer`
        ;            Set current M.P.D Translator controls to computer.

            ; call Refresh position (reset address)
            call ResetCharIndex

            ; call Get data, and next address
            call ReadCharAndNext
            cjne A, #'P', no_priority_computer1
            call ReadCharAndNext
            cjne A, #'r', no_priority_computer2
            call ReadCharAndNext
            cjne A, #'i', no_priority_computer3
            call ReadCharAndNext
```

```
      cjne A, #'o', no_priority_computer4
      call ReadCharAndNext
      cjne A, #'r', no_priority_computer5
      call ReadCharAndNext
      cjne A, #'i', no_priority_computer6
      call ReadCharAndNext
      cjne A, #'t', no_priority_computer7
      call ReadCharAndNext
      cjne A, #'y', no_priority_computer8
      call ReadCharAndNext
      cjne A, #' ', no_priority_computer9
      call ReadCharAndNext
      cjne A, #'c', no_priority_computer10
      call ReadCharAndNext
      cjne A, #'o', no_priority_computer11
      call ReadCharAndNext
      cjne A, #'m', no_priority_computer12
      call ReadCharAndNext
      cjne A, #'p', no_priority_computer13
      call ReadCharAndNext
      cjne A, #'u', no_priority_computer14
      call ReadCharAndNext
      cjne A, #'t', no_priority_computer15
      call ReadCharAndNext
      cjne A, #'e', no_priority_computer16
      call ReadCharAndNext
      cjne A, #'r', no_priority_computer17
      call ReadCharAndNext
      cjne A, #0, no_priority_computer18
         ; [PROCESS]

            ; check for power presence

            jb nPWR_Sense, cancel_priority_computer

               call display_acknowledged

               call smsg_acknowledged

               jmp priority_computer

            cancel_priority_computer:

            ; Cannot set priority to computer,
            ; the computer itself isn't turned on
            call smsg_computer_off

         jmp instruction_done
      no_priority_computer18:
      no_priority_computer17:
      no_priority_computer16:
      no_priority_computer15:
      no_priority_computer14:
      no_priority_computer13:
      no_priority_computer12:
      no_priority_computer11:
      no_priority_computer10:
      no_priority_computer9:
      no_priority_computer8:
      no_priority_computer7:
      no_priority_computer6:
      no_priority_computer5:
      no_priority_computer4:
      no_priority_computer3:
      no_priority_computer2:
      no_priority_computer1:

   instruction_done:

no_message:
```

```
         ; Scan new message every 5 seconds
         ; delay 5 sec.
         call delay_1sec
         call delay_1sec
         call delay_1sec
         call delay_1sec
         call delay_1sec

      jmp check_message ; check for new message

   ; 3 > Set priority/controls to computer

      priority_computer:

   ;    - Send message
      call smsg_disabled

   ;    - Set COMM. Lines/Port and Control bits to HIGH (Mode: As Input)
      mov COMM_Lines, #0FFh

   ;    - Display text
      call LCD_Clear
      call display_controller_disabled

   ; 4 > Turn on computer heartbeat sensor, and activate timeout.

   ;    - Clear heartbeat_failure / Not_Responding flag
      mov A, R5
      clr Acc.0
      mov R5, A

   ;    - Set interrupt for External0 (Heartbeat sensor on)
      setb EX0

   ;    - Activate timeout
      call Timer_Restart

   ;    - Wait for a heartbeat signal from computer (computer-startup-wait)
   ;    (n Minutes)
      mov HEARTBEAT_COUNT, #0 ; reset
      wait_for_startup:
         mov A, HEARTBEAT_COUNT
         cjne A, #3, wait_for_startup2
            jmp startup_done
         wait_for_startup2:
         mov A, vTimer_Minutes
      cjne A, #STARTUP_TIMEOUT, wait_for_startup
      ; startup timeout!! - something wrong (computer problem)
         jmp heartbeat_failure
      startup_done:

   ;    - Turn on OUTPUT_ENABLE on 74LS373 chip, make M.P.D connection with the
computer
      clr OE

      call Timer_Restart
      ; (n Minutes) timeout
      wait_for_timeout:
         mov A, vTimer_Minutes
      cjne A, #NORMAL_TIMEOUT, wait_for_timeout

      heartbeat_failure:

      call Timer_Stop ; stop timeout timer
      clr EX0 ; timeout! - disable sensor
      setb OE ; disable comm.lines to computer.

   ; * > If heartbeat sensor detect something wrong (beat loss) it goes after
here, then we need to restart to the beginning of this program.
```

```
   ;    - Set heartbeat_failure / Not_Responding flag
      mov A, FLAG
      setb Acc.0
      mov FLAG, A

      jmp program_begin

program_end:

;================================================================================
;                      PROGRAM INTERRUPT HANDLER
;================================================================================
external0_interrupt:
   push Acc
   push PSW

   ; External 0 Interrupt Handler
   ; active from HIGH to LOW Transition / Falling edge
   ; a.k.a Heartbeat sensor

   call Timer_Restart ; (restart timeout)

   inc HEARTBEAT_COUNT ; +1

   pop PSW
   pop Acc
reti

timer1_interrupt:
   push Acc
   push PSW

   ; Timer1 Interrupt Handler

   ; x 250 us
   inc vTimer_Reserved

   ; x 1 ms
   mov A, vTimer_Reserved
   cjne A, #4, no_1_ms
      inc vTimer_Milliseconds
      mov vTimer_Reserved, #0 ; reset per ms
   no_1_ms:

   ; x 250 ms
   mov A, vTimer_Milliseconds
   cjne A, #250, no_1p4_sec
      inc vTimer_Reserved2
      mov vTimer_Milliseconds, #0 ; reset per 250ms
   no_1p4_sec:

   ;---------------------------------------------------------------

   ; x 1 s
   mov A, vTimer_Reserved2
   cjne A, #4, no_1_sec
      inc vTimer_Seconds
      mov vTimer_Reserved2, #0 ; reset per second
   no_1_sec:

   ; x 1 M
   mov A, vTimer_Seconds
   cjne A, #60, no_1_min
      inc vTimer_Minutes
      mov vTimer_Seconds, #0 ; reset per minute
   no_1_min:

   ; x 1 H
   mov A, vTimer_Minutes
   cjne A, #60, no_1_hour
```

```
      inc vTimer_Hours
      mov vTimer_Minutes, #0 ; reset per hour
   no_1_hour:

   ;-------------------------------------------------------------

   pop PSW
   pop Acc
reti

;===============================================================================
;                        PROGRAM GENERAL FUNCTIONS
;===============================================================================

func_MPD_Clock:
   ; Give M.P.D Translator a clock pulse

   setb MPD_Clock

   ; delay
   call delay_1ms

   clr MPD_Clock

   ; delay
   call delay_1ms

   setb MPD_Clock

   ; delay
   call delay_1ms

ret

; ### M.P.D Translator software functions ###

func_MPD_input_number:
   ; Input: A

   ; > Set instruction number
   mov COMM_Port, #03h
   call func_MPD_Clock

   ; > Set parameter, digit
   mov COMM_Port, A
   call func_MPD_Clock

   ; > Start execution
   call func_MPD_Clock

   ; > Wait for busy flag
   call delay_1ms ; (delay 1 ms)
   jb MPD_Busy_Flag, $

   ; > Exit from instruction
   call func_MPD_Clock
ret

func_MPD_input_char:
   ; Input: A

   ; > Set instruction number
   mov COMM_Port, #02h
   call func_MPD_Clock

   ; > Set parameter, char
   mov COMM_Port, A
   call func_MPD_Clock

   ; > Start execution
```

```
   call func_MPD_Clock

   ; > Wait for busy flag
   call delay_1ms ; (delay 1 ms)
   jb MPD_Busy_Flag, $

   ; > Exit from instruction
   call func_MPD_Clock
ret

func_MPD_Clear_Storages:
   ; [Clear Storages]

   ; > Set instruction number
   mov COMM_Port, #01h
   call func_MPD_Clock

   ; > Start execution
   call func_MPD_Clock

   ; > Wait for busy flag
   call delay_1ms ; (delay 1 ms)
   jb MPD_Busy_Flag, $

   ; > Exit from instruction
   call func_MPD_Clock
ret

func_MPD_MP_Attention:
   ; [Get mobile-phone attention]

   ; > Set instruction number
   mov COMM_Port, #0Bh
   call func_MPD_Clock

   ; > Start execution
   call func_MPD_Clock

   ; > Wait for busy flag
   call delay_1ms ; (delay 1 ms)
   jb MPD_Busy_Flag, $

   ; > Exit from instruction
   call func_MPD_Clock
ret

func_MPD_encode:
   ; [Encode Message] -- 7-bit

   ; > Set instruction number
   mov COMM_Port, #04h
   call func_MPD_Clock

   ; > Set parameter (0x01)
   mov COMM_Port, #01h
   call func_MPD_Clock

   ; > Start execution
   call func_MPD_Clock

   ; > Wait for busy flag
   call delay_1ms ; (delay 1 ms)
   jb MPD_Busy_Flag, $

   ; > Exit from instruction
   call func_MPD_Clock
ret

func_MPD_decode:
   ; [Get + Decode Message from phone]
```

```
   ; > Send instruction number
   mov COMM_Lines, #05h
   call func_MPD_Clock

   ; > Execute instruction
   call func_MPD_Clock

   ; > Wait for busy flag
   call delay_1ms ; (delay 1 ms)
   jb MPD_Busy_Flag, $

   ; > Exit from this instruction
   call func_MPD_Clock
ret

func_MPD_send_message:
   ; [Send Message]

   ; > Set instruction number
   mov COMM_Port, #06h
   call func_MPD_Clock

   ; > Start execution
   call func_MPD_Clock

   ; > Wait for busy flag
   call delay_1ms ; (delay 1 ms)
   jb MPD_Busy_Flag, $

   ; > Exit from instruction
   call func_MPD_Clock
ret

func_MPD_delete_last_msg:
   ; [Delete Last Message]

   ; > Send instruction number
   mov COMM_Lines, #08h
   call func_MPD_Clock

   ; > Execute instruction
   call func_MPD_Clock

   ; > Wait for busy flag
   call delay_1ms ; (delay 1 ms)
   jb MPD_Busy_Flag, $

   ; > Exit from this instruction
   call func_MPD_Clock
ret

; ### M.P.D Translator software functions END ###

set_number:
   mov TMP_A, #AUTH_NUM_LENGTH

   mov A, #NUM1
   call func_MPD_input_number
   djnz TMP_A, next_num1
   jmp set_number_done

   next_num1:
   mov A, #NUM2
   call func_MPD_input_number
   djnz TMP_A, next_num2
   jmp set_number_done

   next_num2:
   mov A, #NUM3
```

```
call func_MPD_input_number
djnz TMP_A, next_num3
jmp set_number_done

next_num3:
mov A, #NUM4
call func_MPD_input_number
djnz TMP_A, next_num4
jmp set_number_done

next_num4:
mov A, #NUM5
call func_MPD_input_number
djnz TMP_A, next_num5
jmp set_number_done

next_num5:
mov A, #NUM6
call func_MPD_input_number
djnz TMP_A, next_num6
jmp set_number_done

next_num6:
mov A, #NUM7
call func_MPD_input_number
djnz TMP_A, next_num7
jmp set_number_done

next_num7:
mov A, #NUM8
call func_MPD_input_number
djnz TMP_A, next_num8
jmp set_number_done

next_num8:
mov A, #NUM9
call func_MPD_input_number
djnz TMP_A, next_num9
jmp set_number_done

next_num9:
mov A, #NUM10
call func_MPD_input_number
djnz TMP_A, next_num10
jmp set_number_done

next_num10:
mov A, #NUM11
call func_MPD_input_number
djnz TMP_A, next_num11
jmp set_number_done

next_num11:
mov A, #NUM12
call func_MPD_input_number
djnz TMP_A, next_num12
jmp set_number_done

next_num12:
mov A, #NUM13
call func_MPD_input_number
djnz TMP_A, next_num13
jmp set_number_done

next_num13:
mov A, #NUM14
call func_MPD_input_number
djnz TMP_A, set_number_done

set_number_done:
```

```
ret

smsg_ready:
   ; "Sec. Controls - Ready"

   ; [Clear Storages]
      call func_MPD_Clear_Storages

   ; [Input Number]
      call set_number

   ; [Input Message]
      mov A, #'S'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'c'
      call func_MPD_input_char
      mov A, #'.'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'C'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'n'
      call func_MPD_input_char
      mov A, #'t'
      call func_MPD_input_char
      mov A, #'r'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'l'
      call func_MPD_input_char
      mov A, #'s'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'-'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'R'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'a'
      call func_MPD_input_char
      mov A, #'d'
      call func_MPD_input_char
      mov A, #'y'
      call func_MPD_input_char

   ; [Encode Message]
      call func_MPD_encode

   ; [Send Message]
      call func_MPD_Send_message

ret

smsg_disabled:
   ; "Sec. Controls - DISABLED"

   ; [Clear Storages]
      call func_MPD_Clear_Storages
```

```
   ; [Input Number]
      call set_number

   ; [Input Message]
      mov A, #'S'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'c'
      call func_MPD_input_char
      mov A, #'.'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'C'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'n'
      call func_MPD_input_char
      mov A, #'t'
      call func_MPD_input_char
      mov A, #'r'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'l'
      call func_MPD_input_char
      mov A, #'s'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'-'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'D'
      call func_MPD_input_char
      mov A, #'I'
      call func_MPD_input_char
      mov A, #'S'
      call func_MPD_input_char
      mov A, #'A'
      call func_MPD_input_char
      mov A, #'B'
      call func_MPD_input_char
      mov A, #'L'
      call func_MPD_input_char
      mov A, #'E'
      call func_MPD_input_char
      mov A, #'D'
      call func_MPD_input_char


   ; [Encode Message]
      call func_MPD_encode

   ; [Send Message]
      call func_MPD_Send_message
ret

smsg_attention:
   ; "Sec-Ctrl:OK, Comp:[ON/OFF]"

   ; [Clear Storages]
      call func_MPD_Clear_Storages

   ; [Input Number]
      call set_number
```

```
   ; [Input Message]

      mov A, #'S'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'c'
      call func_MPD_input_char
      mov A, #'-'
      call func_MPD_input_char
      mov A, #'C'
      call func_MPD_input_char
      mov A, #'t'
      call func_MPD_input_char
      mov A, #'r'
      call func_MPD_input_char
      mov A, #'l'
      call func_MPD_input_char
      mov A, #':'
      call func_MPD_input_char
      mov A, #'O'
      call func_MPD_input_char
      mov A, #'K'
      call func_MPD_input_char
      mov A, #','
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'C'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'m'
      call func_MPD_input_char
      mov A, #'p'
      call func_MPD_input_char
      mov A, #':'
      call func_MPD_input_char

      jb nPWR_Sense, status_off
         ; Status ON
         mov A, #'O'
         call func_MPD_input_char
         mov A, #'N'
         call func_MPD_input_char

         jmp status_end
      status_off:
         ; Status OFF
         mov A, #'O'
         call func_MPD_input_char
         mov A, #'F'
         call func_MPD_input_char
         mov A, #'F'
         call func_MPD_input_char

      status_end:

   ; [Encode Message]
      call func_MPD_encode

   ; [Send Message]
      call func_MPD_Send_message
ret

smsg_error:
   ; "Sec. Controls - ERROR"

   ; [Clear Storages]
      call func_MPD_Clear_Storages
```

```
   ; [Input Number]
      call set_number

   ; [Input Message]

      mov A, #'S'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'c'
      call func_MPD_input_char
      mov A, #'.'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'C'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'n'
      call func_MPD_input_char
      mov A, #'t'
      call func_MPD_input_char
      mov A, #'r'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'l'
      call func_MPD_input_char
      mov A, #'s'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'-'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'E'
      call func_MPD_input_char
      mov A, #'R'
      call func_MPD_input_char
      mov A, #'R'
      call func_MPD_input_char
      mov A, #'O'
      call func_MPD_input_char
      mov A, #'R'
      call func_MPD_input_char

   ; [Encode Message]
      call func_MPD_encode

   ; [Send Message]
      call func_MPD_Send_message
ret

smsg_acknowledged:
   ; "Sec - CMD Acknowledged"

   ; [Clear Storages]
      call func_MPD_Clear_Storages

   ; [Input Number]
      call set_number

   ; [Input Message]

      mov A, #'S'
      call func_MPD_input_char
      mov A, #'e'
```

```
      call func_MPD_input_char
      mov A, #'c'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'-'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'C'
      call func_MPD_input_char
      mov A, #'M'
      call func_MPD_input_char
      mov A, #'D'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'A'
      call func_MPD_input_char
      mov A, #'c'
      call func_MPD_input_char
      mov A, #'k'
      call func_MPD_input_char
      mov A, #'n'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'w'
      call func_MPD_input_char
      mov A, #'l'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'d'
      call func_MPD_input_char
      mov A, #'g'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'d'
      call func_MPD_input_char

   ; [Encode Message]
      call func_MPD_encode

   ; [Send Message]
      call func_MPD_Send_message

ret

smsg_not_responding:
   ; "Computer NOT Responding!"

   ; [Clear Storages]
      call func_MPD_Clear_Storages

   ; [Input Number]
      call set_number

   ; [Input Message]

      mov A, #'C'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'m'
      call func_MPD_input_char
      mov A, #'p'
      call func_MPD_input_char
      mov A, #'u'
```

```
      call func_MPD_input_char
      mov A, #'t'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'r'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'N'
      call func_MPD_input_char
      mov A, #'O'
      call func_MPD_input_char
      mov A, #'T'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'R'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'s'
      call func_MPD_input_char
      mov A, #'p'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'n'
      call func_MPD_input_char
      mov A, #'d'
      call func_MPD_input_char
      mov A, #'i'
      call func_MPD_input_char
      mov A, #'n'
      call func_MPD_input_char
      mov A, #'g'
      call func_MPD_input_char
      mov A, #'!'
      call func_MPD_input_char

   ; [Encode Message]
      call func_MPD_encode

   ; [Send Message]
      call func_MPD_Send_message
ret

smsg_computer_off:
   ; "Cancelled, power is off!"

   ; [Clear Storages]
      call func_MPD_Clear_Storages

   ; [Input Number]
      call set_number

   ; [Input Message]

      mov A, #'C'
      call func_MPD_input_char
      mov A, #'a'
      call func_MPD_input_char
      mov A, #'n'
      call func_MPD_input_char
      mov A, #'c'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'l'
      call func_MPD_input_char
```

```
      mov A, #'l'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'d'
      call func_MPD_input_char
      mov A, #','
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'p'
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'w'
      call func_MPD_input_char
      mov A, #'e'
      call func_MPD_input_char
      mov A, #'r'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'i'
      call func_MPD_input_char
      mov A, #'s'
      call func_MPD_input_char
      mov A, #' '
      call func_MPD_input_char
      mov A, #'o'
      call func_MPD_input_char
      mov A, #'f'
      call func_MPD_input_char
      mov A, #'f'
      call func_MPD_input_char
      mov A, #'!'
      call func_MPD_input_char

   ; [Encode Message]
      call func_MPD_encode

   ; [Send Message]
      call func_MPD_Send_message
ret

ResetCharIndex:
   mov MESSAGE_INDEX, #0
ret

ReadCharAndNext:
   ; Input: none
   ; Output: A

   ; Check the index first (is over the length?)
   mov A, MESSAGE_INDEX
   cjne A, MESSAGE_LENGTH, check_msg_length
      ; ( A == MESSAGE_LENGTH )
      ; give null terminating char
      mov A, #0
      jmp rcan_end
   check_msg_length:
      jc still_ok ; ( A < MESSAGE_LENGTH )
      jmp msg_over_length ; ( A > MESSAGE_LENGTH )
   end_msg_length_check:

   still_ok:

      ; indirect-addresing
      mov A, MESSAGE_INDEX
      add A, #MESSAGE_ADDRESS_OFFSET
      mov R0, A
```

```
      mov A, @R0 ; get data from RAM

      ; prepare next index address
      inc MESSAGE_INDEX

      jmp rcan_end
   msg_over_length:

      ; give char ' ' instead
      mov A, #' '

   rcan_end:

ret

ResetDigitIndex:
   mov NUMBER_INDEX, #0
ret

ReadDigitAndNext:
   ; Read current sender number data from RAM, and prepare for next data
   ; Input: none
   ; Output: A

   ; Check the index first (is over the length?)
   mov A, NUMBER_INDEX
   cjne A, NUMBER_LENGTH, check_num_length
      ; ( A == MESSAGE_LENGTH )
      ; give null terminating char
      mov A, #0
      jmp rdan_end
   check_num_length:
      jc still_ok2 ; ( A < MESSAGE_LENGTH )
      jmp num_over_length ; ( A > MESSAGE_LENGTH )
   end_num_length_check:

   still_ok2:

      ; indirect-addresing
      mov A, NUMBER_INDEX
      add A, #NUMBER_ADDRESS_OFFSET
      mov R0, A

      mov A, @R0 ; get data from RAM

      ; prepare next index address
      inc NUMBER_INDEX

      jmp rdan_end
   num_over_length:

      ; give char ' ' instead
      mov A, #' '

   rdan_end:

ret

Timer_Restart:
   ; Stop timer, reset it's variables (hour,minute,second) and start the timer
again.

   clr ET1 ; disable interrupt

   ; reset variables
   mov vTimer_Hours, #0
   mov vTimer_Minutes, #0
   mov vTimer_Seconds, #0
   mov vTimer_Milliseconds, #0
```

```
   mov vTimer_Reserved, #0
   mov vTimer_Reserved2, #0

   clr TR1 ; stop timer
   mov TH1, #06h ; 250 = 256 - x, x = 6
   mov TL1, #06h
   clr TF1 ; clear timer overflow flag
   setb TR1 ; start timer

   setb ET1 ; re-enable interrupt
ret

Timer_Start:

   ; reset variables
   mov vTimer_Hours, #0
   mov vTimer_Minutes, #0
   mov vTimer_Seconds, #0
   mov vTimer_Milliseconds, #0
   mov vTimer_Reserved, #0
   mov vTimer_Reserved2, #0

   setb ET1 ; enable interrupt

   setb TR1 ; start timer

ret

Timer_Stop:
   clr TR1
   clr TF1
   clr ET1
ret

convert_hex_to_int:
   ; FOR DEBUG PURPOSE ONLY
   ; [Conversion from HexToInt]
   ; Input : Acc
   ; Output : Acc

   ; [WARNING!!] :
   ;   This conversion result can up to 99 decimal points. The input maximum is
0x63.
   ;   Any overflow input occured, that may set output unexpectedly!
   ;   So precaution must be included before using this function!

   ; Source/Input : A

   mov B, #10
   div AB

   ; A = HIGH-ORDER decimal / puluhan [ division result ]
   ; B = LOW-ORDER decimal / satuan [ modulus result ]

   RL A ; convert to HIGH-order byte
   RL A
   RL A
   RL A

   orl A, B

   ; Destination : A
ret

display_dec:
   ; FOR DEBUG PURPOSE ONLY
   ; Input : Acc
   ; Output : Acc

   ; [WARNING!!]
```

```
   ;   This function only display decimal/digit-type hexadecimal only!!

   push Acc

   ; HIGH BYTE
   anl A, #0F0h ; get HIGH byte only
   RR A ; convert to LSB
   RR A
   RR A
   RR A
   ;   Convert into ASCII
   add A, #48
   ;   Write into LCD
   call LCD_Write_Data

   pop Acc
   ; LOW BYTE
   anl A, #0Fh ; get LOW byte only
   ;   Convert into ASCII
   add A, #48
   ;   Write into LCD
   call LCD_Write_Data

ret

;================================================================================
;                       DISPLAY TEXT FUNCTIONS
;================================================================================

display_wait_mpd:

   mov A, #'W'
   call LCD_Write_Data
   mov A, #'a'
   call LCD_Write_Data
   mov A, #'i'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #'i'
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data
   mov A, #'g'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'M'
   call LCD_Write_Data
   mov A, #'.'
   call LCD_Write_Data
   mov A, #'P'
   call LCD_Write_Data
   mov A, #'.'
   call LCD_Write_Data
   mov A, #'D'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'T'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data

   call LCD_Goto_2nd

ret

display_please_wait:
   mov A, #'P'
```

```
   call LCD_Write_Data
   mov A, #'l'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'a'
   call LCD_Write_Data
   mov A, #'s'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'w'
   call LCD_Write_Data
   mov A, #'a'
   call LCD_Write_Data
   mov A, #'i'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'.'
   call LCD_Write_Data
   mov A, #'.'
   call LCD_Write_Data
   mov A, #'.'
   call LCD_Write_Data
ret

display_checking_for_message:
   mov A, #'C'
   call LCD_Write_Data
   mov A, #'h'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'c'
   call LCD_Write_Data
   mov A, #'k'
   call LCD_Write_Data
   mov A, #'i'
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data
   mov A, #'g'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'f'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data

   call LCD_Goto_2nd

   mov A, #'N'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'w'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'M'
   call LCD_Write_Data
   mov A, #'e'
```

```
   call LCD_Write_Data
   mov A, #'s'
   call LCD_Write_Data
   mov A, #'s'
   call LCD_Write_Data
   mov A, #'a'
   call LCD_Write_Data
   mov A, #'g'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
ret

display_intruder_number:

   mov A, #'U'
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data
   mov A, #'a'
   call LCD_Write_Data
   mov A, #'u'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #'h'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data
   mov A, #'i'
   call LCD_Write_Data
   mov A, #'z'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'d'
   call LCD_Write_Data

   call LCD_Goto_2nd

   mov A, #'N'
   call LCD_Write_Data
   mov A, #'u'
   call LCD_Write_Data
   mov A, #'m'
   call LCD_Write_Data
   mov A, #'b'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data
   mov A, #'!'
   call LCD_Write_Data

ret

display_acknowledged:

   mov A, #'C'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'m'
   call LCD_Write_Data
   mov A, #'m'
   call LCD_Write_Data
   mov A, #'a'
```

```
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data
   mov A, #'d'
   call LCD_Write_Data

   call LCD_Goto_2nd

   mov A, #'A'
   call LCD_Write_Data
   mov A, #'c'
   call LCD_Write_Data
   mov A, #'k'
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'w'
   call LCD_Write_Data
   mov A, #'l'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'d'
   call LCD_Write_Data
   mov A, #'g'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'d'
   call LCD_Write_Data

ret

display_attention:
   mov A, #'A'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #'i'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data

ret

display_controller_disabled:
   mov A, #'S'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'c'
   call LCD_Write_Data
   mov A, #'.'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'C'
   call LCD_Write_Data
```

```
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'l'
   call LCD_Write_Data
   mov A, #'l'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data

   call LCD_Goto_2nd

   mov A, #'['
   call LCD_Write_Data
   mov A, #'D'
   call LCD_Write_Data
   mov A, #'i'
   call LCD_Write_Data
   mov A, #'s'
   call LCD_Write_Data
   mov A, #'a'
   call LCD_Write_Data
   mov A, #'b'
   call LCD_Write_Data
   mov A, #'l'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'d'
   call LCD_Write_Data
   mov A, #']'
   call LCD_Write_Data

ret

display_error:
   mov A, #'S'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'c'
   call LCD_Write_Data
   mov A, #'.'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'C'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'n'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'l'
   call LCD_Write_Data
   mov A, #'l'
```

```
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data

   call LCD_Goto_2nd

   mov A, #'E'
   call LCD_Write_Data
   mov A, #'R'
   call LCD_Write_Data
   mov A, #'R'
   call LCD_Write_Data
   mov A, #'O'
   call LCD_Write_Data
   mov A, #'R'
   call LCD_Write_Data
   mov A, #'!'
   call LCD_Write_Data

ret

display_not_responding:
   mov A, #'C'
   call LCD_Write_Data
   mov A, #'o'
   call LCD_Write_Data
   mov A, #'m'
   call LCD_Write_Data
   mov A, #'p'
   call LCD_Write_Data
   mov A, #'u'
   call LCD_Write_Data
   mov A, #'t'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'r'
   call LCD_Write_Data

   call LCD_Goto_2nd

   mov A, #'N'
   call LCD_Write_Data
   mov A, #'O'
   call LCD_Write_Data
   mov A, #'T'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'R'
   call LCD_Write_Data
   mov A, #'E'
   call LCD_Write_Data
   mov A, #'S'
   call LCD_Write_Data
   mov A, #'P'
   call LCD_Write_Data
   mov A, #'O'
   call LCD_Write_Data
   mov A, #'N'
   call LCD_Write_Data
   mov A, #'D'
   call LCD_Write_Data
   mov A, #'I'
   call LCD_Write_Data
   mov A, #'N'
   call LCD_Write_Data
   mov A, #'G'
```

```
   call LCD_Write_Data
   mov A, #'!'
   call LCD_Write_Data

ret

display_command_is:

   mov A, #'R'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'c'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'i'
   call LCD_Write_Data
   mov A, #'v'
   call LCD_Write_Data
   mov A, #'e'
   call LCD_Write_Data
   mov A, #'d'
   call LCD_Write_Data
   mov A, #' '
   call LCD_Write_Data
   mov A, #'C'
   call LCD_Write_Data
   mov A, #'M'
   call LCD_Write_Data
   mov A, #'D'
   call LCD_Write_Data
   mov A, #':'
   call LCD_Write_Data

ret

;================================================================================
;                      DELAY FUNCTIONS
;================================================================================

delay_1ms:
   clr TF0 ; clear overflow flag
   clr TR0 ; stop timer

   ; delay = 1000 us (1 ms)
   ; set value, 65536 - (delay) = 55536
   mov TH0, #HIGH 55536
   mov TL0, #LOW 55536

   setb TR0 ; start timer
   jnb TF0, $ ; wait until timer overflow
   clr TF0 ; clear overflow flag
ret

delay_50ms:
   clr TF0
   clr TR0
   mov TH0, #HIGH 15538 ; 65536 - 50000 = 15536
   mov TL0, #LOW 15538
   setb TR0
   jnb TF0, $
   clr TF0
ret

delay_1sec:
   mov A, TMP_A
   push Acc ; save TMP
   push PSW
```

```
   mov TMP_A, #20
   delay_1sec_loop:
      call delay_50ms
   djnz TMP_A, delay_1sec_loop

   pop PSW
   pop Acc
   mov TMP_A, A ; restore TMP
ret

;================================================================================
;                        LCD FUNCTIONS
;================================================================================

; ### expanded functions ###

   LCD_Clear_1st:
      ; Clear the 1st line of LCD Display
      ; > cursor goto 1st row, 1st coloumn
      call LCD_goto_1st

      ; Write ' ' for each coloumn
      mov B, #16
      LCD_Clear_1st_a:
         mov A, #' '
         call LCD_Write_Data
      djnz B, LCD_Clear_1st_a
      ; > Back to 1st row 1st coloumn
      call LCD_goto_1st
   ret

   LCD_Clear_2nd:
      ; Clear the 2nd line of LCD Display
      ; > cursor goto 2nd row, 1st coloumn
      call LCD_goto_2nd
      ; Write ' ' for each coloumn
      mov B, #16
      LCD_Clear_2nd_a:
         mov A, #' '
         call LCD_Write_Data
      djnz B, LCD_Clear_2nd_a
      ; > Back to 2nd row 1st coloumn
      call LCD_goto_2nd
   ret

   LCD_goto_1st:
      mov A, #(80h + 00h) ; goto 1st row of the LCD Lines
      call LCD_Write_Cmd
   ret

   LCD_goto_2nd:
      mov A, #(80h + 40h) ; goto 2nd row of the LCD Lines
      call LCD_Write_Cmd
   ret

; ### basic functions ###
LCD_Write_Cmd:
; 1. EN default is LOW.
; 2. Clear RS to LOW (Instruction Mode.)
; 3. Clear RW to LOW (Mode Write.)
; 4. Set EN to HIGH (enable LCD. Comm.)
; 5. Put data into LCD data lines
; 6. Clear EN to LOW
; write mode finished.

   clr EN
   clr RS
   clr RW
   setb EN
   setb EN ; delay 1us
```

```
   mov LCD_Lines, A ; OUT
   clr EN

   call LCD_Wait

ret

LCD_Write_Data:
; 1. EN default is LOW.
; 2. Set RS to HIGH (Data Mode.)
; 3. Clear RW to LOW (Mode Write.)
; 4. Set EN to HIGH (enable LCD. Comm.)
; 5. Put data into LCD data lines
; 6. Clear EN to LOW
; write mode finished.

   clr EN
   setb RS
   clr RW
   setb EN
   setb EN ; delay 1us
   mov LCD_Lines, A ; OUT
   clr EN

   call LCD_Wait
ret

LCD_Read_Data:
; This function read current data on current RAM pointer on LCD.
; data stored at Acc.

; 1. EN default is LOW.
; 2. Set RS to HIGH (Data Mode.)
; 3. Set RW to HIGH (Mode Read.)
; 4. Set EN to HIGH (enable LCD. Comm.)
; 5. Put data into LCD data lines
; 6. Clear EN to LOW
; read mode finished.
   clr EN
   setb RS
   setb RW
   setb EN
   setb EN ; delay 1us
   mov A, LCD_Lines ; data direction IN
   clr EN
ret

LCD_Read_Cmd:
; data stored at Acc.

; 1. EN default is LOW.
; 2. Clear RS to LOW (Instruction Mode.)
; 3. Set RW to HIGH (Mode Read.)
; 4. Set EN to HIGH (enable LCD. Comm.)
; 5. Put data into LCD data lines
; 6. Clear EN to LOW
; read mode finished.
   clr EN
   clr RS
   setb RW
   setb EN
   setb EN ; delay 1us
   mov A, LCD_Lines ; data direction IN
   clr EN
ret

; ### end basic functions ###

; ### complex functions ###
LCD_Init:
```

```
; 1. Function Set
; 2. Display ON/OFF (SWAPPED)
; 3. Entry Mode Set (SWAPPED)
; 4. Return Home.

; 1. Function Set
; configurations : Binary( 001_DL._N_F_XX )
; DL = 1, 8 bit interface.
; N = 1, 2 Line Display. We use both of them.
; F = 0, 5x8 Font. We use 2 line, so font must be small.

   mov A, #00111000b
   call LCD_Write_Cmd

; 2. Display ON/OFF
; configurations : Binary( 0000.1_D_C_B )
; D = 1, Display ON
; C = 0, Cursor OFF
; B = 0, Cursor Blinkin OFF

   mov A, #00001100b
   call LCD_Write_Cmd

; 3. Entry Mode Set, set cursor moving direction
; configurations : Binary( 0000.01_ID_S )
; ID = 0, display shift left
; S = 0, no screen shifting.

   mov A, #00000110b
   call LCD_Write_Cmd

; 4. Return Home, set cursor to top-left position.
; configurations : Binary( 0000.001X )

   mov A, #02h
   call LCD_Write_Cmd

ret

LCD_Clear:
   mov A, #01h
   call LCD_Write_Cmd
ret

LCD_Wait:
; NOTE: WE CAN'T USE LCD_Read_Cmd, coz we need processing inside the function.

; Get system status, busy or not?
; if busy, hold on here, if not, instruction goes on.
; Configurations : Binary( BF[Bit7]_AddressCounterContent[Bit6-0] )

   clr EN
   clr RS ; Instruction Mode.
   setb RW ; Read mode.

   mov LCD_Lines, #0FFh ; pull-up plz

   setb EN ; start E-Cycle.
   setb EN ; delay 1us

   jb LCD_Lines_bit7, $ ; hold here, if BF still 1, which mean busy.
   ; LCD Ready.

   clr EN ; close cycle.

ret
; ### end complex functions ###

end
```