```
;[]------------------------------------------------------------[]
;|                                                              |
;|                                                              |
;|                    Example program for HSETI                 |
;|                                                              |
;|                                                              |
;|                    Steven Murray, AirBorn Electronics, 1996  |
;[]------------------------------------------------------------[]

;This code is copyright (C)1996 _____________________
;If you modify this code, put your name in the copyright entry above.

;( This example code, written by  AirBorn Electronics in 1995, is released )
;(  into the public domain with no copyright claim or cover, to be copied  )
;( and modified freely, in whole or in part, to form the basis of programs )
;(     written by other parties who may in turn claim copyright on any     )
;(    code added or modifications made to this material, as they desire.   )

.include "8051.H"
;Without this file, the TASM assembler does not recognise any of the 8051
;  port locations automatically as other 8051 specific assemblers do.
;The file 8051.H contains all the 8051 port definitions, as equates.

;While this code is written so as to assemble on the TASM301 series
;  assemblers, it should assemble with minor modification on most other
;  8051 assemblers.  Indeed, you might find that other specific 8051
;  assemblers are slightly better for long term multi-project development
;  work, as by its very flexible multi-CPU nature TASM is not quite as
;  good at picking up syntactical coding errors as some other assemblers.

.TITLE "*Demo code, AirBorn Electronics 61-2-9925-0325, V1.00a"
;This 8051 example code is for the HSETI demonstration PCB.


;Revision History ---------------------------------------------------------

;It is common practice to put a version number on each file, and a history
;  at the top of the file that lists past version numbers, features added
;  and bugs fixed at each step, the date, and the programmer.

;19/10/96  SM    V1.00a  Original skeleton code to run HSETI board

;[]------------------------[]
;|  Constants               |
;[]------------------------[]

;By defining constants in your program, and then using the constant's name in
;  place of a numeric value, you can make your program more readable and more
;  easily modifiable.  Care is required, as TASM is case sensitive.

;A classic example might be "LineLen equ 80" used in a program in such a way
;  that a microcontroller that happened to do output printing used "LineLen"
;  in its program to format text, (e.g. "subb A,#LineLen") rather than just
;  "80".  At a later date it might be possible to alter the program for a
;  different printer by changing LineLen to 72 or 64 - without changing code.

Crystal equ     11059200        ;Crystal frequency
                                ;The crystal is entered as an equate so that
                                ;  timing loops may be calculated at assembly
                                ;  time.  By making the value for all timing
                                ;  loops dependent on this equate, just one
                                ;  number needs to be changed to alter the
                                ;  crystal  frequency for all timing in the
                                ;  program.

;[]------------------------[]
;| Input/Output allocation  |
;[]------------------------[]

;By defining input/outputs as equates, and then using the I/O's long name in
;  place of the I/O pins name, it is easy to alter the program later for new
```

```
;  hardware, and the program is more readable.

;Refer to schematic diagram AB9609231:  HSETI 1 CPU PCB, to match bit names

LEDsPort equ    P1
LiteIn   equ    P3.2
SoundOut equ    P3.7


;[]------------------------[]
;| Hardware and Operational |
;[]------------------------[]

;The HSETI PCB is a simple, small PCB with a AT89C2051 CPU and a driver/buffer
;The original application was a LED display unit which was triggered from a
;light sensor, with a simple bit output to drive a speaker as well.  Only the
;skeleton code is provided here - it simply counts up on all the LED outputs,
;and then sets the whole lot to zero if the light sensor goes active.  At
;the same time it pulses the speaker output.  You could use this code as a
;test for a newly built HSETI PCB, as it exercises all the inputs and
;outputs, and then you could write your own code to do whatever you want.

;If you are wanting to add serial COMs to your HSETI PCB, you could look
;at downloading the EX2051 PCB example.asm file, which includes serial
;routines that might be suitable.

;[]------------------------[]
;|  RAM Memory allocation   |
;[]------------------------[]

;Defining memory locations as equates, then using long names in referencing,
;makes a program more readable.

StackBase       equ     30h     ;Start of stack in internal RAM



;[]------------------------[]
;| Reset/Interrupt Vectors  |
;[]------------------------[]

;At reset 8051 type microprocessors start executing at location 0.  There
;should be a jump at location zero to the start of the microprocessor code.

;When an 8051 type microprocessor receives an interrupt, it pushes the
;program counter onto the stack and jumps to a location near 0 - the location
;depends on the Interrupt:  IE0: 3, TF0: 0Bh, IE1: 13h, TF1: 1Bh, Ser: 23h
;There should be interrupt code and/or jumps to interrupt code at these locs.

        ORG     0               ;CPU Jumps here at reset
                                ;The .ORG directive starts code at that loc'tn.
                                ;Other assemblers use "CSEG AT 0"
        ajmp    Init


        ORG     3h              ;CPU Calls here on external int 0, if enabled
                                ;The program should jump here to int0 routine
        ajmp    Stub            ;INT0 is not wanted, INT0 does not happen
                                ;  unless it is enabled, but in case it does
                                ;  accidentally get enabled jump to stub


        ORG     0Bh             ;CPU Calls here on timer 0 overflow, if enabled
        ajmp    Stub            ;The program should jump to Timer 0 routine


        ORG     13h             ;CPU Calls here on external int 1, if enabled
                                ;The program should jump to ext int 1 routine
        ajmp    Stub            ;INT1 int is not enabled in the example program
                                ; and so should never happen - if it does then
                                ;  jump to "stub"
```

```
        ORG     1Bh             ;CPU Calls here on timer 1 overflow, if enabled
                                ;The program should jump to Timer 1 routine
        ajmp    Stub            ;TF1 int is not enabled in the example program
                                ; and so should never happen - if it does then
                                ;  jump to "stub"


        ORG     23h             ;CPU Calls here on Serial interrupt, if enabled
        ajmp    Stub            ;The program should jump to Serial Int routine


;[]------------------------[]
;|    Interrupt Routines    |
;[]------------------------[]

Stub:   clr     EA              ;Unexpected interrupt - disable all ints
                                ;If it is possible to give some indication of
                                ;  error, such as lighting an LED, do it here
        acall   AReti           ;Flag the interrupt routine is finished
                                ;A "reti" instruction tells the CPU that its
                                ;  interrupt routine has finished and it
                                ;  should return to the main program.
                                ;  However as we "called" the reti the CPU is
                                ;  manipulated into returning here, but in an
                                ;  "all interrupts finished" condition.
        ajmp    Init            ;Re-initialize: So unexpected Int causes reset
AReti:  reti


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


;[]------------------------[]
;|   Setup                  |
;[]------------------------[]

Init:                           ;Here at initialization
                                ;Referring to the 8051 data sheets, regs (e.g.
                                ;  Acc) have a known state at reset (usually 0)
                                ;RAM powers up indeterminate, or in the state
                                ;  it was in previously if Vcc stayed above
                                ;  the minimum power-down voltage.
                                ;If "Init" is jumped to as part of program
                                ;  operation, then the the registers could
                                ;  have any value... we init both registers
                                ;  and RAM in this program.

        mov   SP,#(StackBase-1) ;Initialise the Stack pointer
        mov     PSW,#0          ;Set active register bank (Regbnk 0: Locs 0-7)

        clr     A
                                ;Clear all RAM - not all applications would
                                ;  want this - if you are battery backing the
                                ;  CPU and keeping values in RAM, don't clear!
        mov     R0,#7Fh         ;Start at location 7Fh - work down
        mov     R1,#7Eh         ;Do 7Eh locations - stop before R0/R1
                                ;DEBUG - Changed from value 7Dh 22/7/95
Put0:   mov     @R0,A           ;Clear a byte of RAM
        dec     R0              ;Update the RAM pointer
        djnz    R1,Put0         ;Loop until all locations cleared
        dec     R0              ;Clear location 0 - the last location

;[]------------------------[]
;|    Main Program          |
;[]------------------------[]


Main:   jnb     LiteIn,LiteOn   ;Jump if the light input is active (low)
                                ;The light sensor input is not active
```

```
        acall   POFDel1         ;Do a delay
        cpl     SoundOut        ;Complement the sound bit
        inc     LEDsPort        ;Increment the LED port
        ajmp    Main            ;Jump back to start


LiteOn:                         ;The light sensor input is active
        acall   POFDel1         ;Do a delay
        cpl     SoundOut        ;Complement the sound bit
        mov     LEDsPort,#0h    ;Turn all the LEDs off
        ajmp    Main            ;Jump back to start


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


;[]------------------------[]
;|    Subroutines           |
;[]------------------------[]


POFDel  equ     10              ;Desired delay in milliseconds
POFDel1:
        mov  R5,#((Crystal*POFDel)/6180000) ;Work out the djnz delay value
POFDel2:
;       mov     R4,#0
POFDel3:
        djnz    R4,POFDel3      ;This inside loop takes 515 cycles
        djnz    R5,POFDel2      ;This loop happens the correct number of times
                                ;  to get closest to the desired delay in ms
        ret                     ;Return when delay is complete


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

        END
```