3 A51 Examples

In this chapter we present a few assembly language programs which use most of the topics discussed in the previous chapters. Before starting to write the first program, we provide a template which explains the general organisation of an 8051 assembly language program. The remarks by the side of the instructions in the template and in the other example programs provide most of the explanations required.

Further examples in C are provided in the Appendix.

3.1 Template.a51

```
; Template.a51
$NOMOD51
#include <reg52.inc>                ; assuming that we are using an 8032 instead of an 8051
                                    ; reg52.inc would include all the SFRs present on the 8032

start   equ 0000H                   ; these equates can be changed if using a development
                                    ; board, depending on where the code is to reside.
Ext0_IVA        equ 0003H           ; Interrupt Vector address for External 0 interrupt number 0
TF0_IVA         equ 000BH; Interrupt Vector address for Timer 0 interrupt number 1
Ext1_IVA        equ 0013H           ; Interrupt Vector address for External 1 interrupt number 2
TF1_IVA         equ 001BH; Interrupt Vector address for Timer 1 interrupt number 3
Ser_IVA         equ 0023H           ; Interrupt Vector address for Serial interrupt number 4
TF2_IVA         equ 002BH; Interrupt Vector address for Timer 2 interrupt number 5

Past_IVT equ 0030H

; The following equates are used for RAM zero initialisation routines
IDATASTART      EQU 0H              ; the absolute start-address of IDATA memory is always 0
IDATALEN        EQU 100H            ; the length of IDATA memory in bytes for the 8032 (256 bytes).

XDATASTART      EQU 0H              ; the absolute start-address of XDATA memory (say 8100H)
XDATALEN        EQU 0H              ; the length of XDATA memory in bytes.

CSEG AT start
        LJMP Main                   ; this is the first instruction executed on reset

CSEG AT Ext0_IVA
        RETI                        ; good practice to include this if not using interrupt, just in case.
                                    ; comment above code if this interrupt is being used
; or if the ISR code is within 8 bytes long, it can be written directly here.
; if not then use
;       LJMP Ext0_ISR               ; to jump to the correct ISR

CSEG AT TF0_IVA
;       RETI                        ; good practice to include this if not using interrupt, just in case.
                                    ; comment above code if this interrupt is being used
; or if the ISR code is within 8 bytes long, it can be written directly here.
; if not then use
        LJMP TF00_ISR               ; to jump to the correct ISR
; and so on for the other interrupts
```

```
CSEG AT Ext1_IVA
        RETI                    ; good practice to include this if not using interrupt, just in case.
                                ; comment above code if this interrupt is being used
; or if the ISR code is within 8 bytes long, it can be written directly here.
; if not then use
;       LJMP Ext1_ISR           ; to jump to the correct ISR

CSEG AT TF1_IVA
        RETI                    ; good practice to include this if not using interrupt, just in case.
                                ; comment above code if this interrupt is being used
; or if the ISR code is within 8 bytes long, it can be written directly here.
; if not then use
;       LJMP TF1_ISR            ; to jump to the correct ISR

CSEG AT Ser_IVA
        CLR RI                  ; good practice to include this if not using interrupt, just in case.
        CLR TI                  ; good practice to include this if not using interrupt, just in case.
        RETI                    ; good practice to include this if not using interrupt, just in case.
                                ; comment above 3 code lines if this interrupt is being used
; or if the ISR code is within 8 bytes long, it can be written directly here.
; if not then use
;       LJMP Ser_ISR            ; to jump to the correct ISR

CSEG AT TF2_IVA
        CLR TF2                 ; good practice to include this if not using interrupt, just in case.
        CLR EXF2        ; good practice to include this if not using interrupt, just in case.
        RETI                    ; good practice to include this if not using interrupt, just in case.
                                ; comment above 3 code lines if this interrupt is being used
; or if the ISR code is within 8 bytes long, it can be written directly here.
; if not then use
;       LJMP Ext0_ISR           ; to jump to the correct ISR

; skip over Interrupt Vector Table in the code area
org Past_IVT
Main:
; First clear the 8032 Internal RAM (from 0 to FFH)
        CLR A
        MOV R0,#(IDATALEN - 1)
CLR_RAM:
        MOV @R0,A
        DJNZ R0,CLR_RAM

; then clear external RAM if required, using conditional assembly, depending on XDATALEN
IF XDATALEN <> 0
        MOV     DPTR,#XDATASTART
        MOV     R7,#LOW (XDATALEN)
IF (LOW (XDATALEN)) <> 0 ; check needed so that the DJNZ checks below will work
                                ; correctly, since if R7 is zero before the DJNZ, it will loop
                                ; for 256 times and not zero times.
                                ; ( check with XDATALEN of 255 bytes and then 256 bytes !! )
        MOV     R6,#(HIGH (XDATALEN) +1)
ELSE
        MOV     R6,#(HIGH (XDATALEN))
ENDIF
```

```
        CLR       A
CLR_XDATA:
        MOVX      @DPTR, A
        INC       DPTR
        DJNZ      R7,CLR_XDATA
        DJNZ      R6,CLR_XDATA
ENDIF

; set up stack pointer, above Bit-addressable area (not necessarily always set to this point)
        MOV SP,#2FH

; Program starts here
……………….
……………....
……………….

; Long Interrupt Service Routines can be written here, after the main program
TF0_ISR:
        PUSH PSW
    ………..
    ……………
        POP PSW
        RETI

; Constants can be stored here, at the end of the code area.
 OneHundred: DB 100
 SixHundred: DW 600
 Message: DB "Hello !!",10,13

; Variables can be stored either in the internal 256 bytes data area or in the external volatile
; memory. Bit variables are stored in the bit data area

MyBits SEGMENT BIT
RSEG MyBits
        Flag1:    DBIT 1              ; 1 bit in Bit-addressable area, allocated to Flag1
        Flag2:    DBIT 1              ; 1 bit in Bit-addressable area, allocated to Flag2

Var1 SEGMENT DATA
RSEG Var1
        Answer:   DS        1         ; 1 byte in data area, allocated to Answer
        Year:     DS        2         ; 1 bytes in data area, allocated to Year
        Month:    DS`       1         ; 1 byte in data area, allocated to Month

Var2 SEGMENT XDATA
RSEG Var2
        Numbers:  DS        500       ; 500 bytes allocated to Numbers, in external RAM

end
```

The first real program, SerP3.a51 is a serial port example program (section 3.2) which basically initializes the UART and then provides routines for reading and writing characters via the UART.

The second program (3.3) is a simple Traffic light controller which also uses the SerP3.a51 routines. It makes use of Timer 2 interrupt which is used to accurately time the duration in seconds for each traffic pattern. Note the way the Interrupt Vector Table (IVT) is jumped over when the program starts executing from location 0000H. For those Interrupts which are not in use, it is generally a good practice to insert a simple RETI instruction at the corresponding IVT location just in case an inadvertent event causes an undesired interrupt to occur.

3.2 Serial Port Example Program

```
; SERP3.A51
; march 2003 - paul p. debono
; works fine using p3 serial socket
; no interrupts
;
$NOMOD51
#include <reg52.inc>

; These routines are declared PUBLIC so that they can be used in other modules
PUBLIC INIT_SERIAL, TX_CHAR, RX_CHAR
PUBLIC TX_IMSG,TX_CMSG, TX_XMSG
;
; SERIAL PORT RELATED ROUTINES
;
; INIT_SERIAL(BAUDRATE)        Initialise Serial port, 9600, 19200 or 57600 baud.
; RX_CHAR()                    Receive character from port, (WAIT FOR CHARACTER)
; TX_CHAR(ALPHA)               Send character to Port
; TX_MSG(*MESSSAGE)            Transmit null terminated string (Internal RAM)
; TX_CMSG(*MESSSAGE)           Transmit null terminated string (PROGRAM CODE AREA)
; TX_XMSG(*MESSSAGE)           Transmit null terminated string (External DATA AREA)
;

SERIAL_RTN SEGMENT CODE
RSEG SERIAL_RTN

; SUBROUTINES USED IN APPLICATION
;
; ***********************************************
;
; serial port support
;
; initialise the serial port for required baud rate,
; not under interrupt control.

; baud rate passed in r7 bank 0
;     parameter 96 => 9600 baud
;     parameter 192 => 19200 baud
;     parameter 57 => 57600 baud
```

```
INIT_SERIAL:
          ANL TMOD, #00001111B          ; clear all timer 1 bits in tmod
          ORL TMOD, #20H                ; timer 1 8-bit auto reload mode 2
          CLR RCLK                      ; use timer 1 for receive baud rate
          CLR TCLK                      ; use timer 1 for transmit baud rate
          MOV TH1, #0FDH                ; 9600/19200 baud counter value
          MOV TL1, #0FDH
          MOV PCON, #0H                 ; choose 9600 baud
          CJNE R7, #192, CHK_IF_57
          MOV PCON, #80H                ; choose 19200 baud, smod=1
          SJMP BAUD_OK
CHK_IF_57:
          CJNE R7, #57, BAUD_OK
          MOV TH1, #0FFH                ; 57600 baud counter value
          MOV TL1, #0FFH
          MOV PCON, #80H                ; smod=1
BAUD_OK:
          CLR ET1                       ; disable timer 1 interrupts, just in case
          SETB TR1              ; start timer 1 (tr1 = 1) or mov tcon,#40h
          MOV SCON, #52H                ; 1 start, 8 data, 1 stop bit, RI=0, and setTI=1
                                        ; so as to be ready to start the first time
                                        ; enable receiver (ren=1)

    RET
; **********************************************

; **********************************************
; character received through serial port p3, is passed on to r7 bank 0 (address 07)
RX_CHAR:
          JNB RI, $                     ; wait here for character
          CLR RI
          MOV 07, SBUF
          RET
; **********************************************

; **********************************************
; character in r7 is transmitted through serial port p3
TX_CHAR:
          JNB TI, $             ; if tx is ready, then you are clear to send, else wait
          CLR TI
          MOV SBUF, 07                  ; transmission starts, t1 set to 1 when ready

; the following delay might be needed depending on receiving equipment requirements
          PUSH B
          MOV B, #0A0H                  ; small delay between transmissions
          DJNZ B, $             ; since we are not using any handshaking
          POP B

    RET
; **********************************************

; **********************************************
```

```
; transmit message residing in internal memory
; pointer to message passed in r1
; message must terminate with a null (0) character.
; on exit, r1 is corrupted

TX_IMSG:
        MOV A, @R1
        CJNE A, #0, SEND_IT
        RET
SEND_IT:
        MOV 07, A
        ACALL TX_CHAR
        INC R1
        SJMP TX_IMSG
; **********************************************

; **********************************************
; transmit message residing in program (code) memory
; pointer to message passed in dph (hi) and dpl (lo)
; message must terminate with a null (0) character.
; on exit, dptr is corrupted.
TX_CMSG:
        CLR A
        MOVC A, @A + DPTR
        CJNE A, #0, SEND_IT2C
        RET
SEND_IT2C:
        MOV 07, A
        ACALL TX_CHAR
        INC DPTR
        SJMP TX_CMSG
; **********************************************

; **********************************************
; transmit message residing in external memory
; pointer to message passed in dph (hi) and dpl (lo)
; message must terminate with a null (0) character.
; on exit dptr is corrupted.
TX_XMSG:
        MOVX A, @DPTR
        CJNE A, #0, SEND_IT2
        RET
SEND_IT2:
        MOV 07, A
        ACALL TX_CHAR
        INC DPTR
        SJMP TX_XMSG
; **********************************************
 END
; **********************************************
```

The second program (section 3.3) is Traffic lights program, and it is targeted to be run from an EPROM. This means that the code area starts from location 0000H. It is also targeted for the FLT-32 development board, which has an 8255 input/output chip added on, providing three additional 8-bit ports, labelled as PORTA, PORTB and PORTC in this program.

3.3 Traffic Lights A51 Program

```
; lesson07EP.a51 targeted for eprom

$NOMOD51
#include <reg52.inc>
; use model for 8032/8052
; this will ensure that the assembler will recognise
; all the labels referring to the various
; Special Function Registers (SFRs).
;
;
; Traffic lights program with TIMER2 delay
; in 16-bit AUTO-RELOAD mode
;
; Timers count up at 12/11.0592 microseconds per count
; i.e. at 1.085 microseconds per count.
; Thus for a 50 millisecond delay, they need to count
; up 50000/1.085 = 46082 times. Hence the counters
; have to be loaded with 65536-46082 = 19454 decimal
; or 4BFEH
;
;
; The following routines are declared as EXTRN (within brackets)
; since they are actually defined in a different module.
;
EXTRN CODE (INIT_SERIAL, RX_CHAR, TX_CHAR)
EXTRN CODE (TX_IMSG, TX_CMSG, TX_XMSG)
;
; serial port related routines found in serp3.a51
;
; INIT_SERIAL(BAUDRATE)       Initialise Serial port, 9600, 19200 or 57600 baud.
; RX_CHAR()                   Receive character from port, (WAIT FOR CHARACTER)
; TX_CHAR(ALPHA)              Send character to Port
; TX_MSG(*MESSSAGE)           Transmit null terminated string (Internal RAM)
; TX_CMSG(*MESSSAGE)          Transmit null terminated string (PROGRAM CODE AREA)
; TX_XMSG(*MESSSAGE)          Transmit null terminated string (External DATA AREA)
;
CR          EQU 13
LF          EQU 10

CTRL_WD     EQU 91H           ; control word for the 8255
PORTA       EQU 0FF40H        ; 8255 ports addresses in FLT-32 board
PORTB       EQU 0FF41H
PORTC       EQU 0FF42H
CONTROL     EQU 0FF43H

; Interrupts vector table location when targeting EPROM.
RESET       EQU 0000H
EXT0_ISR_VEC        EQU 0003H
T0_ISR_VEC          EQU 000BH
EXT1_ISR_VEC        EQU 0013H
T1_ISR_VEC          EQU 001BH
SERIAL_ISR_VEC      EQU 0023H
T2_ISR_VEC          EQU 002BH
```

```
PROG_AREA                EQU 0030H          ; Main program area starting point
;
;
        ORG RESET
        LJMP MAIN                   ; Continue with MAIN, jumping over the interrupt vector table.

        ORG EXT0_ISR_VEC
;       LJMP EXT0_ISR               ; point to my interrupt service routine
        RETI                        ; remark this line if using ISR

        ORG T0_ISR_VEC
        ; LJMP T0_ISR               ; point to my interrupt service routine
        RETI                        ; remark this line if using ISR

        ORG EXT1_ISR_VEC
;        LJMP EXT1_ISR              ; point to my interrupt service routine
        RETI                        ; remark this line if using ISR

        ORG T1_ISR_VEC
;       LJMP T1_ISR                 ; point to my interrupt service routine
         RETI                       ; remark this line if using ISR

        ORG SERIAL_ISR_VEC
;       LJMP SERIAL_ISR             ; point to my interrupt service routine
        RETI                        ; remark this line if using ISR

        ORG T2_ISR_VEC
        LJMP T2_ISR                 ; point to my timer 2 service routine
;       RETI                        ; remark this line if using ISR

ORG PROG_AREA

MAIN:
; initialise stack pointer past bit-addressable area
        MOV SP, #30H

; First clear the 8032 Internal RAM (from 0 to FFH)
        CLR A
        MOV R0, #0FFH
CLR_RAM:
        MOV @R0, A
        DJNZ R0, CLR_RAM

; initialise serial port
        MOV R7, #57
        LCALL INIT_SERIAL

; print message
        MOV DPTR, #MESSAGE1
        LCALL TX_CMSG

; initialise 8255 i/o chip
        MOV DPTR, #CONTROL
        MOV A, #CTRL_WD
        MOVX @DPTR, A               ; initialise 8255 ports mode
```

```
; initialise timer 2
        MOV RCAP2H, #4BH            ; re-load timer counters
        MOV RCAP2L, #0FEH           ; 50 msec timer delay
        MOV TH2, RCAP2H             ; used for the first interrupt
        MOV TL2, RCAP2L

        MOV T2CON, #0               ; set timer 2 16-bit auto-reload

        SETB TR2                    ; start timer 2
        SETB ET2                    ; enable interrupts from timers
        SETB EA                     ; allow interrupts
        SETB PT2                    ; Timer 2 with high priority

; start traffic lights
        MOV DPTR, #TABLE            ; point DPTR to table
        MOV A, DPL                  ; DEC DPTR
        JNZ DECSKIP
        DEC DPH
DECSKIP:
        DEC DPL                     ; DPTR now points ahead of TABLE
        MOV R7, #1
        MOV R6, #1                  ; R6,R7 set to 1 to start immediately from Top of Table
        SETB TF2                    ; simulate timer 2 interrupt

LOOP: SJMP LOOP                     ; main program simply loops here
                                    ; forever. It could be doing something
                                    ; else whilst the timer takes care of
                                    ; scheduling the display pattern.

; traffic control isr.
; permanent data stored in code (eprom) area
T2_ISR:
        CLR TF2                     ; clear interrupt flag
        PUSH ACC
        DJNZ R6, EXIT_NOW           ; exit immediately if 1 second
                                    ; has not yet passed
        MOV R6, #20                 ; reset R6 otherwise
        DJNZ R7, EXIT_NOW           ; has required time passed ?
        INC DPTR                    ; yes
        CLR A                       ; we need to clear it first
        MOVC A, @A + DPTR           ; get next pattern
        JNZ SKIP                    ; 0 pattern indicates end of table, hence start again
        MOV DPTR, #TABLE            ; acc=0 hence no need to clear it
        MOVC A, @A + DPTR           ; load 1st pattern in Acc,
SKIP: PUSH DPH
        PUSH DPL
        MOV DPTR, #PORTB
        MOVX @DPTR, A               ; light up leds with pattern
        POP DPL
        POP DPH
        INC DPTR
        CLR A
        MOVC A, @A + DPTR           ; get duration byte and
        MOV R7, A                   ; store the seconds in R7
```

```
; initialise timer 2
          MOV RCAP2H, #4BH          ; re-load timer counters
          MOV RCAP2L, #0FEH         ; 50 msec timer delay
          MOV TH2, RCAP2H           ; used for the first interrupt
          MOV TL2, RCAP2L

          MOV T2CON, #0             ; set timer 2 16-bit auto-reload

          SETB TR2                  ; start timer 2
          SETB ET2                  ; enable interrupts from timers
          SETB EA                   ; allow interrupts
          SETB PT2                  ; Timer 2 with high priority

; start traffic lights
          MOV DPTR, #TABLE          ; point DPTR to table
          MOV A, DPL                ; DEC DPTR
          JNZ DECSKIP
          DEC DPH
DECSKIP:
          DEC DPL                   ; DPTR now points ahead of TABLE
          MOV R7, #1
          MOV R6, #1                ; R6,R7 set to 1 to start immediately from Top of Table
          SETB TF2                  ; simulate timer 2 interrupt

LOOP: SJMP LOOP                     ; main program simply loops here
                                    ; forever. It could be doing something
                                    ; else whilst the timer takes care of
                                    ; scheduling the display pattern.

; traffic control isr.
; permanent data stored in code (eprom) area
T2_ISR:
          CLR TF2                   ; clear interrupt flag
          PUSH ACC
          DJNZ R6, EXIT_NOW         ; exit immediately if 1 second
                                    ; has not yet passed
          MOV R6, #20               ; reset R6 otherwise
          DJNZ R7, EXIT_NOW         ; has required time passed ?
          INC DPTR                  ; yes
          CLR A                     ; we need to clear it first
          MOVC A, @A + DPTR         ; get next pattern
          JNZ SKIP                  ; 0 pattern indicates end of table, hence start again
          MOV DPTR, #TABLE          ; acc=0 hence no need to clear it
          MOVC A, @A + DPTR         ; load 1st pattern in Acc,
SKIP: PUSH DPH
          PUSH DPL
          MOV DPTR, #PORTB
          MOVX @DPTR, A             ; light up leds with pattern
          POP DPL
          POP DPH
          INC DPTR
          CLR A
          MOVC A, @A + DPTR         ; get duration byte and
          MOV R7, A                 ; store the seconds in R7
```

```
EXIT_NOW:
        POP ACC
        RETI

; permanent data, can be stored in code area (not rewriteable)
; Table storing Pattern and Duration in seconds
;
TABLE:
        DB 82H,10           ; R - G
        DB 84H,2            ; R - Y
        DB 88H,1            ; R - R
        DB 0C8H,2           ; RY - R
        DB 28H,8            ; G - R
        DB 48H,2            ; Y - R
        DB 88H,1            ; R - R
        DB 8CH,2 ; R - RY
        DB 0                ; end of array marker

; Message must terminate with a zero for correct performance of the print routine
MESSAGE1:       db      13,10,'This is a serial port test',CR,LF,LF
                db      'Read the program carefully and try to',CR,LF
                db      'understand it well',CR,LF,0

        END
```