

Fig. 9(a): Flow-chart for the access-control system, continued in Figs 9(b) and 9(c)

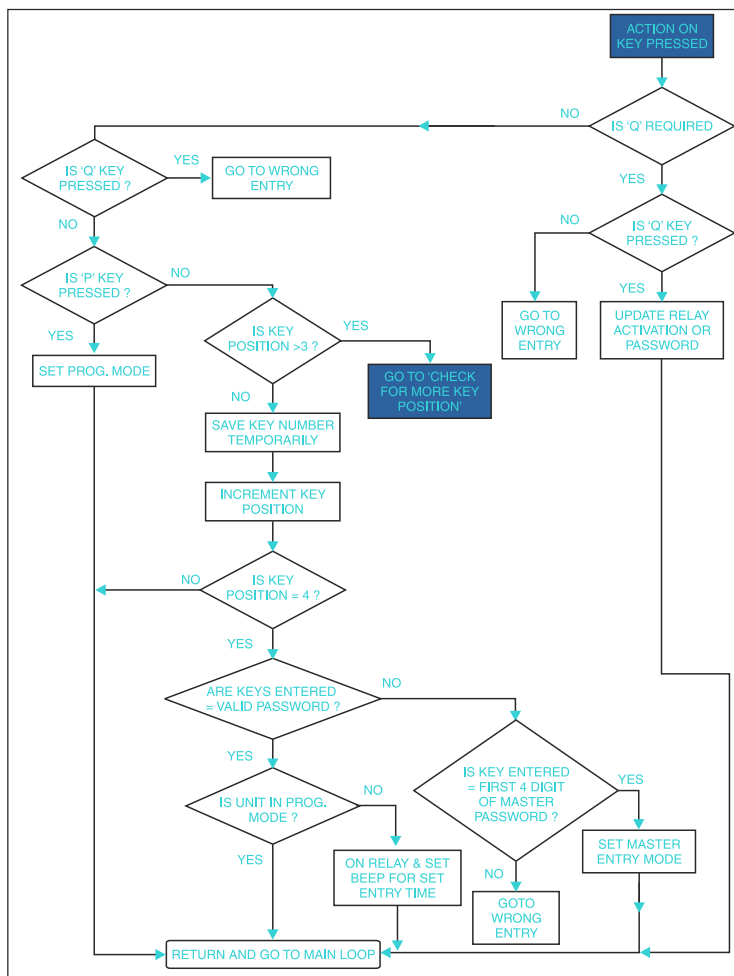


Fig. 9(b): Flow-chart for the access-control system, continued from Fig. 9(a)

tivation time duration. On changing the same, the new password and changed relay-activation time are saved in the memory, which will be recalled at the next power-on. (**Note.** In case you have forgotten the changed password, you cannot operate the unit unless you install a new/blank memory.)

Caution. Take care while connecting and using the live 220V wires.

The software

For software development the author has taken the help of Understanding Small Microcontrollers, MC68HC705KJ1 Technical Data book, and In-Circuit Simulator User's Manual. The development tools used include WinIDE software for KJ1 (including editor, assembler, simulator and programmer), in-circuit simulator (referred to as JICS board), and IBM PC with Windows OS and CD drive.

DOS-based programs can also be used for software development. So if you are comfortable with DOS or have an old computer with limited hard disk capacity, you will still face no difficulty.

(**Note** The books (in pdf format) and WinIDE software are available free of cost on Motorola's Website and have been reproduced by courtesy of Motorola in EFY-CDs of this year's January and February issues. The mentioned CDs also contain DOS-based programs. The JICS board may be bought from Motorola's authorised distributors.)

Program development steps can write the software by using the following steps:

1. Read and understand the microcontroller's operation and instructions as well as the operation of WinIDE software. (The help option of the software will clear most of your doubts.) You should also have a clear knowledge of the logic sequence of the end-product operation. For this, you can make a flow-chart. (Flow-chart for this access control system is shown in Figs 9(a)-(c). The corresponding software source code is given at the end of this article.)

2. Convert the flow-charts to source program in Assembly language making use of the instruction set of the microcontroller and assembler directives.

Now insert the memory IC and change the password and the relay-activation time to 10 seconds as default parameters.

Now insert the memory IC and change the password and the relay-activation time to 10 seconds as default parameters. The

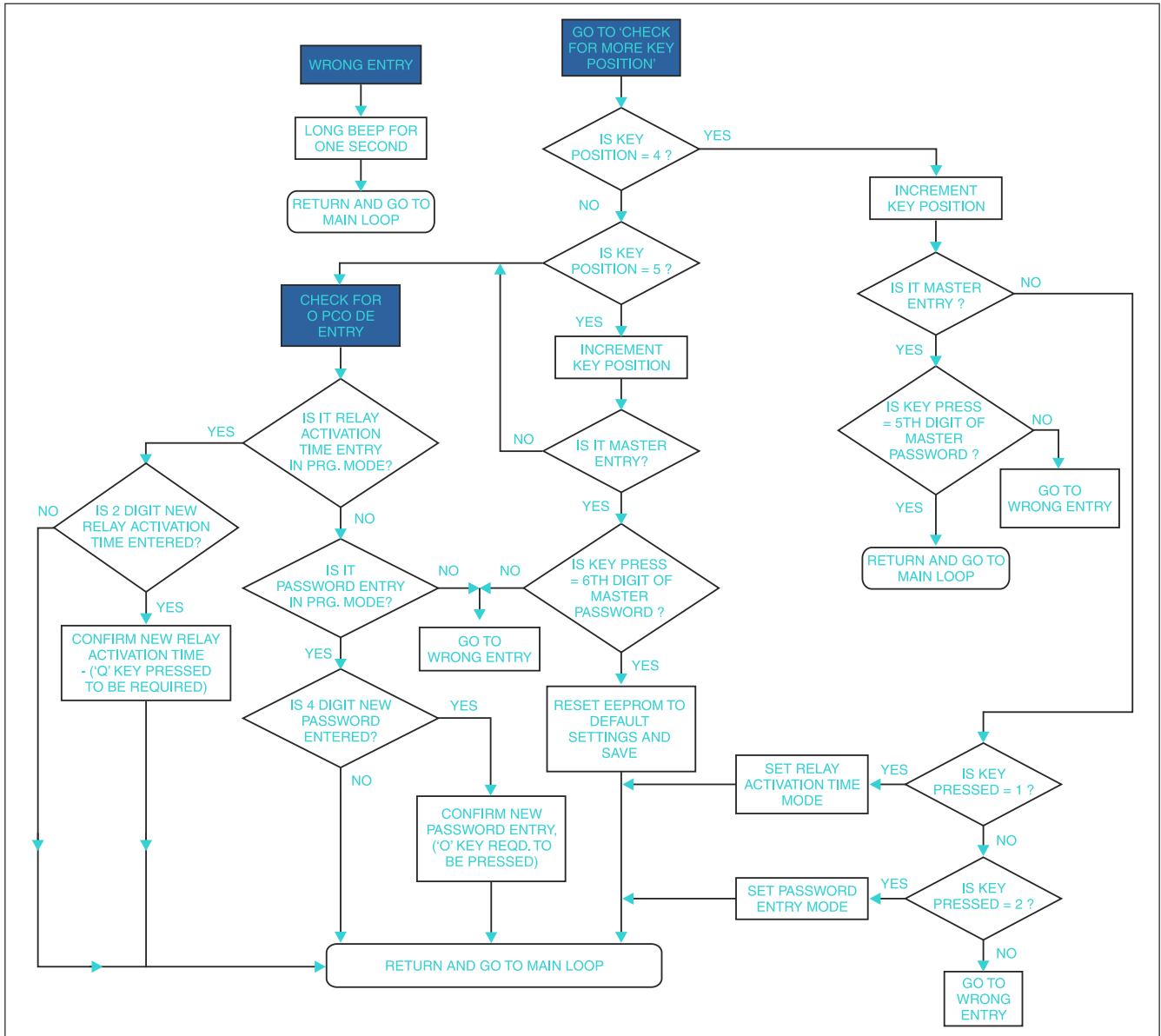


Fig. 9(c): Flow-chart for the access-control system, continued from Fig. 9(b)

Assembly-level program is to be saved in a file with .ASM extension.

3. Assemble the source code, i.e. convert the source code (file with extension .ASM) into object code (machine language) using assembler/compiler tab of environmental setting in WinIDE. The object code will be in S19 format, i.e. the object code file will have extension .S19. You can also choose options within the dialogue box to generate listing file with extension .LST and .MAP file for source-level debugging. Thus if your source program was titled main.asm', you will get main.s19, main.lst, and main.map files after successful assembly.

4. Simulate your program using the WinIDE software, JICS board, and the target board (the PCB with keyboard,

memory, buzzer, etc). JICS board is connected to the computer through serial port (9-pin/25-pin) of the computer. The target board is connected to JICS board through a 16-pin DIP header cable.

During simulation you may find that the program is not behaving properly. Assuming that your hardware is okay, the most probable reason is an error in writing the software. So look for faults in your logic/code and rectify them. You should be able to simulate complete functions without using the actual microcontroller chip.

5. Now, program the microcontroller with the developed and tested software. After programming the microcontroller, insert it into the circuit and check all func-

tions again.

Possible modifications

The circuit can be modified to have more than one password, advanced functions like real-time clock, computer connectivity via serial/parallel port to log data, and interfacing to a bar code reader instead of keypad for opening the lock. These additions may entail using a different microcontroller with more memory and I/O pins, but using essentially the same hardware configuration while writing a fresh program.

Note.The MN1280 is attached to reset pin 9 of the microcontroller. If the MN1280 is not available, you can use only the RC circuit.

MAIN.ASM

```

*****
;;
;;PROJECT   :- ACCESS CONTROL (GENERAL)
;;VERSION   :- 01
;;STARTING DATE :- 09-10-2k day - monday
;;IC        :- KJ1
;;HARDWARE   :-                12
KEYS\1LED\1HOOTER\1MEMORY
;;HARDWARE REC. :- 06-10-2k
;;FEATURES   :- ENTER PASSWORD TO OPEN
DOOR
*****
                org     0c0h
$setnot        testing
$include       "stdj1.asm"
$include       "ports.asm"
$include       "variable.asm"

key_word       equ     14h
key_word1      equ     28h
second_last_kw equ     5h
last_key_word  equ     7h

et_buff        db     2

org            300h

$include       "iic.asm"
$include       "macro.asm"
$include       "readkbd2.asm"

start:        rsp

***** INITIALISE
PORT *****

init_port      ddra      ;; initialise port a
init_port      porta

init_port      ddrb      ;; initialise port b
init_port      portb

***** CLEAR
MEMORY\INITIALISE *****
***** TIMER *****

clear_mem      ;; clear Ram

init_timer     ;; initialise timer

chk_mem        ;; check EEPROM

;; if bad_mem flag = 1 then goto read_defval
;; if bad_mem flag = 0 then read values from
eeprom

brset bad_mem,status,read_defval

;; program comes here when bad_mem flag =
00
;; at power on e_add & mem_ptr = 00
***** READ VALUES
FROM EEPROM *****
;; read 2 byte password/entry time from EEPROM
read_mem_val   clr mem_ptr
               clr e_add
read_next_val: jsr get_eeprom_info ;; read
from eeprom
lda e_dat      ;; save read value in e_dat
ldx mem_ptr    ;; set index reg as pointer
sta password,x ;; save read value in
cmp #0ffh     ;; if value read from EEPROM
is ff then
;; goto read def val
               beq read_defval
inc e_add      ;; increment e_add
inc mem_ptr    ;; increment ptr
lda mem_ptr
cmp #max_iic_bytes ;; is all 3 bytes read
bne read_next_val ;; if no goto read_mem_val
bra main_loop ;; if yes goto main_loop

read_defval:   jsr read_def_val

***** MAIN LOOP *****
;; after every one tick over call sense_kbd
;; after every half second over call chk_set_beep
;; after every second check
kbd_timeout\entry_time_out
main_loop:    brclr
one_tick,tim_status,main_loop
               bclr one_tick,tim_status
               jsr kbd_sense

chk_hs_over   brclr
half_sec,tim_status,chk_1_sec
               bclr half_sec,tim_status
               jsr chk_set_beep

chk_1_sec     brclr
one_sec,tim_status,ret_act1sec
               bclr one_sec,tim_status

;; program comes here after every second over
***** DECREMENT KBD
TIMEOUT *****
a1s_tstkbd   tst kbd_timeout ;; if
timeout = 0 then
beq tst_eto ;; goto check for entry
time
dec kbd_timeout ;; else decrement
kbd time
tst kbd_timeout ;; again chk kbd
timeout
bne tst_eto ;; if # 0 goto tst_eto
jsr wrong_entry ;; give wrong entry
signal

***** DECREMENT EN-
TRY TIME *****
;; check for entry time = 00
tst_eto:     tst entry_time_out ;; if
timeout = 00 then
beq ret_act1sec ;; ret_act1sec
dec entry_time_out ;; else decrement
timeout
tst entry_time_out ;; again chk entry
time
bne ret_act1sec ;; if # zero goto
ret_act1sec
bclr led_arm,led_port ;; else ON led arm

ret_act1sec

***** CHECK FOR KEY *****
;; if new key found flag set then goto act kbd
else goto main_loop
chkbd brclr
new_key_found,status,ret_chkbd ;; if new key
found then set
bclr new_key_found,status ;; flag
jsr act_kbd ;; call actkbd
ret_chkbd jmp main_loop ;;
else goto main loop

***** ACTKBD *****
;; set key press timeout to 10 seconds
act_kbd:     lda #10t ;; set key
press timeout = 10secs
sta kbd_timeout

lda kbd_pos ;; read kbd pos

***** KEY PROGRAM
OK PRESSED *****
act_kbd1:    cmp #k_pgm_ok ;; is
pgm ok key pressed
bne act_kbd2 ;; if no goto act_kbd2
jsr chk_po_status ;; if yes call
chk_po_status
bra ret_actkbd ;; goto ret_actkbd

;; program here checks for
po_password\po_entry_time flag
;; if po_password\po_entry_time flag = 1 and if
some other key press
;; accept pgm_ok_key then goto wrong entry
;; else goto chk_pgm_key
act_kbd2 brclr
po_password,entry_status,chk4poet
jmp wrong_entry

chk4poet:    brclr
po_entry_time,entry_status,chk_pgm_key
jmp wrong_entry

***** KEY PROGRAM
PRESSED *****
chk_pgm_key: cmp #k_program ;;
is pgm_ok key press
bne act_kbd3 ;; if no goto act_kbd3
bset pgm_mode,status ;; if yes set flag of
pgm_mode
clr buff_pointer ;; clear all pointers
clr entry_status ;; clear entry status
clr kbd_timeout
bra ret_actkbd ;; give beep while returning

***** OTHER KEY
PRESSED *****
;; check for password code
;; first chk for buff pointer is buff pointer > 3
if yes then goto is_it_mode
;; else take first digit pressed in kbd_buff,second
digit in kbd_buff+1
;; third digit in kbd_buff+2 & fourth digit in
kbd_buff+3

act_kbd3 ldx buff_pointer ;; is all 4
digit password enters
cpx #3
bhi is_it_mode ;; if yes then goto
is_it_mode
lda kbd_pos ;; else store kbd_pos in
kbd_buff+ptr
sta kbd_buff,x
inc buff_pointer ;; increment pointer
lda buff_pointer ;; is it 4th digit to be
entered
cmp #4 ;; if no then return
bne ret_actkbd

;; program comes here when all 4 keys entered
;; check for valid code
;; if not valid code then give long beep and
clear buff_pointer\kbd_timeout
;; and return
;; else clear sys_arm flag and give accp beep
jsr pack_buff ;; call pack buffer

;; check for 4 key press
;; if it is equals to password then
;; return
;; if it is not equals to password then goto wrong
entry
lda kbd_buff
cmp password
bne chk4master_kw
lda kbd_buff+1
cmp password+1
bne chk4master_kw

;; PROGRAM COMES HERE WHEN 4 DIGIT COR-
RECT PASSWORD IS ENTERED
brset pgm_mode,status,ret_actkbd
bset led_arm,led_port ;; off led
arm
lda entry_time ;; set entry_time_out
sta entry_time_out

```

```

jmp entry_over ; call entry_over

;; here program checks for master key word
;; if key sequence entered is equals to first 4
master key word then
;; e_key_word flag is set
;; else
;; long beep is heard as wrong entry

chk4master_kw:
lda kbd_buff
cmp #key_word ;; 14
bne wrong_entry
lda kbd_buff+1
cmp #key_word1 ;; 28
bne wrong_entry
bset es_key_word,entry_status
bra ret_actkbd

;; program comes here when unit is in program-
ming mode and 4 digit password enters
;; if 4 digit entered # password then goto wrong
entry
;; else return
xxxx: lda kbd_buff ;
compare kbd_buff with
cmp password ; password
bne wrong_entry ; if # goto wrong entry
lda kbd_buff+1 ; if = compare
kbd_buff+1 with
cmp password+1 ; password+1
bne wrong_entry ; if # goto wrong
entry
ret_actkbd jmp quick_beep ; give
small beep after every
ret_actkbd1: rts ; key press
; return

is_it_mode: cpx #04 ; is
buffer pointer = 4
bne chk4parameters ; if # goto
chk4parameters
inc buff_pointer ; else increment
pointer

brclr es_key_word,entry_status,iim1
;; program comes here when key word entry is
checked
;; check is 5th key press = 8 then return
;; else
;; goot wrong key and give long beep
lda kbd_pos
cmp #second_last_kw ;;
next digit is 5
bne wrong_entry
jmp ret_actkbd

iim1:
;; key 1 is for entry time
;; key 2 for password change
lda kbd_pos ; read kbd_pos
cmp #01 ; is key 1 press
bne chk2 ; if # goto chk2
set_entry_time bset es_entry_time,entry_status
; set flag of es_entry_time
bra ret_actkbd ; return

chk2: cmp #02 ; is key 2
press
bne chk3 ; if # goto chk3
set_new_password bset
es_password,entry_status ; else set flag of
es_password
bra ret_actkbd ; return

chk3:
;***** WRONG ENTRY
;*****
wrong_entry jsr long_beep ; give
long beep
jmp entry_over ; goto
entry over

```

```

;; program comes here when buffer pointer is >
4
chk4parameters:
cpx #05 ; if buff_pointer > 5
then
bne more_parameters ; goto
more_parameters
inc buff_pointer ; else increment
pointer
brclr

es_key_word,entry_status,c4p1
lda kbd_pos
cmp #last_key_word ; last
digit for master key word is 7
bne wrong_entry
jmp master_reset_eeprom

c4p1:
;; program comes here when buff_pointer = 6
;; check is it es_entry_time = 1
;; if yes then store key press in last_key_val
;; set flag of po_entry_time
;; return
;; if no then goto chk4es_pw
brclr

es_entry_time,entry_status,chk4es_pw
lda kbd_pos
sta et_buff
jmp ret_actkbd

;; program comes here when buff_pointer = 6
and es_entry_time = 0
;; check es_password flag
;; if flag set then
;; save key press in kbd_buff
;; else goto wrong entry
more_parameters:
brclr

es_entry_time,entry_status,chk4es_pw
bset po_entry_time,entry_status
lda kbd_pos
sta et_buff+1
tst et_buff
bne ret_actkbd
tst et_buff+1
bne ret_actkbd
jmp wrong_entry

chk4es_pw:
brclr
es_password,entry_status,wrong_entry
lda buff_pointer ; subtract
buff_pointer with 6
sub #6
tax ; set subtracted val as pointer
lda kbd_pos ; read kbd_pos
sta kbd_buff,x ; save in kbd_buff+ptr
inc buff_pointer ; increment pointer
lda buff_pointer ; if pointer = 10
cmp #10t ; if no then
return
bne ret_actkbd
bset po_password,entry_status ; else set
po_password flag
bra ret_actkbd ; return

entry_table db 5t,2,4,6,8,10t,12t,14t,16t,18t

;; program comes here when pgm_ok key press
;; chck is po_entry_time flag = 1
;; if yes then
;; set last key press as pointer
;; take corresponding entry time from entry
table
;; and save in entry_time
;; goto com_po_ret
chk_po_status:
brclr
po_entry_time,entry_status,chk4popassword
bclr po_entry_time,entry_status
jsr pack_et_buff
bra com_po_ret

```

```

;; program comes here when po_entry_time =
0
;; program here checks for po_password
;; if po_password = 1 then
;; call pack_buff
;; store change password in password vari-
able
;; store in eeprom
;; call entry_over
;; give acc_beep
;; return
chk4popassword
brclr

po_password,entry_status,chk4more
bclr po_password,entry_status
upd_password jsr pack_buff ; call
pack_buff
lda kbd_buff ; save kbd_buff in
sta password ; password
lda kbd_buff+1 ; save kbd_buff+1 in
sta password+1 ; password+1

com_po_ret jsr store_memory ;
save changed parameter in eeprom
jsr entry_over ; call entry over
jsr acc_beep ; give acceptance
beep
jmp ret_actkbd1 ; return

chk4more bra wrong_entry ;
else give long beep

; SUBROUTINES :-
;*****
;***** ACCEPTANCE
BEEP *****
;; give beep thrice
acc_beep jsr short_beep
jsr short_delay
jsr short_delay
jsr short_beep
jsr short_delay
jsr short_delay
jmp short_beep

;***** ENTRY OVER
*****
;; clear pointer(timeout/entry_status)pgm_mode
flag
entry_over: bclr pgm_mode,status
clr buff_pointer
clr kbd_timeout
clr entry_status
rts

; ***** SHORT DELAY
*****
short_delay lda running_ticks
add #beep_time
sta delay_temp
sd_wait lda delay_temp
cmp running_ticks
bne sd_wait
rts

;***** LONG ENTRY
*****
;; give this beep when wrong entry
;; giva a long beep for around 1 sec
;; stay here till 1 second is over
long_beep lda #ticks_1_sec
sta buzzer_time_out
bclr buzzer,buzzer_port
lb_wait: bsr delay
bsr toggle_buzzer_pin
tst buzzer_time_out
bne lb_wait
bset buzzer,buzzer_port
rts

```

```

***** SHORT BEEP
*****
;; this routine is called from accp_bEEP and when
entry time # 0
;; and after every key press
;; beep for small time
;; set buzzer_time_out = beep_time
;; wait untill buzzer time out # 00
quick_bEEP:
short_bEEP lda #beep_time
           sta buzzer_time_out
           bclr buzzer,buzzer_port
sb_wait:   bsr delay
           bsr toggle_buzzer_pin
           tst buzzer_time_out
           bne sb_wait
           bset buzzer,buzzer_port
           rts

***** TOGGLE BUZZER
PIN *****
;; if buzzer time out # 00 then toggle buzzer pin
toggle_buzzer_pin:
                                           breset
buzzer,buzzer_port,reset_buzzer
           bset buzzer,buzzer_port
           bra ret_tbp
reset_buzzer: bclr buzzer,buzzer_port
ret_tbp:    rts

***** DELAY FOR HALF
MSEC *****
;; this delay is approximately = 499usec
;; 2+4+[(5+4+3)83]= 10998cycles
;; 998/.5 = 499usec = .5msec
delay:    lda #83t
           sta temp
wait_0:   dec temp
           tst temp
           bne wait_0
           rts

***** PACK
BUFFER *****
pack_buff lda kbd_buff
           lsla
           lsla
           lsla
           lsla
           ora kbd_buff+1
           sta kbd_buff
           lda kbd_buff+2
           lsla
           lsla
           lsla
           ora kbd_buff+3
           sta kbd_buff+1
           rts

***** STORE
MEMORY *****
;; store 2byte password in eeprom
store_memory: breset bad_mem,status,ret_sm
              clr e_add ;; clear e_add
              clr mem_ptr ;; clear
mem_ptr
nxt_data:
;; read data from RAM location
;; and store it in memory
ldx mem_ptr ;; set index register as ptr
lda password,x ;; read upper byte of
password
sta e_dat ;; save in e_dat
jsr set_eeeprom_info ;; tx to eeprom
inc e_add ;; increment address
inc mem_ptr ;; increment pointer
lda mem_ptr ;; is all 3 bytes written
cmp #max_iic_bytes ;; if not goto nxt_data
bne nxt_data ;; else return
ret_sm: rts
***** TIMINT
*****
timint: lda #def_timer ;; set
tscr = 14h
sta tscr
bset one_tick,tim_status ;; set flag for
One tick over
inc ticks ;; increment ticks
inc running_ticks
;; if buzzer time out is not zero
;; then decrement buzzer timeout
;; interrupt comes here afetr every 8.2msec

tst buzzer_time_out
beq chk_half_sec
dec buzzer_time_out

chk_half_sec: lda ticks ;;
compare ticks with
cmp #ticks_in_hsec ;; ticks in half sec
bne chk4secover ;; if # goto
chk4secover
bset half_sec,tim_status ;; set flag of
half sec over

chk4secover lda ticks ;;
compare ticks with
cmp #ticks_1_sec ;; ticks in one
second
bne ret_timint ;; if # then return
bset half_sec,tim_status ;; set flag of
half sec
bset one_sec,tim_status ;; set flag of
one sec

; clr running_ticks
; clr ticks ;; clear ticks
dummy:
ret_timint: rti

;; start beep when entry or exit time is not zero
chk_set_bEEP tst entry_time_out
beq ret_csb
jsr short_bEEP
ret_csb rts

;; master key word received
;; if key entered in following sequence then re-
set EEPROM to default settings
;; Key word is 142587
;; default setting is that password entry will
change to 1111
master_reset_eeeprom:
bsr read_def_val
jsr acc_bEEP ; give
acceptance beep
jsr entry_over
bra store_memory

read_def_val clr
rdv_loop: lda def_table,x
           sta password,x
           incx
           cpx #max_iic_bytes
           bne rdv_loop
           rts

;; here program pack entry time from
et_buff+1
;; first byte is in et_buff
;; second byte is in et_buff+1
;; output to entry_time var

;; for decimal selection multiply first number by
10t and then add with next number

pack_et_buff: lda et_buff
              ldx #10t
              mul
              add et_buff+1
              sta entry_time
              rts

***** DEFAULT
TABLE *****
def_table db 11h ; password
change defult password from 1234 to 1111
           db 11h ; password+1
           db 10t ; entry time

org 7cdh
jmp start

org 7f1h
db 20h

org 7f8h
fdb timint

org 7fah
fdb dummy

org 7fch
fdb dummy

org 7feh
fdb start

```

IIC.ASM

```

;; IIC_TX
;; function : transfer 5 bytes from iic_buff to iic
bus
;; input : iic_buff
;; output : to iic
;; variables: rega, regx
;; constants: scl
;; sda
;; iicport
;; iicont

;; input in a register
byte_iic: bset sda,iicont ; set sda
as output port

ldx #8 ; count of 8 bits
bit_iic: rora ; shift msb
to carry
bcc sda_low ; if no carry(msb
low)
sda_high: bset sda,iicport ; carry
set msb high
bra pulse_scl
sda_low: bclr sda,iicport
pulse_scl: bsr delay_small ; delay
bset scl,iicport ; set scl

high
bsr delay_small
bclr scl,iicport ; then scl is set low
; bsr delay_small

decx ; is count over
bne bit_iic ; no next bit
bclr sda,iicont ; leave sda high by
making it input
bsr delay_small
bsr delay_small
bset scl,iicport
bsr delay_small
clc ; normal - clear carry
brclr sda,iicport,byte_over ;error if ackn
not rcvd
sec ; error - set carry
byte_over: bclr scl,iicport ; set scl
low
bsr delay_small

```

```

bsr    delay_small
bclr   sda,iicport ;
rts    ; leave with sda as input

delay_small:  nop
            nop
            nop
            nop
            nop
            rts

set_eeprom_info
iic_tx:
;; generate start condition
;; first set sda then scl then make sda low while
scl is high
;; on return sda is low and scl is low
;; variables : iic_counter,iic_buff(six bytes)

restart_tx:

            bsr    gen_start
            lda    #0a0h
            bsr    byte_iic
            bcs    restart_tx ; restart

if carry set
            lda    e_add
            bsr    byte_iic
            bcs    restart_tx
            lda    e_dat
            bsr    byte_iic
            bcs    restart_tx

;; generate stop condition
;; sda is set as output and low
;; first sda is cleared the scl is set high

;; then make sda high keeping scl high
;; on return scl is high and sda is also high

gen_stop:  bclr   sda,iicport
            bset   sda,iicont ; set sda as
output
            jsr    delay_small
            bset   scl,iicport
            bsr    delay_small
            bset   sda,iicport ; leave with sda
and
            rts    ; scl high and
output

gen_start: bset   sda,iicont ;
sda as o/p
            bset   sda,iicport ; and high
            bsr    delay_small
            bset   scl,iicport ; scl also high
            bsr    delay_small

            bclr   sda,iicport
            bsr    delay_small
            bclr   scl,iicport

get_eeprom_info
;; iic_rx
;; generate start byte
;; transfer address byte with bit 0 set to 1
;; if memory write e_add also
;; read one byte
;; and save in iic_status
;; generate stop byte
;; input : iicbuff (one byte- address of iic)
;; output : iic_status
;; variables : rega,rega

;; constants : scl,sda,iicport,iicont
iic_rx:
restart_rx:

            bsr    gen_start
            lda    #0a0h
            dev_addr: jsr    byte_iic ;
sda is input on return
            bcs    restart_rx
            lda    e_add
            jsr    byte_iic ;
second byte as mem add
            bcs    restart_rx
            bsr    gen_start
            lda    #0a1h
            jsr    byte_iic ; sda is
input on return
            read_iicbyte: ldx    #8
            read_iicbit: bset   scl,iicport ;
set scl high
            ; jsr    delay_small ;
; delay
; bclr   scl,iicport ; and
again low
            brset  sda,iicport,iic_1 ; read data bit
            iic_0   clc
            iic_1   bra    read_iic
            read_iic sec
            read_iic rola
            jsr    delay_small ;
delay
            bclr   scl,iicport ; and
again low
            decx
            bne    read_iicbit
            sta    e_dat
            bra    gen_stop
    
```

STDJ1.ASM

```

porta equ 00h
portb equ 01h
ddra equ 04h
ddrb equ 05h

pdra equ 10h
pdrb equ 11h
tscr equ 08h
tcr equ 09h

iscr equ 0ah
copr equ 7f0h
    
```

VARIABLE.ASM

```

last_key_val db 00
entry_status db 00
es_password equ 1
es_entry_time equ 2
po_entry_time equ 3
po_password equ 4
es_key_word equ 5

temp db 00
active_scan db 00
kbd_temp db 00
delay_temp db 00
running_ticks db 00
mem_ptr db 00

kbd_timeout db 00
buff_pointer db 00
kbd_buff db 00,00,00,00

status db 00
new_key_found equ 7

key_alarm equ 6
bad_mem equ 5
sys_arm equ 4
pgm_mode equ 3

password db 00,00 ; stored in
eeprom
entry_time db 00 ; stored in
eeprom

buzzer_time_out db 00
beep_time equ 10t

entry_time_out db 00
hooter_time equ 2
hooter_alarm_tout db 00

e_add db 00
e_dat db 00
iic_buff db 00

kbd_pos db 00
last_key db 00
same_key db 00

def_timer equ 14h

tim_status db 00
one_tick equ 7
half_sec equ 6
one_sec equ 5
one_min equ 4

mins db 00
ticks_1_sec equ 122t
ticks_in_hsec equ 61t
ticks db 00
max_iic_bytes equ 3

key_scan_cntr db 00
    
```

READKBD2.ASM

```

scan_table: db 0eh,0dh,0bh,07h
key_scan_port equ porta

;; sense2 line is at irq

            kbd_sense
sense_line lda key_port ;read
key port

            and #30h
            bil key_found

            ora #40h
            cmp #70h
            bne key_found ; no some
key pressed
            bra no_key_found ; yes no
    
```

```

key pressed
    key_found sta kbd_temp
    lda key_port
key_port with kbd table
    and #0fh
; remove
unused line
    ora kbd_temp
    clrx

    try_nxt_code cmp kbd_table,x
    beq key_matched ;if equal goto key
matched
    incx ;else increment index
register
    cmpx #max_keys ;compare it with
maximum keys
    bne try_nxt_code ;if not equal goto
try nxt code

    no_key_found ldx #0fh ;
    key_matched txa ;load
accumulator with 'X'
    cmp kbd_pos ;compare it with
kbd pos
    beq ret_kbs ;if equal return
    cmp last_key ;compare it with last
key
    bne new_key ;if equal return
    inc same_key ;else goto new key
& inc same
    lda same_key ;for max debounce
load same key
    cmp #max_debounce ;compare it
with 4
    bne ret_kbs ;if not equal goto ret
kbs
    upd_key lda last_key ;load
last key
    sta kbd_pos ;store it at kbd pos
    cmp #0fh ;is it key release
    beq ret_kbs ;yes-do not set flag

    bset new_key_found,status ;set bit of new
key found in
    bra ret_kbs ;status and goto ret
kbs

    new_key sta last_key
    clr same_key
    bra kbs_over

    ret_kbs lda kbd_pos ;load
kbd pos
    cmp #0fh ;
    bne kbs_over ;

    change_sense inc key_scan_cntr
    lda key_scan_cntr
    cmp #04
    blo cs1
    clr key_scan_cntr

cs1:
    lda key_scan_port
    and #0f0h
    sta key_scan_port ;reset all
scan lines to zero on ports

    ldx key_scan_cntr ;output
scan table to scan port one by one
    lda scan_table,x
    ora key_scan_port
    sta key_scan_port

ret_sense_line
kbs_over rts

max_keys equ 12t

$if testing
max_debounce equ 1
$elseif
max_debounce equ 3t
$endif

;; code1 pin
;;scan0 bit pa0 ;16
;;scan1 bit pa1 ;15
;;scan2 bit pa2 ;14
;;scan3 bit pa3 ;13
;;sense0 bit pa4 ;12
;;sense1 bit pa5 ;11
;;sense2 bit irq

;; code 0 13-irq (pa3-pa5)
;; code 1 16-12 (pa0-pa4)
;; code 2 16-11 (pa0-pa5)

;; code 3 16-irq (pa0-irq)
;; code 4 15-12 (pa1-pa4)
;; code 5 15-11 (pa1-pa5)

;; code 6 15-irq (pa1-irq)
;; code 7 14-12 (pa2-pa4)
;; code 8 14-11 (pa2-pa5)

;; code 9 14-irq (pa2-irq)
;; code 10 13-12 (pa3-pa4) ; key
program

;; code 12 13-11 (pa3-irq) ; key
program ok

kbd_table db 057h ;; code for 00
db 06eh ;; code for 01
db 05eh ;; code for 02
db 03eh ;; code for 03
db 06dh ;; code for 04
db 05dh ;; code for 05
db 03dh ;; code for 06
db 06bh ;; code for 07
db 05bh ;; code for 08
db 03bh ;; code for 09
db 067h ;; code for pgm
key
db 037h ;; code for pgm
ok key

```

```

k_program equ 10t
k_pgm_ok equ 11t

scl equ 2
sda equ 3
iicport equ portb
iicont equ ddrb
;; 7 6 5 4 3 2 1 0
def_ddra equ 0cfh ;; hoot led sen1
sen0 scan3 scan2 scan1 scan0
def_porta equ 080h ;; active low hooter

and led
;; at power on system armed led
def_ddrb equ 0ch ;; x x x x
sda scl x x
def_portb equ 00

key_port equ porta

scan0 equ 0 ;16
scan1 equ 1 ;15
scan2 equ 2 ;14

scan3 equ 3 ;13
sense0 equ 4 ;12
sense1 equ 5 ;11
;;sense2 equ irq ;irq

led_port equ porta
led_arm equ 6
toggle_led equ 40h

buzzer_port equ porta
buzzer equ 7

```

MACRO.ASM

```

$macro chk_mem
    bclr bad_mem,status ;; clear flag
bad_mem
    jsr gen_start ;; call gen_start
    lda #0a0h ;; send device add =
0a0h
    jsr byte_iic ;; to memory
    bcc cm_over ;; of carry clear then
return
    bset bad_mem,status ;; if carry set then
set flag
    cm_over ;; bad mem
$macroend

;; clear memory from 0c0h

```

```

$macro clear_mem
    ldx #0c0h ;;clear memory
    next_mm clr ,x
    incx
    bne next_mm
$macroend

;; initialise timer
$macro init_timer
    lda #def_timer
    sta tscr
    cli ;enable interrupt
$macroend
;; initialise porta , portb

```

```

$macro init_port port
    lda #def_%1
    sta %1
$macroend

```

EFY noteAll relevant files will be included in Nov. '02 EFY-CD.

* This project is a result of teamwork by Dimpi Thukral, Harnam Singh, Ruchi Sharma, and Satish Pal Singh led by Vinay Chaddha, proprietor GVC Systems, Noida.