```
 1:  ;-------------------------------------------------------
 2:  ; Get string (of maximum length 80) from keyboard.
 3:  ;     AX <-- pointer to a buffer to store the input string
 4:  ;     CX <-- buffer size = string length + 1 for NULL
 5:  ; If CX < 2,  CX := 2 is used to read at least one character.
 6:  ; If CX > 81, CX := 81 is used to read at most 80 characters.
 7:  ;-------------------------------------------------------
 8:  proc_GetStr  PROC
 9:              push        DX      ; save registers
10:              push        SI
11:              push        DI
12:              push        ES
13:              mov         DX,DS   ; set up ES to point to DS
14:              mov         ES,DX   ;  for string instruction use
15:              mov         DI,AX   ; DI := buffer pointer
16:              ; check CX bounds
17:              cmp         CX,2
18:              jl          set_CX_2
19:              cmp         CX,81
20:              jle         read_str
21:              mov         CX,81
22:              jmp         SHORT read_str
23:  set_CX_2:
24:              mov         CX,2
25:  read_str:
```

```
25:  read_str:
26:              ; use temporary buffer str_buffer to read the string
27:              ;  in using function 0AH of int 21H
28:              mov         DX,OFFSET str_buffer
29:              mov         SI,DX
30:              mov         [SI],CL  ; first byte = # of chars. to read
31:              DOScall     0AH
32:              inc         SI       ; second byte = # of chars. read
33:              mov         CL,[SI] ; CX := # of bytes to copy
34:              inc         SI       ; SI = input string first char.
35:              cld                  ; forward direction for copy
36:              rep         movsb
37:              mov         BYTE PTR [DI],0   ; append NULL character
38:              pop         ES       ; restore registers
39:              pop         DI
40:              pop         SI
41:              pop         DX
42:              ret
43:  proc_GetStr  ENDP
```

```
 1:  COMMENT |       A string read program       FUNNYSTR.ASM
 2:          Objective: To demonstrate the use of BIOS keyboard
 3:                     functions 0, 1, and 2.
 4:              Input: Prompts for a string
 5:  |          Output: Displays the input string and its length
 6:
 7:  STR_LENGTH  EQU  81
 8:  .MODEL SMALL
 9:  .STACK   100H
10:  .DATA
11:  string      DB  STR_LENGTH DUP (?)
12:  prompt_msg  DB  'Please enter a string (< 81 chars): ',0
13:  string_msg  DB  'The string entered is ',0
14:  length_msg  DB  ' with a length of ',0
15:  end_msg     DB  ' characters.',0
16:
17:  .CODE
18:  INCLUDE  io.mac
19:  main    PROC
20:          .STARTUP
21:          PutStr  prompt_msg
22:          mov     AX,STR_LENGTH-1
23:          push    AX                ; push max. string length
24:          mov     AX,OFFSET string
25:          push    AX                ; and string pointer parameters
26:          call    read_string       ; to call read_string procedure
```

```
27:           nwln
28:           PutStr  string_msg
29:           PutStr  string
30:           PutStr  length_msg
31:           PutInt  AX
32:           PutStr  end_msg
33:           nwln
34:           .EXIT
35:   main    ENDP
36:   ;---------------------------------------------------------
37:   ; String read procedure using BIOS int 16H. Receives string
38:   ; pointer and the length via the stack. Length of the string
39:   ; is returned in AX.
40:   ;---------------------------------------------------------
41:   read_string     PROC
42:           push    BP
43:           mov     BP,SP
44:           push    BX
45:           push    CX
46:           mov     CX,[BP+6]    ; CX := length
47:           mov     BX,[BP+4]    ; BX := string pointer
```

```
48:  read_loop:
49:          mov     AH,2          ; read keyboard status
50:          int     16H           ; status returned in AL
51:          and     AL,3          ; mask off most significant 6 bits
52:          cmp     AL,3          ; if equal both shift keys depressed
53:          jz      end_read
54:          mov     AH,1          ; otherwise, see if a key has been
55:          int     16H           ; struck
56:          jnz     read_key      ; if so, read the key
57:          jmp     read_loop
58:  read_key:
59:          mov     AH,0          ; read the next key from keyboard
60:          int     16H           ; key returned in AL
61:          mov     [BX],AL       ; copy to buffer and increment
62:          inc     BX            ;  buffer pointer
63:          PutCh   AL            ; display the character
64:          loop    read_loop
65:  end_read:
66:          mov     BYTE PTR[BX],0  ; append NULL
67:          sub     BX,[BP+4]     ; find the input string length
68:          mov     AX,BX         ; return string length in AX
69:          pop     CX
70:          pop     BX
71:          pop     BP
72:          ret     4
73:  read_string     ENDP
74:          END     main
```

```
 1:  TITLE   Single-step program      STEPINTR.ASM
 2:  COMMENT |
 3:          Objective: To demonstrate how ISRs can be defined
 4:                     and installed.
 5:              Input: None
 6:             Output: Displays AX and BX values for
 7:  |                  the single-step code
 8:
 9:  .MODEL SMALL
10:  .STACK 100H
11:  .DATA
12:  old_offset  DW  ?    ; for old ISR offset
13:  old_seg     DW  ?    ;   and segment values
14:  start_msg   DB  'Starts single stepping process.',0
15:  AXequ       DB  'AX = ',0
16:  BXequ       DB  ' BX = ',0
17:
18:  .CODE
19:  INCLUDE io.mac
20:
21:  main    PROC
22:          .STARTUP
23:          PutStr  start_msg
24:          nwln
25:
```

```
26:         ; get current interrupt vector for int 1H
27:         mov     AX,3501H        ; AH := 35H and AL := 01H
28:         int     21H             ; returns the offset in BX
29:         mov     old_offset,BX   ;   and the segment in ES
30:         mov     old_seg,ES
31:
32:         ;set up interrupt vector to our ISR
33:         push    DS              ; DS is used by function 25H
34:         mov     AX,CS           ; copy current segment to DS
35:         mov     DS,AX
36:         mov     DX,OFFSET sstep_ISR  ; ISR offset in DX
37:         mov     AX,2501H        ; AH := 25H and AL := 1H
38:         int     21H
39:         pop     DS              ; restore DS
40:
41:         ; set trap flag to start single stepping
42:         pushf
43:         pop     AX              ; copy flags into AX
44:         or      AX,100H         ; set trap flag bit (TF = 1)
45:         push    AX              ; copy modified flag bits
46:         popf                    ;   back to flags register
47:
```

```
48:           ; from now on int 1 is generated after executing
49:           ;  each instruction. Some test instructions follow.
50:           mov     AX,100
51:           mov     BX,20
52:           add     AX,BX
53:
54:           ; clear trap flag to end single stepping
55:           pushf
56:           pop     AX              ; copy flags into AX
57:           and     AX,0FEFFH       ; clear trap flag bit (TF = 0)
58:           push    AX              ; copy modified flag bits
59:           popf                    ;   back to flags register
60:
61:           ; restore the original ISR
62:           mov     DX,old_offset
63:           push    DS
64:           mov     AX,old_seg
65:           mov     DS,AX
66:           mov     AX,2501H
67:           int     21H
68:           pop     DS
69:
70:           .EXIT
71:   main    ENDP
```

```
72:  ;-------------------------------------------------------
73:  ;Single-step interrupt service routine replaces int 01H.
74:  ;-------------------------------------------------------
75:  sstep_ISR  PROC
76:          sti                     ; enable interrupt
77:          PutStr  AXequ           ; display AX contents
78:          PutInt  AX
79:          PutStr  BXequ           ; display BX contents
80:          PutInt  BX
81:          nwln
82:          iret
83:  sstep_ISR  ENDP
84:          END   main
```

```
 1:  ;-------------------------------------------------------
 2:  ; Sends CR and LF to the screen. Uses display function 2
 3:  ;-------------------------------------------------------
 4:  proc_nwln  PROC
 5:             push     DX
 6:             mov      DL,0DH      ; carraige return
 7:             DOScall  2
 8:             mov      DL,0AH      ; line feed
 9:             DOScall  2
10:             pop    DX
11:             ret
12:  proc_nwln  ENDP
```

```
 1:  TITLE   Keyboard interrupt service program   KEYBOARD.ASM
 2:  COMMENT |
 3:          Objective: To demonstrate how the keyboard works.
 4:              Input: Key strokes from the keyboard. Only left
 5:                     and right shift keys are recognized.
 6:                     ESC key restores the original keyboard ISR
 7:                     and terminates the program.
 8:  |           Output: Displays the key on the screen.
 9:
10:  ESC_KEY      EQU  1BH   ; ASCII code for ESC key
11:  CR           EQU  0DH   ; ASCII code for carriage return
12:  KB_DATA      EQU  60H   ; 8255 port PA
13:  KB_CTRL      EQU  61H   ; 8255 port PB
14:  LEFT_SHIFT   EQU  2AH   ; left shift scan code
15:  RIGHT_SHIFT  EQU  36H   ; right shift scan code
16:  EOI          EQU  20H   ; end-of-interrupt byte for 8259 PIC
17:  PIC_CMD_PORT EQU  20H   ; 8259 PIC command port
18:
19:  .MODEL SMALL
20:  .STACK 100H
21:  .DATA
22:  install_msg    DB  'New keyboard ISR installed.',0
23:  keyboard_data  DB  -1   ; keyboard buffer
24:  keyboard_flag  DB  0    ; keyboard shift status
25:  old_offset     DW  ?    ; storage for old int 09H vector
26:  old_segment    DW  ?
```

```
27:  ; lowercase scan code to ASCII conversion table.
28:  ; ASCII code 0 is used for scan codes we are not interested.
29:  lcase_table    DB  01BH,'1234567890-=',08H,09H
30:                DB  'qwertyuiop[]',CR,0
31:                DB  'asdfghjkl;',27H,60H,0,'\'
32:                DB  'zxcvbnm,./',0,'*',0,' ',0
33:                DB  0,0,0,0,0,0,0,0,0,0
34:                DB  0,0,0,0,0,0,0,0,0,0
35:                DB  0,0,0,0,0,0,0,0,0,0
36:  ; uppercase scan code to ASCII conversion table.
37:  ucase_table    DB  01BH,'!@#$%^&*()_+',08H,09H
38:                DB  'QWERTYUIOP{}',0DH,0
39:                DB  'ASDFGHJKL:','"','~',0,'|'
40:                DB  'ZXCVBNM<>?',0,'*',0,' '
41:                DB  0,0,0,0,0,0,0,0,0,0
42:                DB  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
43:  .CODE
44:  INCLUDE io.mac
45:
46:  main    PROC
47:          .STARTUP
48:          PutStr  install_msg
49:          nwln
50:
51:          ; save int 09H vector for later restoration
52:          mov     AX,3509H     ; AH := 35H and AL := 09H
53:          int     21H          ; DOS function 35H returns
54:          mov     old_offset,BX   ; offset in BX and
55:          mov     old_segment,ES  ; segment in ES
```

```
57:             ;set up interrupt vector to our keyboard ISR
58:             push    DS              ; DS is used by function 25H
59:             mov     AX,CS           ; copy current segment to DS
60:             mov     DS,AX
61:             mov     DX,OFFSET kbrd_ISR  ; ISR offset in DX
62:             mov     AX,2509H        ; AH := 25H and AL := 09H
63:             int     21H
64:             pop     DS              ; restore DS
65:
66:     repeat:
67:             call    read_kb_key     ; read a key
68:             cmp     AL,ESC_KEY      ; if ESC key
69:             je      done            ; then done
70:             cmp     AL,CR           ; if carriage return
71:             je      newline         ; then display new line
72:             PutCh   AL              ; else display character
73:             jmp     repeat
74:     newline:
75:             nwln
76:             jmp     repeat
77:     done:
78:             ; restore original keyboard interrupt int 09H vector
79:             mov     DX,old_offset
80:             push    DS
81:             mov     AX,old_segment
82:             mov     DS,AX
83:             mov     AX,2509H
84:             int     21H
85:             pop     DS
86:
87:             .EXIT
88:     main    ENDP
```

```
 89:  ;-------------------------------------------------------
 90:  ;This procedure waits until a valid key is entered at the
 91:  ; keyboard. The ASCII value of the key is returned in AL.
 92:  ;-------------------------------------------------------
 93:  read_kb_key  PROC
 94:          cmp     keyboard_data,-1  ; -1 is an invalid entry
 95:          je      read_kb_key
 96:          mov     AL,keyboard_data
 97:          mov     keyboard_data,-1
 98:          ret
 99:  read_kb_key  ENDP
100:  ;-------------------------------------------------------
101:  ;This keyboard ISR replaces the original int 09H ISR.
102:  ;-------------------------------------------------------
103:  kbrd_ISR  PROC
104:          sti                     ; enable interrupt
105:          push    AX              ; save registers used by ISR
106:          push    BX
107:          in      AL,KB_DATA      ; read keyboard scan code and the
108:          mov     BL,AL           ;  key status (down or released)
```

```
109:          ; send keyboard acknowledge signal by momentarily
110:          ;  setting and clearing PB7 bit
111:          in      AL,KB_CTRL
112:          mov     AH,AL
113:          or      AL,80H
114:          out     KB_CTRL,AL   ; set PB7 bit
115:          xchg    AL,AH
116:          out     KB_CTRL,AL   ; clear PB7 bit
117:
118:          mov     AL,BL        ; AL := scan code + key status
119:          and     BL,7FH       ; isolate scan code
120:          cmp     BL,LEFT_SHIFT   ; left or right shift key
121:          je      left_shift_key  ;  changed status?
122:          cmp     BL,RIGHT_SHIFT
123:          je      right_shift_key
124:          test    AL,80H          ; if not, check status bit
125:          jnz     EOI_to_8259     ; if key released, do nothing
126:          mov     AH,keyboard_flag ; AH := shift key status
127:          and     AH,1         ; AH = 1 if left/right shift is ON
128:          jnz     shift_key_on
129:          ; no shift key is pressed
130:          mov     BX,OFFSET lcase_table ; shift OFF, use lowercase
131:          jmp     SHORT get_ASCII       ;  conversion table
132:  shift_key_on:
133:          mov     BX,OFFSET ucase_table ; shift key ON, use uppercase
```

```
134:  get_ASCII:                              ;  conversion table
135:          dec     AL              ; index is one less than scan code
136:          xlat
137:          cmp     AL,0            ; ASCII code of 0 => uninterested key
138:          je      EOI_to_8259
139:          mov     keyboard_data,AL  ; save ASCII code in keyboard bu
140:          jmp     SHORT EOI_to_8259
141:
142:  left_shift_key:
143:  right_shift_key:
144:          test    AL,80H          ; test key status bit (0=down, 1=up)
145:          jnz     shift_off
146:  shift_on:
147:          or      keyboard_flag,1     ; shift bit (i.e., LSB) := 1
148:          jmp     SHORT EOI_to_8259
149:  shift_off:
150:          and     keyboard_flag,0FEH  ; shift bit (i.e., LSB) := 0
151:          jmp     SHORT EOI_to_8259
152:
153:  EOI_to_8259:
154:          mov     AL,EOI              ; send EOI to 8259 PIC
155:          out     PIC_CMD_PORT,AL     ; indicating end of ISR
156:          pop     BX                  ; restore registers
157:          pop     AX
158:          iret
159:  kbrd_ISR  ENDP
160:          END   main
```