```
;-------------------------------------------------
;PutInt8 procedure displays a signed 8-bit integer
;that is in AL. All registers are preserved.
;-------------------------------------------------
PutInt8 PROC
    push    BP
    mov     BP,SP
    sub     SP,3             ; local buffer space
    push    AX
    push    BX
    push    SI
    test    AL,80H           ; negative number?
    jz      positive
negative:
    PutCh   '-'              ; sign for -ve numbers
    neg     AL               ; convert to magnitude
```

```
positive:
    mov     BL,10   ; divisor  = 10
    sub     SI,SI   ; SI:=0(SI points to buffer)
repeat:
    sub     AH,AH   ; AH:=0(AX is the dividend)
    div     BL
    ; AX/BL leaves AL:= quotient & AH := remainder
    add     AH,'0'  ; convert remainder to ASCII
    mov     [BP+SI-3],AH ; copy into the buffer
    inc     SI
    cmp     AL,0    ; quotient = zero?
    jne     repeat  ; if so, display the number
display_digit:
    dec     SI
    mov     AL,[BP+SI-3]
                    ;display digit pointed by SI
    PutCh   AL
    jnz     display_digit
                    ;if SI<0, done displaying
```

```
display_done:
    pop     SI    ;restore registers
    pop     BX
    pop     AX
    mov     SP,BP ;clear local variable space
    pop     BP
    ret
PutInt8 ENDP
```

```
;----------------------------------------------
;GetInt8 procedure reads an integer from the
;keyboard and stores its equivalent binary in AL.
;If the number is within -128 and +127
;(both inclusive), CF is cleared; otherwise,
;CF is set to indicate out-of-range error.
;No error check is done to see if the input
;consists of digits only. All registers are
;preserved except for AX.
;-----------------------------------------------
CR    EQU   0DH

GetInt8 PROC
    push    BX           ; save registers
    push    CX
    push    DX
    sub     DX,DX        ; DX := 0
    sub     BX,BX        ; BX := 0
```

```
get_next_char:
    GetCh   DL        ; read input from keyboard
    cmp     DL,'-'    ; is it negative sign?
    je      sign      ; if so, save the sign
    cmp     DL,'+'    ; is it positive sign?
    jne     digit     ; if not, process the digit
sign:
    mov     BH,DL   ; BH keeps sign of input number
    jmp     get_next_char
digit:
    sub     AX,AX   ; AX := 0
    mov     BL,10   ; BL holds the multiplier
    sub     DL,'0'  ; convert ASCII to numeric
    mov     AL,DL
    mov     CX,2    ; maximum 2 more digits to read
```

```
convert_loop:
    GetCh   DL
    cmp     DL,CR        ; carraige return?
    je      convert_done
                    ;if so, done reading the number
    sub     DL,'0'  ;else, convert ASCII to numeric
    mul     BL      ; multiply total (in AL) by 10
    add     AX,DX   ; and add the current digit
    loop    convert_loop
convert_done:
    cmp     AX,128
    ja      out_of_range
                 ; if AX > 128, number out of range
    jb      number_OK
                 ; if AX < 128, number is valid
    cmp     BH,'-'
                 ; AX = 128. Must be a negative;
    jne     out_of_range
                 ; otherwise, an invalid number
```

```
number_OK:
    cmp     BH,'-'    ; number negative?
    jne     number_done
                 ; if not, we are done
    neg     AL   ; else, convert to 2's complement
number_done:
    clc               ; CF := 0 (no error)
    jmp     done
out_of_range:
    stc               ; CF := 1 (range error)
done:
    pop     DX        ; restore registers
    pop     CX
    pop     BX
    ret
GetInt8 ENDP
```

```
;---------------------------------------------------
;Multiplies two 64-bit unsigned numbers A and B.
;A is received in EBX:EAX and B in EDX:ECX.
;The 128-bit result is returned in EDX:ECX:EBX:EAX.
;This procedure uses longhand multiplication.
;Preserves all registers except EAX,EBX,ECX,and EDX.
;---------------------------------------------------
COUNT   EQU   WORD PTR [BP-2]   ; local variable

mult64   PROC
   push    BP
   mov     BP,SP
   sub     SP,2              ; local variable
   push    ESI
   push    EDI
   mov     ESI,EDX           ; SI:DI := B
   mov     EDI,ECX
   sub     EDX,EDX           ; P := 0
   sub     ECX,ECX
   mov     COUNT,64 ; count = 64 (64-bit number)
```

```
step:
   test    AX,1         ; LSB of A is 1?
   jz      shift1       ; if not, skip add
   add     ECX,EDI      ; Otherwise, P := P+B
   adc     EDX,ESI
shift1:                 ; shift right P and A
   rcr     EDX,1
   rcr     ECX,1
   rcr     EBX,1
   rcr     EAX,1

   dec     COUNT        ; if COUNT is not zero
   jnz     step         ;  repeat the process
   ; restore registers
   pop     EDI
   pop     ESI
   mov     SP,BP   ; clear local variable space
   pop     BP
   ret
mult64   ENDP
```

```
;---------------------------------------------------
;Multiplies two 64-bit unsigned numbers A and B.
;A is received in EBX:EAX and B in EDX:ECX.
;The 64-bit result is returned in EDX:ECX:EBX:EAX.
;Uses mul instruction to multiply 32-bit numbers.
;Preserves all registers except EAX,EBX,ECX,and EDX.
;----------------------------------------------------
; local variables
RESULT3  EQU  DWORD PTR [BP-4]
            ; most significant 32 bits of result
RESULT2  EQU  DWORD PTR [BP-8]
RESULT1  EQU  DWORD PTR [BP-12]
RESULT0  EQU  DWORD PTR [BP-16]
            ; least significant 32 bits of result

```

```
mult64w   PROC
   push    BP
   mov     BP,SP
   sub     SP,16 ;local variables for the result
   push    ESI
   push    EDI
   mov     EDI,EAX         ; ESI:EDI := A
   mov     ESI,EBX
   mov     EBX,EDX         ; EBX:ECX := B
   ; multiply A0 and B0
   mov     EAX,ECX
   mul     EDI
   mov     RESULT0,EAX
   mov     RESULT1,EDX
   ; multiply A1 and B0
   mov     EAX,ECX
   mul     ESI
   add     RESULT1,EAX
   adc     EDX,0
   mov     RESULT2,EDX
```

```
        sub     EAX,EAX   ; store 1 in RESULT3 if
        rcl     EAX,1     ; a carry was generated
        mov     RESULT3,EAX
        ; multiply A0 and B1
        mov     EAX,EBX
        mul     EDI
        add     RESULT1,EAX
        adc     RESULT2,EDX
        adc     RESULT3,0
        ; multiply A1 and B1
        mov     EAX,EBX
        mul     ESI
        add     RESULT2,EAX
        adc     RESULT3,EDX
```

```
   ; copy result to the registers
   mov     EAX,RESULT0
   mov     EBX,RESULT1
   mov     ECX,RESULT2
   mov     EDX,RESULT3
   ; restore registers
   pop     EDI
   pop     ESI
   mov     SP,BP    ; clear local variable space
   pop     BP
   ret
mult64w   ENDP
```

```
;--------------------------------------------------
;Divides two 64-bit unsigned numbers A and B (A/B).
;A is received in EBX:EAX and B in EDX:ECX.
;The 64-bit quotient is returned in EBX:EAX and
;the remainder in EDX:ECX.
;Divide by zero error is indicated by setting
;the carry flag; CF is cleared otherwise.
;Preserves all registers except EAX,EBX,ECX,and EDX.
;---------------------------------------------------
; local variables
SIGN       EQU  BYTE PTR [BP-1]
BIT_COUNT  EQU  BYTE PTR [BP-2]
div64  PROC
   push    BP
   mov     BP,SP
   sub     SP,2           ; local variable space
   push    ESI
   push    EDI
   ; check for zero divisor in DX:CX
   cmp     ECX,0
   jne     non_zero
   cmp     EDX,0
   jne     non_zero
   stc                 ; if zero, set carry flag to
   jmp     SHORT skip  ; indicate error and return
non_zero:
   mov     ESI,EDX     ; SI:DI := B
   mov     EDI,ECX
   sub     EDX,EDX     ; P := 0
   sub     ECX,ECX
   mov     SIGN,0
   mov     BIT_COUNT,64  ; BIT_COUNT := # of bits
next_pass: ; *** main loop iterates 64 times ***
   test    SIGN,1        ; if P is positive
   jz      P_positive    ; jump to P_positive
```

```
P_negative:
   rcl     EAX,1          ; right shift P and A
   rcl     EBX,1
   rcl     ECX,1
   rcl     EDX,1
   rcl     SIGN,1
   add     ECX,EDI        ; P := P + B
   adc     EDX,ESI
   adc     SIGN,0
   jmp     test_sign
P_positive:
   rcl     EAX,1          ; right shift P and A
   rcl     EBX,1
   rcl     ECX,1
   rcl     EDX,1
   rcl     SIGN,1
   sub     ECX,EDI        ; P := P + B
   sbb     EDX,ESI
   sbb     SIGN,0
```

```
test_sign:
   test    SIGN,1      ; if P is negative
   jnz     bit0        ; set lower bit of A to 0
bit1:                  ; else, set it to 1
   or      AL,1
   jmp     one_pass_done ;set lower bit of A to 0
bit0:
   and     AL,0FEH     ; set lower bit of A to 1
   jmp     one_pass_done
one_pass_done:
   dec     BIT_COUNT   ; iterate for 32 times
   jnz     next_pass
```

```
div_done:               ; division completed
   test    SIGN,1       ; if P is positive
   jz      div_wrap_up  ; we are done
   add     ECX,EDI      ; otherwise, P := P + B
   adc     EDX,ESI
div_wrap_up:
   clc          ; clear carry to indicate no error
skip:
   pop     EDI       ; restore registers
   pop     ESI
   mov     SP,BP     ; clear local variable space
   pop     BP
   ret
div64  ENDP
```