```
 1:  ;-----------------------------------------------
 2:  ;PutInt8 procedure displays a signed 8-bit integer
 3:  ;that is in AL. All registers are preserved.
 4:  ;-----------------------------------------------
 5:  PutInt8 PROC
 6:      push    BP
 7:      mov     BP,SP
 8:      sub     SP,3          ; local buffer space
 9:      push    AX
10:      push    BX
11:      push    SI
12:      test    AL,80H        ; negative number?
13:      jz      positive
14:  negative:
15:      PutCh   '-'           ; sign for -ve numbers
16:      neg     AL            ; convert to magnitude
```

Arithmetic: 1

```
17:  positive:
18:      mov     BL,10   ; divisor  = 10
19:      sub     SI,SI   ; SI:=0(SI points to buffer)
20:  repeat:
21:      sub     AH,AH   ; AH:=0(AX is the dividend)
22:      div     BL
23:      ; AX/BL leaves AL:= quotient & AH := remainder
24:      add     AH,'0'  ; convert remainder to ASCII
25:      mov     [BP+SI-3],AH ; copy into the buffer
26:      inc     SI
27:      cmp     AL,0    ; quotient = zero?
28:      jne     repeat  ; if so, display the number
29:  display_digit:
30:      dec     SI
31:      mov     AL,[BP+SI-3]
                         ;display digit pointed by SI
32:      PutCh   AL
33:      jnz     display_digit
                         ;if SI<0, done displaying
```

Arithmetic: 2

```
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
```

```
17:  get_next_char:
18:      GetCh   DL        ; read input from keyboard
19:      cmp     DL,'-'    ; is it negative sign?
20:      je      sign      ; if so, save the sign
21:      cmp     DL,'+'    ; is it positive sign?
22:      jne     digit     ; if not, process the digit
23:  sign:
24:      mov     BH,DL   ; BH keeps sign of input number
25:      jmp     get_next_char
26:  digit:
27:      sub     AX,AX   ; AX := 0
28:      mov     BL,10   ; BL holds the multiplier
29:      sub     DL,'0'  ; convert ASCII to numeric
30:      mov     AL,DL
31:      mov     CX,2    ; maximum 2 more digits to read
```



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

```
46:  number_OK:
47:      cmp     BH,'-'     ; number negative?
48:      jne     number_done
                      ; if not, we are done
49:      neg     AL    ; else, convert to 2's complement
50:  number_done:
51:      clc               ; CF := 0 (no error)
52:      jmp     done
53:  out_of_range:
54:      stc               ; CF := 1 (range error)
55:  done:
56:      pop     DX        ; restore registers
57:      pop     CX
58:      pop     BX
59:      ret
60:  GetInt8 ENDP
```

Arithmetic: 7

```
 1:  ;-------------------------------------------------
 2:  ;Multiplies two 64-bit unsigned numbers A and B.
 3:  ;A is received in EBX:EAX and B in EDX:ECX.
 4:  ;The 128-bit result is returned in EDX:ECX:EBX:EAX.
 5:  ;This procedure uses longhand multiplication.
 6:  ;Preserves all registers except EAX,EBX,ECX,and EDX.
 7:  ;-------------------------------------------------
 8:  COUNT   EQU   WORD PTR [BP-2]   ; local variable
 9:
10:  mult64   PROC
11:     push    BP
12:     mov     BP,SP
13:     sub     SP,2          ; local variable
14:     push    ESI
15:     push    EDI
16:     mov     ESI,EDX       ; SI:DI := B
17:     mov     EDI,ECX
18:     sub     EDX,EDX       ; P := 0
19:     sub     ECX,ECX
20:     mov     COUNT,64 ; count = 64 (64-bit number)
```

Arithmetic: 8

```
 21:  step:
 22:     test    AX,1          ; LSB of A is 1?
 23:     jz      shift1        ; if not, skip add
 24:     add     ECX,EDI       ; Otherwise, P := P+B
 25:     adc     EDX,ESI
 26:  shift1:                  ; shift right P and A
 27:     rcr     EDX,1
 28:     rcr     ECX,1
 29:     rcr     EBX,1
 30:     rcr     EAX,1
 31:
 32:     dec     COUNT         ; if COUNT is not zero
 33:     jnz     step          ;  repeat the process
 34:     ; restore registers
 35:     pop     EDI
 36:     pop     ESI
 37:     mov     SP,BP   ; clear local variable space
 38:     pop     BP
 39:     ret
 40:  mult64   ENDP
```



```
 1:  ;-------------------------------------------------
 2:  ;Multiplies two 64-bit unsigned numbers A and B.
 3:  ;A is received in EBX:EAX and B in EDX:ECX.
 4:  ;The 64-bit result is returned in EDX:ECX:EBX:EAX.
 5:  ;Uses mul instruction to multiply 32-bit numbers.
 6:  ;Preserves all registers except EAX,EBX,ECX,and EDX.
 7:  ;-------------------------------------------------
 8:  ; local variables
 9:  RESULT3  EQU  DWORD PTR [BP-4]
                 ; most significant 32 bits of result
10:  RESULT2  EQU  DWORD PTR [BP-8]
11:  RESULT1  EQU  DWORD PTR [BP-12]
12:  RESULT0  EQU  DWORD PTR [BP-16]
                 ; least significant 32 bits of result
13:
```

```
14:  mult64w   PROC
15:     push    BP
16:     mov     BP,SP
17:     sub     SP,16 ;local variables for the result
18:     push    ESI
19:     push    EDI
20:     mov     EDI,EAX        ; ESI:EDI := A
21:     mov     ESI,EBX
22:     mov     EBX,EDX        ; EBX:ECX := B
23:     ; multiply A0 and B0
24:     mov     EAX,ECX
25:     mul     EDI
26:     mov     RESULT0,EAX
27:     mov     RESULT1,EDX
28:     ; multiply A1 and B0
29:     mov     EAX,ECX
30:     mul     ESI
31:     add     RESULT1,EAX
32:     adc     EDX,0
33:     mov     RESULT2,EDX
```



```
34:     sub     EAX,EAX   ; store 1 in RESULT3 if
35:     rcl     EAX,1     ; a carry was generated
36:     mov     RESULT3,EAX
37:     ; multiply A0 and B1
38:     mov     EAX,EBX
39:     mul     EDI
40:     add     RESULT1,EAX
41:     adc     RESULT2,EDX
42:     adc     RESULT3,0
43:     ; multiply A1 and B1
44:     mov     EAX,EBX
45:     mul     ESI
46:     add     RESULT2,EAX
47:     adc     RESULT3,EDX
```

```
 48:     ; copy result to the registers
 49:     mov     EAX,RESULT0
 50:     mov     EBX,RESULT1
 51:     mov     ECX,RESULT2
 52:     mov     EDX,RESULT3
 53:     ; restore registers
 54:     pop     EDI
 55:     pop     ESI
 56:     mov     SP,BP    ; clear local variable space
 57:     pop     BP
 58:     ret
 59:  mult64w   ENDP
```

Arithmetic: 13

```
 1:  ;-------------------------------------------------
 2:  ;Divides two 64-bit unsigned numbers A and B (A/B).
 3:  ;A is received in EBX:EAX and B in EDX:ECX.
 4:  ;The 64-bit quotient is returned in EBX:EAX and
 5:  ;the remainder in EDX:ECX.
 6:  ;Divide by zero error is indicated by setting
 7:  ;the carry flag; CF is cleared otherwise.
 8:  ;Preserves all registers except EAX,EBX,ECX,and EDX.
 9:  ;-------------------------------------------------
10:  ; local variables
11:  SIGN       EQU  BYTE PTR [BP-1]
12:  BIT_COUNT  EQU  BYTE PTR [BP-2]
13:  div64  PROC
14:     push    BP
15:     mov     BP,SP
16:     sub     SP,2           ; local variable space
17:     push    ESI
18:     push    EDI
```

Arithmetic: 14

```
19:     ; check for zero divisor in DX:CX
20:     cmp     ECX,0
21:     jne     non_zero
22:     cmp     EDX,0
23:     jne     non_zero
24:     stc                 ; if zero, set carry flag to
25:     jmp     SHORT skip  ; indicate error and return
26:  non_zero:
27:     mov     ESI,EDX     ; SI:DI := B
28:     mov     EDI,ECX
29:     sub     EDX,EDX     ; P := 0
30:     sub     ECX,ECX
31:     mov     SIGN,0
32:     mov     BIT_COUNT,64  ; BIT_COUNT := # of bits
33:  next_pass: ; *** main loop iterates 64 times ***
34:     test    SIGN,1        ; if P is positive
35:     jz      P_positive    ; jump to P_positive
```



```
36:  P_negative:
37:     rcl     EAX,1       ; right shift P and A
38:     rcl     EBX,1
39:     rcl     ECX,1
40:     rcl     EDX,1
41:     rcl     SIGN,1
42:     add     ECX,EDI     ; P := P + B
43:     adc     EDX,ESI
44:     adc     SIGN,0
45:     jmp     test_sign
46:  P_positive:
47:     rcl     EAX,1       ; right shift P and A
48:     rcl     EBX,1
49:     rcl     ECX,1
50:     rcl     EDX,1
51:     rcl     SIGN,1
52:     sub     ECX,EDI     ; P := P + B
53:     sbb     EDX,ESI
54:     sbb     SIGN,0
```

```
55:  test_sign:
56:     test    SIGN,1      ; if P is negative
57:     jnz     bit0        ; set lower bit of A to 0
58:  bit1:                  ; else, set it to 1
59:     or      AL,1
60:     jmp     one_pass_done ;set lower bit of A to 0
61:  bit0:
62:     and     AL,0FEH     ; set lower bit of A to 1
63:     jmp     one_pass_done
64:  one_pass_done:
65:     dec     BIT_COUNT   ; iterate for 32 times
66:     jnz     next_pass
```



```
67:  div_done:              ; division completed
68:     test    SIGN,1      ; if P is positive
69:     jz      div_wrap_up  ; we are done
70:     add     ECX,EDI     ; otherwise, P := P + B
71:     adc     EDX,ESI
72:  div_wrap_up:
73:     clc        ; clear carry to indicate no error
74:  skip:
75:     pop     EDI       ; restore registers
76:     pop     ESI
77:     mov     SP,BP     ; clear local variable space
78:     pop     BP
79:     ret
80:  div64  ENDP
```