

```

1:  TITLE    Parameter passing via registers      PROCEX1.ASM
2:  COMMENT  |
3:           Objective: To show parameter passing via registers
4:           Input: Requests two integers from the user.
5:  |           Output: Outputs the sum of the input integers.
6:  .MODEL  SMALL
7:  .STACK  100H
8:  .DATA
9:  prompt_msg1  DB    'Please input the first number: ',0
10: prompt_msg2  DB    'Please input the second number: ',0
11: sum_msg      DB    'The sum is ',0
12:
13:  .CODE
14:  INCLUDE io.mac
15:
16:  main  PROC
17:      .STARTUP
18:      PutStr  prompt_msg1      ; request first number
19:      GetInt  CX                ; CX := first number
20:      nwln
21:      PutStr  prompt_msg2      ; request second number
22:      GetInt  DX                ; DX := second number
23:      nwln

```

```

24:         call    sum            ; returns sum in AX
25:         PutStr  sum_msg        ; display sum
26:         PutInt  AX
27:         nwnln
28: done:
29:         .EXIT
30: main ENDP
31:
32: ;-----
33: ;Procedure sum receives two integers in CX and DX.
34: ; The sum of the two integers is returned in AX.
35: ;-----
36: sum PROC
37:     mov     AX,CX            ; sum := first number
38:     add    AX,DX            ; sum := sum + second number
39:     ret
40: sum ENDP
41: END main

```

```

1:  TITLE    Parameter passing via registers          PROCEX2.ASM
2:  COMMENT  |
3:           Objective: To show parameter passing via registers
4:           Input: Requests a character string from the user.
5:  |       Output: Outputs the length of the input string.
6:
7:  BUF_LEN    EQU    41                ; string buffer length
8:  .MODEL    SMALL
9:  .STACK    100H
10: .DATA
11: string      DB    BUF_LEN DUP (?)    ;input string < BUF_LEN chars.
12: prompt_msg  DB    'Please input a string: ',0
13: length_msg  DB    'The string length is ',0
14:
15: .CODE
16: INCLUDE io.mac
17:
18: main  PROC
19:     .STARTUP
20:     PutStr  prompt_msg          ; request string input
21:     GetStr  string,BUF_LEN      ; read string from keyboard
22:     nwnl
23:     mov     BX,OFFSET string    ; BX := string address
24:     call    str_len             ; returns string length in AX

```

```

25:          PutStr  length_msg      ; display string length
26:          PutInt  AX
27:          nwnln
28:  done:
29:          .EXIT
30:  main  ENDP
31:
32:  ;-----
33:  ;Procedure str_len receives a pointer to a string in BX.
34:  ; String length is returned in AX.
35:  ;-----
36:  str_len PROC
37:          push    BX
38:          sub     AX,AX              ; string length := 0
39:  repeat:
40:          cmp     BYTE PTR [BX],0  ; compare with NULL char.
41:          je     str_len_done      ; if NULL we are done
42:          inc     AX                ; else, increment string length
43:          inc     BX                ; point BX to the next char.
44:          jmp     repeat           ; and repeat the process
45:  str_len_done:
46:          pop     BX
47:          ret
48:  str_len ENDP
49:          END      main

```

```

1:  TITLE    Parameter passing via the stack          PROCEX3.ASM
2:  COMMENT  |
3:           Objective: To show parameter passing via the stack
4:           Input: Requests two integers from the user.
5:  |           Output: Outputs the sum of the input integers.
6:  .MODEL  SMALL
7:  .STACK  100H
8:  .DATA
9:  prompt_msg1  DB    'Please input the first number: ',0
10: prompt_msg2  DB    'Please input the second number: ',0
11: sum_msg      DB    'The sum is ',0
12:
13:  .CODE
14:  INCLUDE io.mac
15:
16:  main  PROC
17:      .STARTUP
18:      PutStr  prompt_msg1      ; request first number
19:      GetInt  CX                ; CX := first number
20:      nwln
21:      PutStr  prompt_msg2      ; request second number
22:      GetInt  DX                ; DX := second number
23:      nwln

```

```

24:         push    CX                ; place first number on stack
25:         push    DX                ; place second number on stack
26:         call    sum                ; returns sum in AX
27:         PutStr  sum_msg           ; display sum
28:         PutInt  AX
29:         nwnln
30:  done:
31:         .EXIT
32:  main  ENDP
33:
34:  ;-----
35:  ;Procedure sum receives two integers via the stack.
36:  ; The sum of the two integers is returned in AX.
37:  ;-----
38:  sum   PROC
39:         push    BP                ; we will use BP, so save it
40:         mov     BP,SP
41:         mov     AX,[BP+6]         ; sum := first number
42:         add     AX,[BP+4]         ; sum := sum + second number
43:         pop     BP                ; restore BP
44:         ret     4                ; return and clear parameters
45:  sum   ENDP
46:         END      main

```

```

1:  TITLE    Parameter passing via the stack          PROC_SWAP.ASM
2:  COMMENT  |
3:           Objective: To show parameter passing via the stack
4:           Input:  Requests a character string from the user.
5:           Output: Outputs the input string with the first
6:  |                two characters swapped.
7:
8:  BUF_LEN    EQU    41                ; string buffer length
9:  .MODEL    SMALL
10: .STACK    100H
11: .DATA
12: string      DB    BUF_LEN DUP (?)    ;input string < BUF_LEN chars.
13: prompt_msg  DB    'Please input a string: ',0
14: output_msg  DB    'The swapped string is: ',0
15:
16: .CODE
17: INCLUDE io.mac
18:

```

```

19:  main  PROC
20:      .STARTUP
21:      PutStr  prompt_msg      ; request string input
22:      GetStr  string,BUF_LEN  ; read string from the user
23:      nwnln
24:      mov     AX,OFFSET string ; AX := string[0] pointer
25:      push   AX                ; push string[0] pointer on stack
26:      inc    AX                ; AX := string[1] pointer
27:      push   AX                ; push string[1] pointer on stack
28:      call   swap              ; swaps the first two characters
29:      PutStr  output_msg      ; display the swapped string
30:      PutStr  string
31:      nwnln
32:  done:
33:      .EXIT
34:  main  ENDP
35:

```



```

36: ;-----
37: ;Procedure swap receives two pointers (via the stack) to
38: ; characters of a string. It exchanges these two characters.
39: ;-----
40: swap PROC
41:     push    BP                ; save BP - procedure uses BP
42:     mov     BP,SP            ; copy SP to BP
43:     push    BX                ; save BX - procedure uses BX
44:     ; swap begins here. Because of xchg, AL is preserved.
45:     mov     BX,[BP+6]        ; BX := first character pointer
46:     xchg   AL,[BX]
47:     mov     BX,[BP+4]        ; BX := second character pointer
48:     xchg   AL,[BX]
49:     mov     BX,[BP+6]        ; BX := first character pointer
50:     xchg   AL,[BX]
51:     ; swap ends here
52:     pop     BX                ; restore registers
53:     pop     BP
54:     ret     4                ; return and clear parameters
55: swap ENDP
56:     END     main

```

```

1:  COMMENT |           Bubble sort procedure           BBSORT.ASM
2:  Objective: To implement the bubble sort algorithm
3:  Input: A set of non-zero integers to be sorted.
4:  Input is terminated by entering zero.
5:  |           Output: Outputs the numbers in ascending order.
6:  CRLF      EQU    0DH,0AH
7:  MAX_SIZE  EQU    20
8:  .MODEL SMALL
9:  .STACK 100H
10: .DATA
11: array      DW    MAX_SIZE DUP (?) ; input array for integers
12: prompt_msg DB   'Enter non-zero integers to be sorted.',CRLF
13:           DB   'Enter zero to terminate the input.',0
14: output_msg DB   'Input numbers in ascending order:',0
15:
16: .CODE
17: .486
18: INCLUDE io.mac
19: main PROC
20:   .STARTUP
21:   PutStr prompt_msg ; request input numbers
22:   nwln
23:   mov     BX,OFFSET array ; BX := array pointer
24:   mov     CX,MAX_SIZE ; CX := array size
25:   sub     DX,DX ; number count := 0

```

```

26:  read_loop:
27:      GetInt  AX          ; read input number
28:      nwnln
29:      cmp    AX,0        ; if the number is zero
30:      je     stop_reading ; no more numbers to read
31:      mov    [BX],AX     ; copy the number into array
32:      add    BX,2        ; BX points to the next element
33:      inc    DX          ; increment number count
34:      loop   read_loop   ; reads a max. of MAX_SIZE numbers
35:  stop_reading:
36:      push   DX          ; push array size onto stack
37:      push   OFFSET array ; place array pointer on stack
38:      call   bubble_sort
39:      PutStr output_msg   ; display sorted input numbers
40:      nwnln
41:      mov    BX,OFFSET array
42:      mov    CX,DX       ; CX := number count
43:  print_loop:
44:      PutInt [BX]
45:      nwnln
46:      add    BX,2
47:      loop   print_loop
48:  done:
49:      .EXIT
50:  main  ENDP

```

```

51:  ;-----
52:  ;This procedure receives a pointer to an array of integers
53:  ; and the size of the array via the stack. It sorts the
54:  ; array in ascending order using the bubble sort algorithm.
55:  ;-----
56:  SORTED      EQU      0
57:  UNSORTED    EQU      1
58:  bubble_sort  PROC
59:      pusha
60:      mov      BP,SP
61:
62:      ;CX serves the same purpose as the end_index variable
63:      ; in the C procedure. CX keeps the number of comparisons
64:      ; to be done in each pass. Note that CX is decremented
65:      ; by 1 after each pass.
66:      mov      CX, [BP+20] ; load array size into CX
67:      mov      BX, [BP+18] ; load array address into BX
68:
69:  next_pass:
70:      dec      CX          ; if # of comparisons is zero
71:      jz       sort_done   ; then we are done
72:      mov      DI,CX       ; else start another pass
73:
74:      ;DX is used to keep SORTED/UNSORTED status
75:      mov      DX,SORTED   ; set status to SORTED
76:

```

```

77:          ;SI points to element X and SI+2 to the next element
78:      mov     SI,BX          ; load array address into SI
79:  pass:
80:          ;This loop represents one pass of the algorithm.
81:          ;Each iteration compares elements at [SI] and [SI+2]
82:          ; and swaps them if ([SI]) < ([SI+2]).
83:      mov     AX,[SI]
84:      cmp     AX,[SI+2]
85:      jg      swap
86:  increment:
87:          ;Increment SI by 2 to point to the next element
88:      add     SI,2
89:      dec     DI
90:      jnz     pass
91:
92:      cmp     DX,SORTED      ; if status remains SORTED
93:      je      sort_done     ; then sorting is done
94:      jmp     next_pass     ; else initiate another pass
95:

```

```

96:  swap:
97:      ; swap elements at [SI] and [SI+2]
98:      xchg    AX,[SI+2]
99:      mov     [SI],AX
100:     mov     DX,UNSORTED      ; set status to UNSORTED
101:     jmp     increment
102:
103:  sort_done:
104:     popa
105:     ret     4                ; return and clear parameters
106:  bubble_sort    ENDP
107:     END     main

```

```

1:  TITLE    Variable # of parameters passed via stack    VARPARA.ASM
2:  COMMENT |
3:          Objective: To show how variable number of parameters
4:                  can be passed via the stack
5:          Input: Requests variable number of non-zero integers.
6:          A zero terminates the input.
7:  |          Output: Outputs the sum of input numbers.
8:  CRLF     EQU    0DH,0AH      ; carriage return and line feed
9:  .MODEL  SMALL
10: .STACK  100H
11: .DATA
12: prompt_msg  DB  'Please input a set of non-zero integers.',CRLF
13:             DB  'You must enter at least one integer.',CRLF
14:             DB  'Enter zero to terminate the input.',0
15: sum_msg     DB  'The sum of the input numbers is: ',0
16:
17: .CODE
18: INCLUDE io.mac
19:
20: main  PROC
21:     .STARTUP
22:     PutStr  prompt_msg      ; request input numbers
23:     nwln
24:     sub     CX,CX           ; CX keeps number count

```

```

25:  read_number:
26:      GetInt  AX          ; read input number
27:      nwnln
28:      cmp    AX,0        ; if the number is zero
29:      je     stop_reading ; no more numbers to read
30:      push   AX          ; place the number on stack
31:      inc    CX          ; increment number count
32:      jmp    read_number
33:  stop_reading:
34:      push   CX          ; place number count on stack
35:      call  variable_sum ; returns sum in AX
36:      ; clear parameter space on the stack
37:      inc    CX          ; increment CX to include count
38:      add    CX,CX       ; CX := CX * 2 (space in bytes)
39:      add    SP,CX       ; update SP to clear parameter
40:      ; space on the stack
41:      PutStr sum_msg     ; display the sum
42:      PutInt AX
43:      nwnln
44:  done:
45:      .EXIT
46:  main  ENDP
47:

```



```

48: ;-----
49: ;This procedure receives variable number of integers via the
50: ; stack. The last parameter pushed on the stack should be
51: ; the number of integers to be added. Sum is returned in AX.
52: ;-----
53: variable_sum PROC
54:     push    BP                ; save BP - procedure uses BP
55:     mov     BP,SP            ; copy SP to BP
56:     push    BX                ; save BX and CX
57:     push    CX
58:
59:     mov     CX,[BP+4]        ; CX := # of integers to be added
60:     mov     BX,BP
61:     add     BX,6              ; BX := pointer to first number
62:     sub     AX,AX            ; sum := 0
63: add_loop:
64:     add     AX,SS:[BX]       ; sum := sum + next number
65:     add     BX,2              ; BX points to the next integer
66:     loop   add_loop         ; repeat count in CX
67:
68:     pop     CX                ; restore registers
69:     pop     BX
70:     pop     BP
71:     ret                    ; parameter space cleared by main
72: variable_sum ENDP
73:     END      main

```

```

1:  TITLE    Fibonacci numbers (register version)      PROCFIB1.ASM
2:  COMMENT  |
3:           Objective: To compute Fibonacci number using registers
4:           for local variables.
5:           Input: Requests a positive integer from the user.
6:           Output: Outputs the largest Fibonacci number that
7:  |         is less than or equal to the input number.
8:
9:  .MODEL SMALL
10: .STACK 100H
11: .DATA
12: prompt_msg    DB  'Please input a positive number (>1): ',0
13: output_msg1   DB  'The largest Fibonacci number less than '
14:               DB  'or equal to ',0
15: output_msg2   DB  ' is ',0
16:
17: .CODE
18: INCLUDE io.mac
19:

```

```
20:  main  PROC
21:      .STARTUP
22:      PutStr  prompt_msg      ; request input number
23:      GetInt  DX              ; DX := input number
24:      nwnln
25:      call    fibonacci
26:      PutStr  output_msg1     ; display Fibonacci number
27:      PutInt  DX
28:      PutStr  output_msg2
29:      PutInt  AX
30:      nwnln
31:  done:
32:      .EXIT
33:  main  ENDP
34:
```

```

35:  ;-----
36:  ;Procedure fibonacci receives an integer in DX and computes
37:  ; the largest Fibonacci number that is less than or equal to
38:  ; the input number. The Fibonacci number is returned in AX.
39:  ;-----
40:  fibonacci PROC
41:      push    BX
42:      ; AX maintains the smaller of the last two Fibonacci
43:      ; numbers computed; BX maintains the larger one.
44:      mov     AX,1           ; initialize AX and BX to
45:      mov     BX,AX         ; first two Fibonacci numbers
46:  fib_loop:
47:      add     AX,BX         ; compute next Fibonacci number
48:      xchg   AX,BX         ; maintain the required order
49:      cmp    BX,DX         ; compare with input number in DX
50:      jle   fib_loop      ; if not greater, find next number
51:      ; AX contains the required Fibonacci number
52:      pop    BX
53:      ret
54:  fibonacci ENDP

```

```

1:  TITLE    Fibonacci numbers (stack version)      PROCFIB2.ASM
2:  COMMENT  |
3:           Objective: To compute Fibonacci number using the stack
4:           for local variables.
5:           Input: Requests a positive integer from the user.
6:           Output: Outputs the largest Fibonacci number that
7:           | is less than or equal to the input number.
8:  .MODEL  SMALL
9:  .STACK  100H
10: .DATA
11: prompt_msg  DB  'Please input a positive number (>1): ',0
12: output_msg1 DB  'The largest Fibonacci number less than '
13:             DB  'or equal to ',0
14: output_msg2 DB  ' is ',0
15:
16: .CODE
17: INCLUDE io.mac
18:

```

```

19:  main  PROC
20:      .STARTUP
21:      PutStr  prompt_msg      ; request input number
22:      GetInt  DX              ; DX := input number
23:      nwnln
24:      call   fibonacci
25:      PutStr  output_msg1     ; print Fibonacci number
26:      PutInt  DX
27:      PutStr  output_msg2
28:      PutInt  AX
29:      nwnln
30:  done:
31:      .EXIT
32:  main  ENDP
33:
34:  ;-----
35:  ;Procedure fibonacci receives an integer in DX and computes
36:  ; the largest Fibonacci number that is less than the input
37:  ; number. The Fibonacci number is returned in AX.
38:  ;-----
39:  FIB_LO    EQU    WORD PTR [BP-2]
40:  FIB_HI    EQU    WORD PTR [BP-4]

```

```

41: fibonacci PROC
42:     push    BP
43:     mov     BP,SP
44:     sub     SP,4           ; space for local variables
45:     push    BX
46:     ; FIB_LO maintains the smaller of the last two Fibonacci
47:     ; numbers computed; FIB_HI maintains the larger one.
48:     mov     FIB_LO,1      ; initialize FIB_LO and FIB_HI to
49:     mov     FIB_HI,1      ; first two Fibonacci numbers
50: fib_loop:
51:     mov     AX,FIB_HI     ; compute next Fibonacci number
52:     mov     BX,FIB_LO
53:     add     BX,AX
54:     mov     FIB_LO,AX
55:     mov     FIB_HI,BX
56:     cmp     BX,DX         ; compare with input number in DX
57:     jle    fib_loop      ; if not greater, find next number
58:     ; AX contains the required Fibonacci number
59:     pop     BX
60:     mov     SP,BP        ; clear local variable space
61:     pop     BP
62:     ret
63: fibonacci ENDP
64:     END     main

```

```
1:  TITLE    Multimodule program for string length    MODULE1.ASM
2:  COMMENT  |
3:           Objective: To show parameter passing via registers
4:           Input: Requests two integers from keyboard.
5:  |           Output: Outputs the sum of the input integers.
6:  BUF_SIZE EQU 41    ; string buffer size
7:  .MODEL SMALL
8:  .STACK 100H
9:  .DATA
10: prompt_msg  DB    'Please input a string: ',0
11: length_msg  DB    'String length is: ',0
12: string1     DB    BUF_SIZE DUP (?)
13:
14:  .CODE
15:  INCLUDE io.mac
```



```

16:  EXTRN    string_length:PROC
17:  main  PROC
18:      .STARTUP
19:      PutStr  prompt_msg      ; request a string
20:      GetStr  string1,BUF_SIZE ; read string input
21:      nwnln
22:      mov     BX,OFFSET string1 ; BX := string pointer
23:      call    string_length    ; returns string length in AX
24:      PutStr  length_msg      ; display string length
25:      PutInt  AX
26:      nwnln
27:  done:
28:      .EXIT
29:  main  ENDP
30:      END      main

```

```

1:  TITLE           String length procedure           MODULE2.ASM
2:  COMMENT |
3:           Objective: To write a procedure to compute string
4:           length of a NULL terminated string.
5:           Input: String pointer in BX register.
6:  |           Output: Returns string length in AX.
7:  .MODEL SMALL
8:  .CODE
9:  PUBLIC string_length
10: string_length PROC
11:           ; all registers except AX are preserved
12:           push     SI                ; save SI
13:           mov      SI,BX             ; SI := string pointer
14:  repeat:
15:           cmp      BYTE PTR [SI],0   ; is it NULL?
16:           je       done              ; if so, done
17:           inc      SI                ; else, move to next character
18:           jmp      repeat            ; and repeat
19:  done:
20:           sub      SI,BX             ; compute string length
21:           mov      AX,SI             ; return string length in AX
22:           pop      SI                ; restore SI
23:           ret
24:  string_length ENDP
25:           END

```