```
TITLE   Parameter passing via registers    PROCEX1.ASM
COMMENT |
        Objective: To show parameter passing via registers
            Input: Requests two integers from the user.
|          Output: Outputs the sum of the input integers.
.MODEL SMALL
.STACK 100H
.DATA
prompt_msg1  DB   'Please input the first number: ',0
prompt_msg2  DB   'Please input the second number: ',0
sum_msg      DB   'The sum is ',0

.CODE
INCLUDE io.mac

main  PROC
      .STARTUP
      PutStr  prompt_msg1    ; request first number
      GetInt  CX             ; CX := first number
      nwln
      PutStr  prompt_msg2    ; request second number
      GetInt  DX             ; DX := second number
      nwln
      call    sum            ; returns sum in AX
      PutStr  sum_msg        ; display sum
      PutInt  AX
      nwln
done:
      .EXIT
main  ENDP

;--------------------------------------------------
;Procedure sum receives two integers in CX and DX.
; The sum of the two integers is returned in AX.
;--------------------------------------------------
sum   PROC
      mov     AX,CX          ; sum := first number
      add     AX,DX          ; sum := sum + second number
      ret
sum   ENDP
      END     main
```

```
TITLE   Parameter passing via registers      PROCEX2.ASM
COMMENT |
        Objective: To show parameter passing via registers
            Input: Requests a character string from the user.
|          Output: Outputs the length of the input string.

BUF_LEN     EQU  41          ; string buffer length
.MODEL SMALL
.STACK 100H
.DATA
string      DB  BUF_LEN DUP (?)  ;input string < BUF_LEN chars.
prompt_msg  DB  'Please input a string: ',0
length_msg  DB  'The string length is ',0

.CODE
INCLUDE io.mac

main  PROC
      .STARTUP
      PutStr  prompt_msg     ; request string input
      GetStr  string,BUF_LEN ; read string from keyboard
      nwln
      mov     BX,OFFSET string  ; BX := string address
      call    str_len        ; returns string length in AX
      PutStr  length_msg     ; display string length
      PutInt  AX
      nwln
done:
      .EXIT
main  ENDP

;-------------------------------------------------------------
;Procedure str_len receives a pointer to a string in BX.
; String length is returned in AX.
;-------------------------------------------------------------
str_len PROC
      push    BX
      sub     AX,AX          ; string length := 0
repeat:
      cmp     BYTE PTR [BX],0 ; compare with NULL char.
      je      str_len_done   ; if NULL we are done
      inc     AX             ; else, increment string length
      inc     BX             ; point BX to the next char.
      jmp     repeat         ;  and repeat the process
str_len_done:
      pop     BX
      ret
str_len ENDP
      END     main
```

```
TITLE   Parameter passing via the stack      PROCEX3.ASM
COMMENT |
        Objective: To show parameter passing via the stack
            Input: Requests two integers from the user.
|          Output: Outputs the sum of the input integers.
.MODEL SMALL
.STACK 100H
.DATA
prompt_msg1  DB   'Please input the first number: ',0
prompt_msg2  DB   'Please input the second number: ',0
sum_msg      DB   'The sum is ',0

.CODE
INCLUDE io.mac

main  PROC
      .STARTUP
      PutStr  prompt_msg1    ; request first number
      GetInt  CX             ; CX := first number
      nwln
      PutStr  prompt_msg2    ; request second number
      GetInt  DX             ; DX := second number
      nwln
      push    CX             ; place first number on stack
      push    DX             ; place second number on stack
      call    sum            ; returns sum in AX
      PutStr  sum_msg        ; display sum
      PutInt  AX
      nwln
done:
      .EXIT
main  ENDP

;------------------------------------------------------------
;Procedure sum receives two integers via the stack.
; The sum of the two integers is returned in AX.
;------------------------------------------------------------
sum   PROC
      push    BP             ; we will use BP, so save it
      mov     BP,SP
      mov     AX,[BP+6]      ; sum := first number
      add     AX,[BP+4]      ; sum := sum + second number
      pop     BP             ; restore BP
      ret     4              ; return and clear parameters
sum   ENDP
      END     main
```

```
TITLE   Parameter passing via the stack      PROCSWAP.ASM
COMMENT |
        Objective: To show parameter passing via the stack
            Input: Requests a character string from the user.
           Output: Outputs the input string with the first
|                  two characters swapped.

BUF_LEN     EQU  41          ; string buffer length
.MODEL SMALL
.STACK 100H
.DATA
string      DB  BUF_LEN DUP (?)  ;input string < BUF_LEN chars.
prompt_msg  DB  'Please input a string: ',0
output_msg  DB  'The swapped string is: ',0

.CODE
INCLUDE io.mac

```

```
main  PROC
      .STARTUP
      PutStr  prompt_msg     ; request string input
      GetStr  string,BUF_LEN ; read string from the user
      nwln
      mov     AX,OFFSET string  ; AX := string[0] pointer
      push    AX             ; push string[0] pointer on stack
      inc     AX             ; AX := string[1] pointer
      push    AX             ; push string[1] pointer on stack
      call    swap           ; swaps the first two characters
      PutStr  output_msg     ; display the swapped string
      PutStr  string
      nwln
done:
      .EXIT
main  ENDP

```

```
;------------------------------------------------------------
;Procedure swap receives two pointers (via the stack) to
; characters of a string. It exchanges these two characters.
;------------------------------------------------------------
swap  PROC
      push    BP               ; save BP - procedure uses BP
      mov     BP,SP            ; copy SP to BP
      push    BX               ; save BX - procedure uses BX
      ; swap begins here. Because of xchg, AL is preserved.
      mov     BX,[BP+6]        ; BX := first character pointer
      xchg    AL,[BX]
      mov     BX,[BP+4]        ; BX := second character pointer
      xchg    AL,[BX]
      mov     BX,[BP+6]        ; BX := first character pointer
      xchg    AL,[BX]
      ; swap ends here
      pop     BX               ; restore registers
      pop     BP
      ret     4                ; return and clear parameters
swap  ENDP
      END     main
```

```
COMMENT |          Bubble sort procedure     BBLSORT.ASM
  Objective: To implement the bubble sort algorithm
      Input: A set of non-zero integers to be sorted.
             Input is terminated by entering zero.
|          Output: Outputs the numbers in ascending order.
CRLF       EQU   0DH,0AH
MAX_SIZE   EQU   20
.MODEL SMALL
.STACK 100H
.DATA
array       DW  MAX_SIZE DUP (?)  ; input array for integers
prompt_msg  DB  'Enter non-zero integers to be sorted.',CRLF
      DB  'Enter zero to terminate the input.',0
output_msg  DB  'Input numbers in ascending order:',0

.CODE
.486
INCLUDE   io.mac
main  PROC
      .STARTUP
      PutStr  prompt_msg     ; request input numbers
      nwln
      mov     BX,OFFSET array  ; BX := array pointer
      mov     CX,MAX_SIZE    ; CX := array size
      sub     DX,DX          ; number count := 0
```

```
read_loop:
      GetInt  AX              ; read input number
      nwln
      cmp     AX,0            ; if the number is zero
      je      stop_reading    ; no more numbers to read
      mov     [BX],AX         ; copy the number into array
      add     BX,2            ; BX points to the next element
      inc     DX              ; increment number count
      loop    read_loop       ; reads a max. of MAX_SIZE numbers
stop_reading:
      push    DX              ; push array size onto stack
      push    OFFSET array    ; place array pointer on stack
      call    bubble_sort
      PutStr  output_msg      ; display sorted input numbers
      nwln
      mov     BX,OFFSET array
      mov     CX,DX           ; CX := number count
print_loop:
      PutInt  [BX]
      nwln
      add     BX,2
      loop    print_loop
done:
      .EXIT
main  ENDP
```

```
;----------------------------------------------------------
;This procedure receives a pointer to an array of integers
; and the size of the array via the stack. It sorts the
; array in ascending order using the bubble sort algorithm.
;----------------------------------------------------------
SORTED    EQU   0
UNSORTED  EQU   1
bubble_sort     PROC
      pusha
      mov     BP,SP

      ;CX serves the same purpose as the end_index variable
      ; in the C procedure. CX keeps the number of comparisons
      ; to be done in each pass. Note that CX is decremented
      ; by 1 after each pass.
      mov     CX, [BP+20]  ; load array size into CX
      mov     BX, [BP+18]  ; load array address into BX

next_pass:
      dec     CX           ; if # of comparisons is zero
      jz      sort_done         ; then we are done
      mov     DI,CX        ; else start another pass

      ;DX is used to keep SORTED/UNSORTED status
      mov     DX,SORTED    ; set status to SORTED
      ;SI points to element X and SI+2 to the next element
      mov     SI,BX        ; load array address into SI
pass:
      ;This loop represents one pass of the algorithm.
      ;Each iteration compares elements at [SI] and [SI+2]
      ; and swaps them if ([SI]) < ([SI+2]).
      mov     AX,[SI]
      cmp     AX,[SI+2]
      jg      swap
increment:
      ;Increment SI by 2 to point to the next element
      add     SI,2
      dec     DI
      jnz     pass

      cmp     DX,SORTED       ; if status remains SORTED
      je      sort_done       ; then sorting is done
      jmp     next_pass       ; else initiate another pass

```

```
swap:
      ; swap elements at [SI] and [SI+2]
      xchg    AX,[SI+2]
      mov     [SI],AX
      mov     DX,UNSORTED     ; set status to UNSORTED
      jmp     increment

sort_done:
      popa
      ret     4               ; return and clear parameters
bubble_sort    ENDP
      END    main
```

```
TITLE   Variable # of parameters passed via stack   VARPARA.ASM
COMMENT |
        Objective: To show how variable number of parameters
                   can be passed via the stack
            Input: Requests variable number of non-zero integers.
                   A zero terminates the input.
|          Output: Outputs the sum of input numbers.
CRLF   EQU   0DH,0AH    ; carriage return and line feed
.MODEL SMALL
.STACK 100H
.DATA
prompt_msg  DB  'Please input a set of non-zero integers.',CRLF
            DB  'You must enter at least one integer.',CRLF
            DB  'Enter zero to terminate the input.',0
sum_msg     DB  'The sum of the input numbers is: ',0

.CODE
INCLUDE io.mac

main  PROC
      .STARTUP
      PutStr  prompt_msg     ; request input numbers
      nwln
      sub     CX,CX          ; CX keeps number count
```

```
read_number:
      GetInt  AX             ; read input number
      nwln
      cmp     AX,0           ; if the number is zero
      je      stop_reading   ; no more numbers to read
      push    AX             ; place the number on stack
      inc     CX             ; increment number count
      jmp     read_number
stop_reading:
      push    CX             ; place number count on stack
      call    variable_sum   ; returns sum in AX
      ; clear parameter space on the stack
      inc     CX             ; increment CX to include count
      add     CX,CX          ; CX := CX * 2 (space in bytes)
      add     SP,CX          ; update SP to clear parameter
                             ; space on the stack
      PutStr  sum_msg        ; display the sum
      PutInt  AX
      nwln
done:
      .EXIT
main  ENDP

```

```
;-----------------------------------------------------------
;This procedure receives variable number of integers via the
; stack. The last parameter pushed on the stack should be
; the number of integers to be added. Sum is returned in AX.
;-----------------------------------------------------------
variable_sum  PROC
      push    BP               ; save BP - procedure uses BP
      mov     BP,SP            ; copy SP to BP
      push    BX               ; save BX and CX
      push    CX

      mov     CX,[BP+4]        ; CX := # of integers to be added
      mov     BX,BP
      add     BX,6             ; BX := pointer to first number
      sub     AX,AX            ; sum := 0
add_loop:
      add     AX,SS:[BX]       ; sum := sum + next number
      add     BX,2             ; BX points to the next integer
      loop    add_loop         ; repeat count in CX

      pop     CX               ; restore registers
      pop     BX
      pop     BP
      ret                      ; parameter space cleared by main
variable_sum  ENDP
      END     main
```

```
TITLE   Fibonacci numbers (register version)    PROCFIB1.ASM
COMMENT |
        Objective: To compute Fibonacci number using registers
                   for local variables.
            Input: Requests a positive integer from the user.
           Output: Outputs the largest Fibonacci number that
|                  is less than or equal to the input number.

.MODEL SMALL
.STACK 100H
.DATA
prompt_msg   DB  'Please input a positive number (>1): ',0
output_msg1  DB  'The largest Fibonacci number less than '
             DB  'or equal to ',0
output_msg2  DB  ' is ',0

.CODE
INCLUDE io.mac

```

```
main  PROC
      .STARTUP
      PutStr  prompt_msg     ; request input number
      GetInt  DX             ; DX := input number
      nwln
      call    fibonacci
      PutStr  output_msg1    ; display Fibonacci number
      PutInt  DX
      PutStr  output_msg2
      PutInt  AX
      nwln
done:
      .EXIT
main  ENDP

```

```
;------------------------------------------------------
;Procedure fibonacci receives an integer in DX and computes
; the largest Fibonacci number that is less than or equal to
; the input number. The Fibonacci number is returned in AX.
;------------------------------------------------------
fibonacci  PROC
      push    BX
      ; AX maintains the smaller of the last two Fibonacci
      ;  numbers computed; BX maintains the larger one.
      mov     AX,1           ; initialize AX and BX to
      mov     BX,AX          ;  first two Fibonacci numbers
fib_loop:
      add     AX,BX          ; compute next Fibonacci number
      xchg    AX,BX          ; maintain the required order
      cmp     BX,DX          ; compare with input number in DX
      jle     fib_loop       ; if not greater, find next number
      ; AX contains the required Fibonacci number
      pop     BX
      ret
fibonacci  ENDP
```

```
TITLE   Fibonacci numbers (stack version)    PROCFIB2.ASM
COMMENT |
        Objective: To compute Fibonacci number using the stack
                   for local variables.
            Input: Requests a positive integer from the user.
           Output: Outputs the largest Fibonacci number that
|                  is less than or equal to the input number.
.MODEL SMALL
.STACK 100H
.DATA
prompt_msg   DB  'Please input a positive number (>1): ',0
output_msg1  DB  'The largest Fibonacci number less than '
             DB  'or equal to ',0
output_msg2  DB  ' is ',0

.CODE
INCLUDE io.mac

```

```
main  PROC
      .STARTUP
      PutStr  prompt_msg     ; request input number
      GetInt  DX             ; DX := input number
      nwln
      call    fibonacci
      PutStr  output_msg1    ; print Fibonacci number
      PutInt  DX
      PutStr  output_msg2
      PutInt  AX
      nwln
done:
      .EXIT
main  ENDP

;-------------------------------------------------------------
;Procedure fibonacci receives an integer in DX and computes
; the largest Fibonacci number that is less than the input
; number. The Fibonacci number is returned in AX.
;-------------------------------------------------------------
FIB_LO   EQU  WORD PTR [BP-2]
FIB_HI   EQU  WORD PTR [BP-4]
```

```
fibonacci  PROC
      push    BP
      mov     BP,SP
      sub     SP,4           ; space for local variables
      push    BX
      ; FIB_LO maintains the smaller of the last two Fibonacci
      ;  numbers computed; FIB_HI maintains the larger one.
      mov     FIB_LO,1       ; initialize FIB_LO and FIB_HI to
      mov     FIB_HI,1       ;  first two Fibonacci numbers
fib_loop:
      mov     AX,FIB_HI      ; compute next Fibonacci number
      mov     BX,FIB_LO
      add     BX,AX
      mov     FIB_LO,AX
      mov     FIB_HI,BX
      cmp     BX,DX          ; compare with input number in DX
      jle     fib_loop       ; if not greater, find next number
      ; AX contains the required Fibonacci number
      pop     BX
      mov     SP,BP          ; clear local variable space
      pop     BP
      ret
fibonacci  ENDP
      END     main
```

```
TITLE   Multimodule program for string length   MODULE1.ASM
COMMENT |
        Objective: To show parameter passing via registers
            Input: Requests two integers from keyboard.
|          Output: Outputs the sum of the input integers.
BUF_SIZE  EQU  41   ; string buffer size
.MODEL SMALL
.STACK 100H
.DATA
prompt_msg   DB   'Please input a string: ',0
length_msg   DB   'String length is: ',0
string1      DB   BUF_SIZE DUP (?)

.CODE
INCLUDE io.mac
```

```
EXTRN   string_length:PROC
main  PROC
      .STARTUP
      PutStr  prompt_msg     ; request a string
      GetStr  string1,BUF_SIZE  ; read string input
      nwln
      mov     BX,OFFSET string1 ; BX := string pointer
      call    string_length  ; returns string length in AX
      PutStr  length_msg     ; display string length
      PutInt  AX
      nwln
done:
      .EXIT
main  ENDP
      END     main
```

```
TITLE         String length procedure         MODULE2.ASM
COMMENT |
        Objective: To write a procedure to compute string
                   length of a NULL terminated string.
            Input: String pointer in BX register.
|          Output: Returns string length in AX.
.MODEL SMALL
.CODE
PUBLIC string_length
string_length PROC
      ; all registers except AX are preserved
      push    SI              ; save SI
      mov     SI,BX           ; SI := string pointer
repeat:
      cmp     BYTE PTR [SI],0  ; is it NULL?
      je      done             ; if so, done
      inc     SI               ; else, move to next character
      jmp     repeat           ;       and repeat
done:
      sub     SI,BX           ; compute string length
      mov     AX,SI           ; return string length in AX
      pop     SI              ; restore SI
      ret
string_length ENDP
      END
```