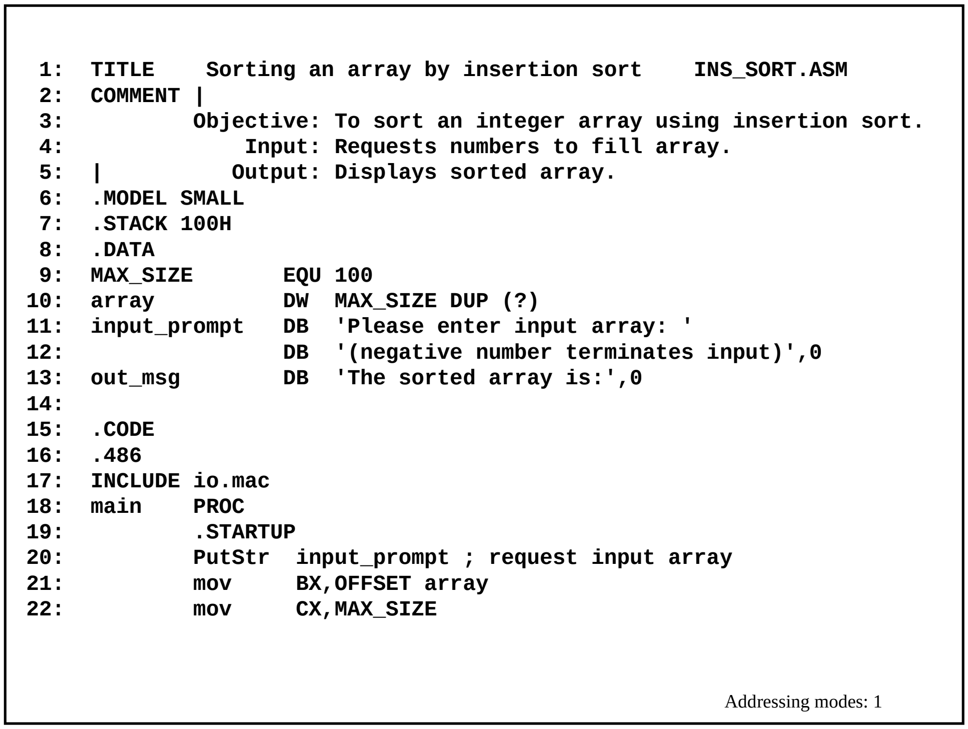
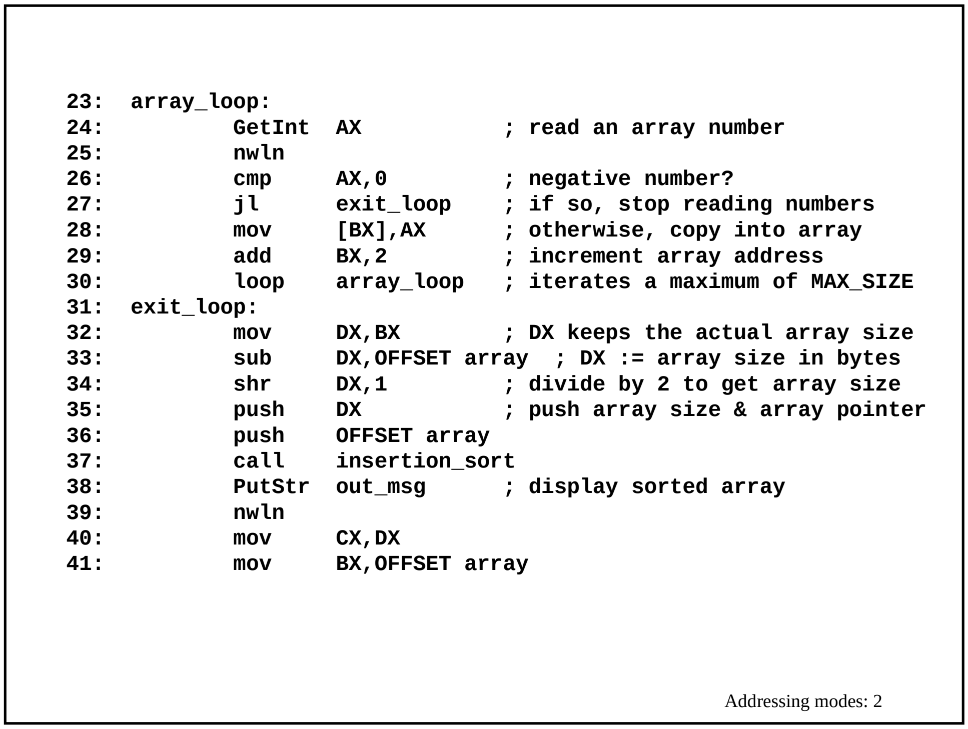


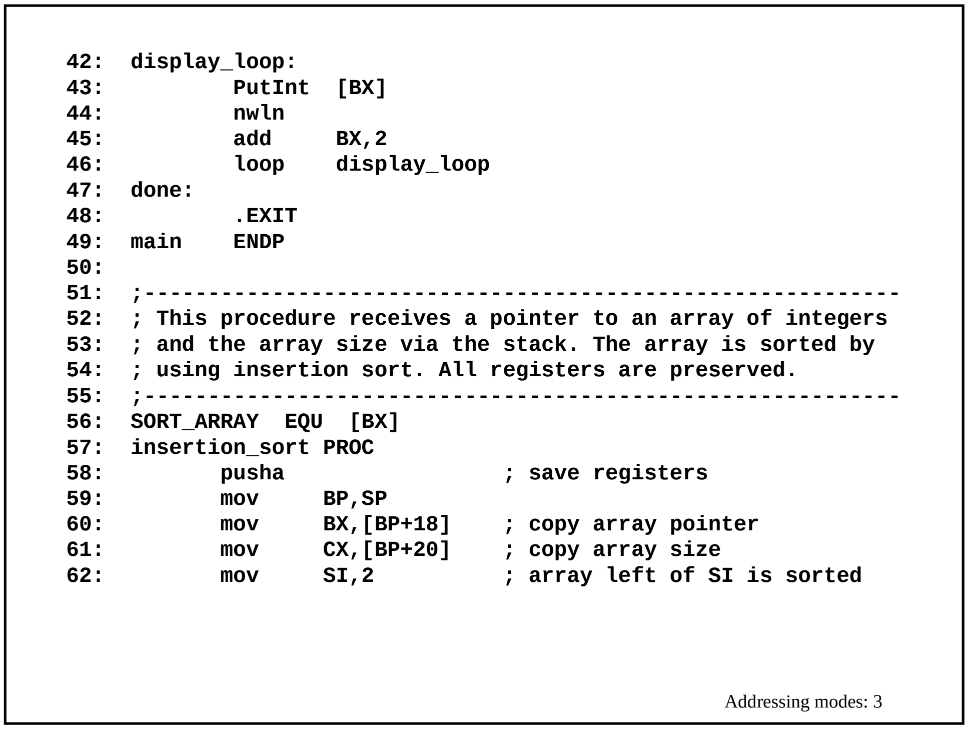
```
TITLE    Sorting an array by insertion sort    INS_SORT.ASM
COMMENT |
        Objective: To sort an integer array using insertion sort.
            Input: Requests numbers to fill array.
|          Output: Displays sorted array.
.MODEL SMALL
.STACK 100H
.DATA
MAX_SIZE       EQU 100
array          DW  MAX_SIZE DUP (?)
input_prompt   DB  'Please enter input array: '
               DB  '(negative number terminates input)',0
out_msg        DB  'The sorted array is:',0

.CODE
.486
INCLUDE io.mac
main    PROC
        .STARTUP
        PutStr  input_prompt ; request input array
        mov     BX,OFFSET array
        mov     CX,MAX_SIZE
```

Addressing modes: 1



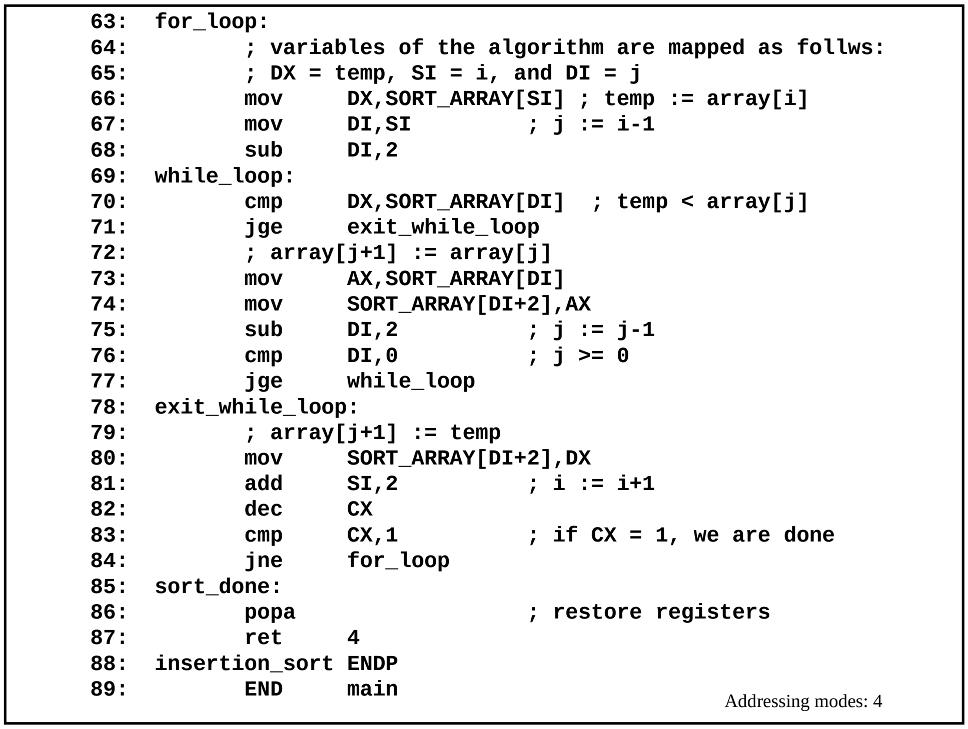
```
array_loop:
        GetInt  AX           ; read an array number
        nwln
        cmp     AX,0         ; negative number?
        jl      exit_loop    ; if so, stop reading numbers
        mov     [BX],AX      ; otherwise, copy into array
        add     BX,2         ; increment array address
        loop    array_loop   ; iterates a maximum of MAX_SIZE
exit_loop:
        mov     DX,BX        ; DX keeps the actual array size
        sub     DX,OFFSET array  ; DX := array size in bytes
        shr     DX,1         ; divide by 2 to get array size
        push    DX           ; push array size & array pointer
        push    OFFSET array
        call    insertion_sort
        PutStr  out_msg      ; display sorted array
        nwln
        mov     CX,DX
        mov     BX,OFFSET array
```

Addressing modes: 2



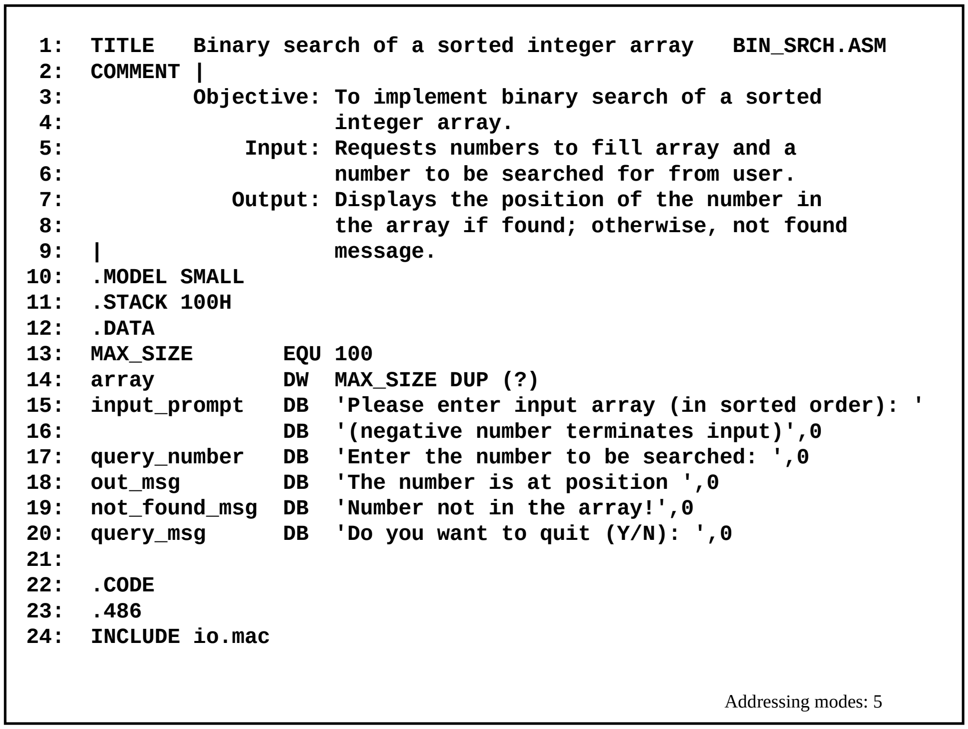
```
display_loop:
        PutInt  [BX]
        nwln
        add     BX,2
        loop    display_loop
done:
        .EXIT
main    ENDP

;-----------------------------------------------------------
; This procedure receives a pointer to an array of integers
; and the array size via the stack. The array is sorted by
; using insertion sort. All registers are preserved.
;-----------------------------------------------------------
SORT_ARRAY  EQU  [BX]
insertion_sort PROC
       pusha                  ; save registers
       mov     BP,SP
       mov     BX,[BP+18]     ; copy array pointer
       mov     CX,[BP+20]     ; copy array size
       mov     SI,2           ; array left of SI is sorted
```

Addressing modes: 3

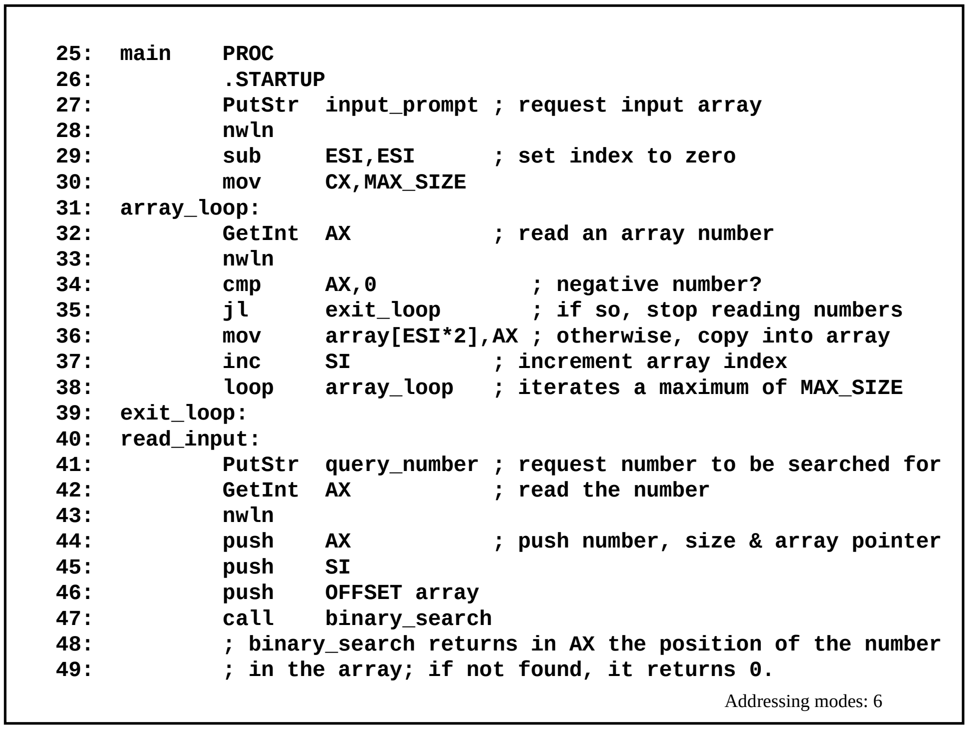


```
for_loop:
       ; variables of the algorithm are mapped as follws:
       ; DX = temp, SI = i, and DI = j
       mov     DX,SORT_ARRAY[SI] ; temp := array[i]
       mov     DI,SI          ; j := i-1
       sub     DI,2
while_loop:
       cmp     DX,SORT_ARRAY[DI]  ; temp < array[j]
       jge     exit_while_loop
       ; array[j+1] := array[j]
       mov     AX,SORT_ARRAY[DI]
       mov     SORT_ARRAY[DI+2],AX
       sub     DI,2           ; j := j-1
       cmp     DI,0           ; j >= 0
       jge     while_loop
exit_while_loop:
       ; array[j+1] := temp
       mov     SORT_ARRAY[DI+2],DX
       add     SI,2           ; i := i+1
       dec     CX
       cmp     CX,1           ; if CX = 1, we are done
       jne     for_loop
sort_done:
       popa                   ; restore registers
       ret     4
insertion_sort ENDP
       END     main
```

Addressing modes: 4



```
TITLE   Binary search of a sorted integer array   BIN_SRCH.ASM
COMMENT |
        Objective: To implement binary search of a sorted
                   integer array.
            Input: Requests numbers to fill array and a
                   number to be searched for from user.
           Output: Displays the position of the number in
                   the array if found; otherwise, not found
|                  message.
.MODEL SMALL
.STACK 100H
.DATA
MAX_SIZE       EQU 100
array          DW  MAX_SIZE DUP (?)
input_prompt   DB  'Please enter input array (in sorted order): '
               DB  '(negative number terminates input)',0
query_number   DB  'Enter the number to be searched: ',0
out_msg        DB  'The number is at position ',0
not_found_msg  DB  'Number not in the array!',0
query_msg      DB  'Do you want to quit (Y/N): ',0

.CODE
.486
INCLUDE io.mac
```



```
main    PROC
        .STARTUP
        PutStr  input_prompt ; request input array
        nwln
        sub     ESI,ESI      ; set index to zero
        mov     CX,MAX_SIZE
array_loop:
        GetInt  AX           ; read an array number
        nwln
        cmp     AX,0            ; negative number?
        jl      exit_loop       ; if so, stop reading numbers
        mov     array[ESI*2],AX ; otherwise, copy into array
        inc     SI           ; increment array index
        loop    array_loop   ; iterates a maximum of MAX_SIZE
exit_loop:
read_input:
        PutStr  query_number ; request number to be searched for
        GetInt  AX           ; read the number
        nwln
        push    AX           ; push number, size & array pointer
        push    SI
        push    OFFSET array
        call    binary_search
        ; binary_search returns in AX the position of the number
        ; in the array; if not found, it returns 0.
        cmp     AX,0         ; number found?
        je      not_found    ; if not, display number not found
        PutStr  out_msg      ; else, display number position
        PutInt  AX
        jmp     user_query
not_found:
        PutStr  not_found_msg
user_query:
        nwln
        PutStr  query_msg    ; query user whether to terminate
        GetCh   AL           ; read response
        nwln
        cmp     AL,'Y'       ; if response is not 'Y'
        jne     read_input   ; repeat the loop
done:                        ; otherwise, terminate program
        .EXIT
main    ENDP

;------------------------------------------------------------
; This procedure receives a pointer to an array of integers,
; the array size, and a number to be searched via the stack.
; It returns in AX the position of the number in the array
; if found; otherwise, returns 0.
; All registers, except AX, are preserved.
;------------------------------------------------------------
```

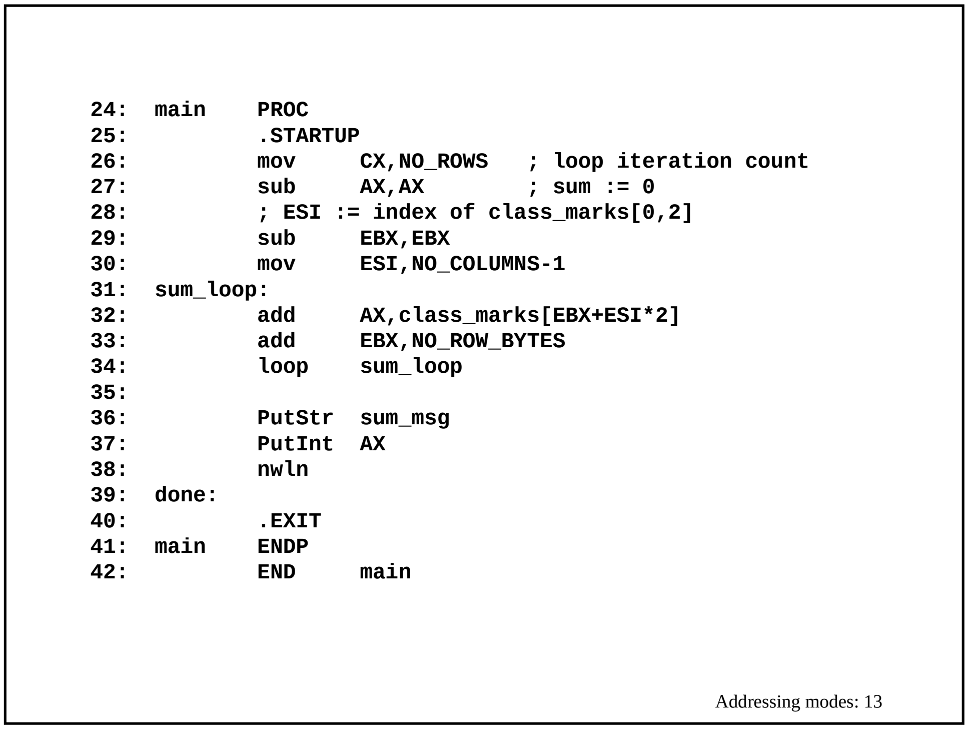
```
binary_search  PROC
       push    BP            ; save registers
       mov     BP,SP
       push    EBX
       push    ESI
       push    CX
       push    DX
       sub     EBX,EBX       ; EBX := 0
       mov     BX,[BP+4]     ; copy array pointer
       mov     CX,[BP+6]     ; copy array size
       mov     DX,[BP+8]     ; copy number to be searched
       sub     AX,AX         ; lower := 0
       dec     CX            ; upper := size-1
while_loop:
       cmp     AX,CX         ;lower > upper?
       ja      end_while
       sub     ESI,ESI
       mov     SI,AX         ; middle := (lower + upper)/2
       add     SI,CX
       shr     SI,1
       cmp     DX,[EBX+ESI*2]    ; number = array[middle]?
       je      search_done
       jg      upper_half
```

```
lower_half:
       dec     SI             ; middle := middle-1
       mov     CX,SI          ; upper := middle-1
       jmp     while_loop
upper_half:
       inc     SI             ; middle := middle+1
       mov     AX,SI          ; lower := middle+1
       jmp     while_loop
end_while:
       sub     AX,AX          ; number not found (clear AX)
       jmp     skip1
search_done:
       inc     SI             ; position := index+1
       mov     AX,SI          ; return position
skip1:
       pop     DX             ; restore registers
       pop     CX
       pop     ESI
       pop     EBX
       pop     BP
       ret     6
binary_search  ENDP
       END     main
```

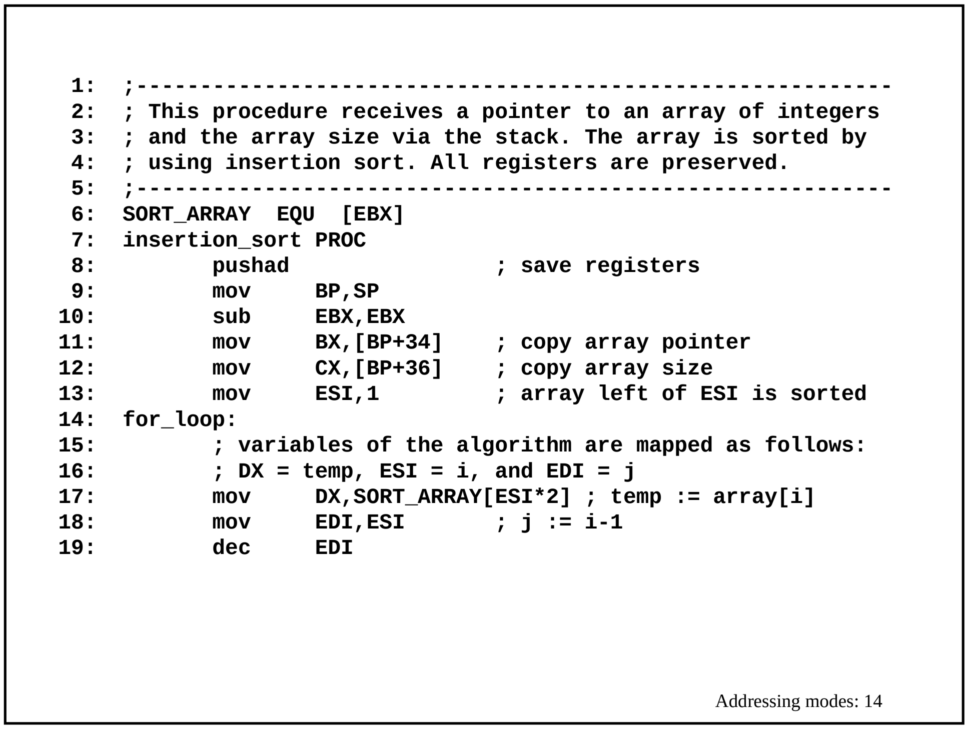
```
TITLE       Sum of a long integer array       ARAY_SUM.ASM
COMMENT |
        Objective: To find sum of all elements of an array.
            Input: None
|          Output: Displays the sum.
.MODEL SMALL
.STACK 100H
.DATA
test_marks     DD  90,50,70,94,81,40,67,55,60,73
NO_STUDENTS    EQU ($-test_marks)/4     ; number of students
sum_msg        DB  'The sum of test marks is: ',0

.CODE
.486
INCLUDE io.mac
```

```
main    PROC
        .STARTUP
        mov     CX,NO_STUDENTS   ; loop iteration count
        sub     EAX,EAX          ; sum := 0
        sub     ESI,ESI          ; array index := 0
add_loop:
        mov     EBX,test_marks[ESI*4]
        PutLInt EBX
        nwln
        add     EAX,test_marks[ESI*4]
        inc     ESI
        loop    add_loop

        PutStr  sum_msg
        PutLInt EAX
        nwln
        .EXIT
main    ENDP
        END     main
```

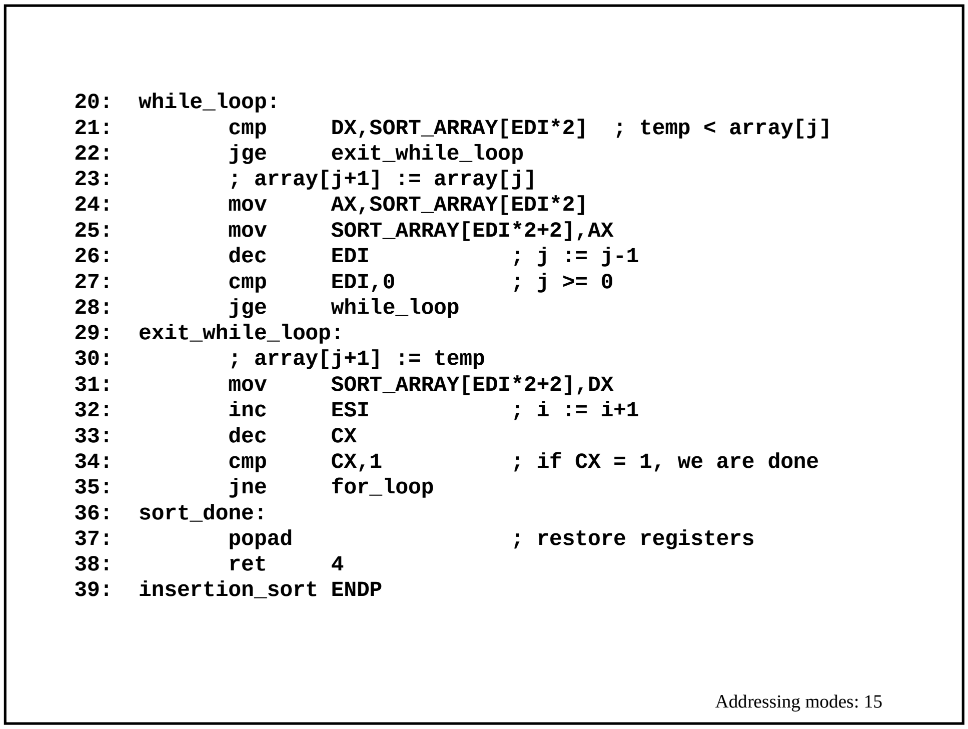
```
TITLE  Sum of a column in a 2-dimensional array  TEST_SUM.ASM
COMMENT |
        Objective: To demonstrate array index manipulation
                   in a two-dimensional array of integers.
            Input: None
|          Output: Displays the sum.
.MODEL SMALL
.STACK 100H
.DATA
NO_ROWS        EQU  5
NO_COLUMNS     EQU  3
NO_ROW_BYTES   EQU  NO_COLUMNS * 2  ; number of bytes per row
class_marks    DW   90,89,99
               DW   79,66,70
               DW   70,60,77
               DW   60,55,68
               DW   51,59,57

sum_msg        DB   'The sum of the last test marks is: ',0

.CODE
.486
INCLUDE io.mac
```



```
main    PROC
        .STARTUP
        mov     CX,NO_ROWS   ; loop iteration count
        sub     AX,AX        ; sum := 0
        ; ESI := index of class_marks[0,2]
        sub     EBX,EBX
        mov     ESI,NO_COLUMNS-1
sum_loop:
        add     AX,class_marks[EBX+ESI*2]
        add     EBX,NO_ROW_BYTES
        loop    sum_loop

        PutStr  sum_msg
        PutInt  AX
        nwln
done:
        .EXIT
main    ENDP
        END     main
```



```
;-----------------------------------------------------------
; This procedure receives a pointer to an array of integers
; and the array size via the stack. The array is sorted by
; using insertion sort. All registers are preserved.
;-----------------------------------------------------------
SORT_ARRAY  EQU  [EBX]
insertion_sort PROC
       pushad                ; save registers
       mov     BP,SP
       sub     EBX,EBX
       mov     BX,[BP+34]    ; copy array pointer
       mov     CX,[BP+36]    ; copy array size
       mov     ESI,1         ; array left of ESI is sorted
for_loop:
       ; variables of the algorithm are mapped as follows:
       ; DX = temp, ESI = i, and EDI = j
       mov     DX,SORT_ARRAY[ESI*2] ; temp := array[i]
       mov     EDI,ESI       ; j := i-1
       dec     EDI
```



```
while_loop:
       cmp     DX,SORT_ARRAY[EDI*2]  ; temp < array[j]
       jge     exit_while_loop
       ; array[j+1] := array[j]
       mov     AX,SORT_ARRAY[EDI*2]
       mov     SORT_ARRAY[EDI*2+2],AX
       dec     EDI           ; j := j-1
       cmp     EDI,0         ; j >= 0
       jge     while_loop
exit_while_loop:
       ; array[j+1] := temp
       mov     SORT_ARRAY[EDI*2+2],DX
       inc     ESI           ; i := i+1
       dec     CX
       cmp     CX,1          ; if CX = 1, we are done
       jne     for_loop
sort_done:
       popad                 ; restore registers
       ret     4
insertion_sort ENDP
```