Flag setting instructions

- set one flag, do not change the other flags
- do not have operands

clc	clear carry flag: CF =0
stc	set carry flag: CF = 1
cmc	complement carry flag

- inverts CF

cld	clear direction flag: DF = 0

std set direction flag: DF = 1

cli clear interrupt flag: IF = 0

sti set interrupt flag: IF = 1

lahf load flags into AH register

- copies the lowest 8 bits of the EFLAGS register to AH:

7	6	5	4	3	2	1	0
SF	ZF	0	AF	0	PF	1	CF

sahf store AH register to flags

- copies AH to the lowest byte of the EFLAGS register.

String instructions

- useful when operating on arrays of type byte, word and dword.

Before a string instruction is executed, ESI must be set to the address (offset) of the source string and EDI to the address of the destination string.

The ESI and EDI registers hold indexes to the strings; after the operation (e.g. copy, comparison, ...) on one element of the string was performed, the ESI and/or EDI registers are automatically increased (if DF is 0) or decreased (if DF is 1) by the size of the string element (1, 2 or 4 according to the string type).

Value of DF	Impact on ESI and EDI	Direction of the string operation	Addresses in the string are accessed
0	increment	forward	from lower to higher
1	decrement	backward	from higher to lower

The type of the string may be specified either by the operand (the name of the string variable) or by the postfix in the instruction name (letter b, w, or d, respectively).

Prefix rep repeats the string instruction until the ECX becomes zero. If ECX = 0 before the instruction, the instruction is executed not once.

movs destination, source
movsb
movsw
movsd

move data from source string to destination string

- copies a data element from the location pointed by ESI into the location pointed by EDI
- does not change flags nor EAX

Instruction	Copy	ESI and EDI are increased/decreased by
movsb	byte	1
movsw	word	2
movsd	doubleword	4

- Copy the array of 20 doublewords from variable Source to variable Destination.

After the operation the ESI and EDI registers will point behind the arrays.

```
.data
Source DD 20 DUP(0FFFFFFFFh)
Destination DD 20 DUP(0)
.code
main PROC
  cld; direction = forward
  mov ecx, length Source; the number of repetitions
  mov esi, offset Source; ESI points to Source
  mov edi, offset Destination; EDI points to Destination
  rep movs Destination,Source; or rep movsd
exit
main ENDP
END main
```

lods source string
lodsb
lodsw
lodsd

load data from string

- copies a byte from the string to AL (resp. word to AX, resp. dword to EAX)
- does not change flags

```
cld
mov esi,offset String
lodsb          → AC
```

better!

the same as

```
mov al,[esi]   → 8A 06
inc esi        → 46
```

stos destination string
stosb
stosw
stosd

store data to string

- copies the contents of AL (resp. AX, resp. EAX) to the string
- does not change flags

- Set each element of Array to zero using instruction stosw.

```
.data
ArrayLength EQU 20
Array DW ArrayLength dup(0FFFFh)
.code
main PROC


exit
main ENDP
END main
```

- Set each element of Array to zero using instruction stosw.

```
.data
ArrayLength EQU 20
Array DW ArrayLength dup(0FFFFh)
.code
main PROC

cld; direction = forward
mov edi,offset Array
mov ecx,ArrayLength
xor ax,ax; AX = 0
rep stosw

exit
main ENDP
END main
```

scas destination string
scasb
scasw
scasd

scan string

- compares a byte (word or doubleword) of the string with register AL (AX or EAX) in such a way that subtracts an element of the string from AL (AX or EAX) and sets flags.

cmps destination, source
cmpsb
cmpsw
cmpsd

compare string operands

- compares strings in such a way that subtracts an element of the destination string from an element of the source string and sets flags

Prefix `repe` (`repz`) repeats the string instruction while ECX > 0 and ZF = 1. If ECX = 0 before the instruction, the instruction is executed not once.

Prefix `repne` (`repnz`) repeats the string instruction while at once ECX > 0 and ZF = 0. If ECX = 0 before the instruction, the instruction is executed not once.

- Read the name of a file. Find the dot in it.

```
.data
FileName DB 20 dup(?)
.code
main PROC

  mov edx,offset FileName
  mov ecx,20
  call ReadString; read the file name

  mov ecx,eax; store the number of characters typed to ecx
  mov al,'.'
  cld
  mov edi,edx
  repne scasb; edi points behind '.', if it is there
  jne IsNot
  dec edi; edi point to '.'
IsNot:
```

Macroinstruction

Macro(instruction) is a block of text to which you assign a name.

Whenever the compiler encounters that name in the source code, it replaces the name by the actual block of text.

Comparing to a procedure, a macro is executed faster (call and ret are not executed), but it does not save memory.

- Read three characters, store them to variables Letter1, Letter2, Letter3, order them according to the alphabet and write them.

```
.data
Letter1 DB ?
Letter2 DB ?
Letter3 DB ?
.code
ReadLetter MACRO paLetter
      call ReadChar
      call WriteChar
      mov paLetter,al
ENDM
```

```
WriteLetter MACRO WhichLetter
      mov al,Letter&WhichLetter
      call WriteChar
ENDM
```

The special symbol & concatenates two strings: Letter and the actual parameter corresponding to the formal parameter WhichLetter; we get the name of a variable.

```
Order MACRO First,Second
LOCAL Finish
      mov al,First
      cmp al,Second
      jbe Finish
      xchg al,Second
      mov First,al
Finish:
ENDM
```

```
main PROC

      ReadLetter Letter1
      ReadLetter Letter2
      ReadLetter Letter3
      Order Letter1,Letter2
      Order Letter2,Letter3
      Order Letter1,Letter2
      WriteLetter 1
      WriteLetter 2
      WriteLetter 3

exit
main ENDP
END main
```

```
call ReadChar
call WriteChar
mov Letter1,al
```

```
mov al,Letter1
cmp al,Letter2
jbe ??0002
xchg al,Letter2
mov Letter1,al
??0002:
```

```
mov al,Letter1
call WriteChar
```

Directives for repeating blocks of statements

```
REPT  the number of repetitions (constant)
    }  block of statements to be repeated
ENDM
```

```
Write MACRO
      i = 0
      REPT 3
            i = i + 1
            WriteLetter %i
      ENDM
ENDM
```

The special symbol % causes that i will be evaluated to a number before using as the actual parameter.

```
main PROC
      ReadLetter Letter1
      ReadLetter Letter2
      ReadLetter Letter3
      Order Letter1,Letter2
      Order Letter2,Letter3
      Order Letter1,Letter2

      Write

      exit
main ENDP
```

```
mov al,Letter1
call WriteChar
mov al,Letter2
call WriteChar
mov al,Letter3
call WriteChar
```

Directives for repeating blocks of statements

```
IRP parameter,<arg1, arg2, ..., argN>
    block of statements – will be repeated N-times with
    parameter being substituted by arg1, arg2, ..., argN

ENDM
```

```
WriteAll MACRO
      IRP Letter,<Letter1,Letter2,Letter3>
            mov al,Letter
            call WriteChar
      ENDM
ENDM
```

```
main PROC
      ReadLetter Letter1
      ReadLetter Letter2
      ReadLetter Letter3
      Order Letter1,Letter2
      Order Letter2,Letter3
      Order Letter1,Letter2

      WriteAll

      exit
main ENDP
```

Actual parameters may be:

- numbers
- strings
- symbolic constants
- variables
- registers
- labels

Two-dimensional arrays

They are usually stored by rows.

They are accessed using an indirect address with base, index and displacement, where the displacement is the name of the array:

name[base + index]

Base is the offset of the row relative to the beginning of the array.

Index is the offset of the column relative to the beginning of the row.

- Store the element [1,2] of a two-dimensional array Matrix to the ax register.

```
.data
Matrix DW 10h, 20h, 30h, 40h, 50h
DW 60h, 70h, 80h, 90h, 0A0h
DW 0B0h, 0C0h, 0D0h, 0E0h, 0F0h
RowLentgh EQU sizeof Matrix; 10 bytes

.code
main PROC

RowIndex EQU 1
ColumnIndex EQU 2

mov ebx, RowLength*RowIndex; offset of the row
mov esi, ColumnIndex
mov ax, Matrix[ebx + esi*type Matrix]; AX = 80h
```

- Generate a matrix of 4 * 4 random non-negative numbers of type word. Store it to the memory and display it.

We use the procedures:

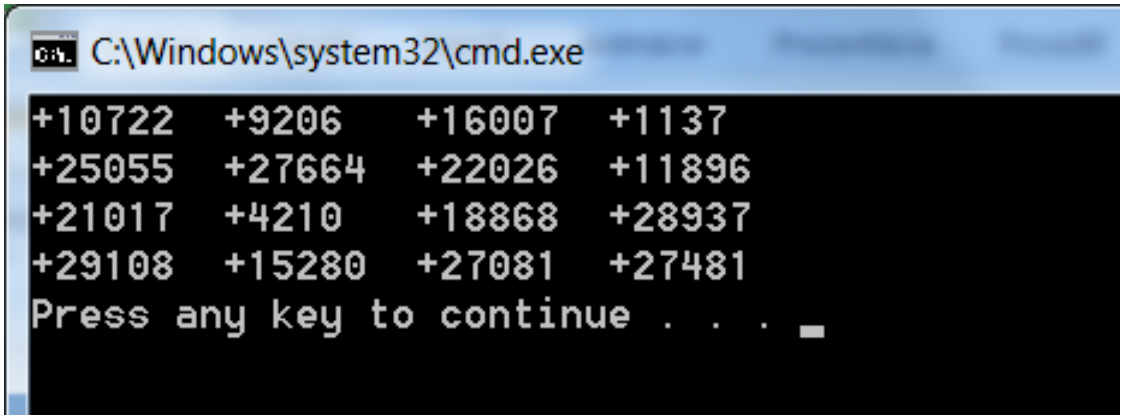
Procedure	Description	Input parameters	Output parameters
RandomRange	Generates a random number $\in \langle 0; n-1 \rangle$.	EAX - n	EAX – generated number
WriteInt	Write a signed integer.	EAX – number	

```
.data
n EQU 4
Matrix DW n*n dup(?)
.code
NewLine MACRO
        mov al,0Dh
        call WriteChar
        mov al,0Ah
        call WriteChar
ENDM
```

```
GenerateMatrix PROC USES EAX ECX ESI
; generate n*n random numbers and store them to Matrix
        mov esi,0
        mov ecx,n*n
Generate:
        mov eax,8000h
        call RandomRange
        mov Matrix[esi*type Matrix],ax
        inc esi
        loop Generate
        ret
GenerateMatrix ENDP
```

```
DisplayMatrix PROC USES EAX EBX ECX ESI
        mov ebx,0
        mov ecx,n; number of rows
WriteRow:
        mov esi,0; column index
        push ecx; save row counter
        mov ecx,n; number of columns
Write:
        movzx eax, Matrix[ebx+esi*type Matrix]
        call WriteInt
        mov al,9; Tab – indent columns
        call WriteChar
        inc esi
        loop Write
        NewLine
        pop ecx; restore row counter
        add ebx,n*type Matrix; update offset of the next row
        loop WriteRow
        ret
DisplayMatrix ENDP
```

```
main PROC
        call GenerateMatrix
        call DisplayMatrix
        exit
main ENDP
```



- Write a procedure that sums numbers on the main diagonal of the previously generated matrix. The procedure returns the sum in the EAX register.