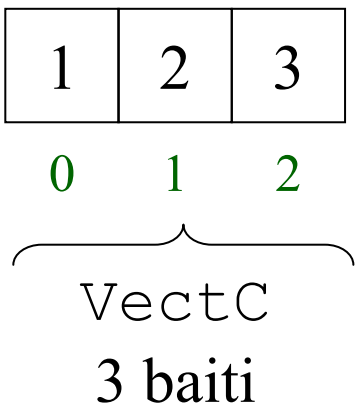
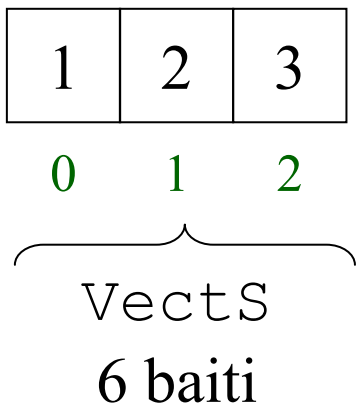
Atmiņas adresēšana

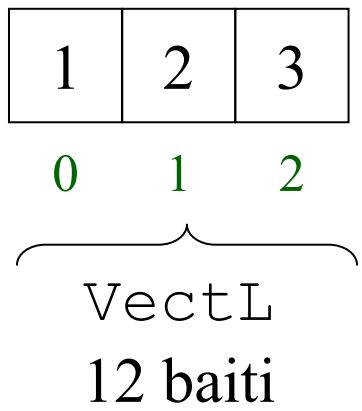
Viendimensijas masīvi *C++* valodā:

```
const int N = 3;
char  VectC[N] = {1, 2, 3};
short VectS[N] = {1, 2, 3};
long  VectL[N] = {1, 2, 3};
```

Iekšēja realizācija: indeksēšana (0, 1, 2) un masīvu izmēri.



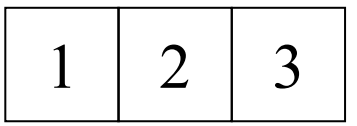




Viendimensijas masīvi *Assembler* valodā

```
N           Equ 3
VectB       DB  1, 2, 3
VectW       DW  1, 2, 3
VectD       DD  1, 2, 3
```

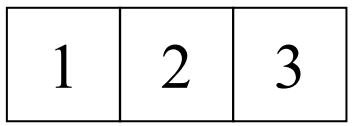
Iekšēja realizācija: indeksēšana *atkarīga* no elementu izmēra.



x x+1 x+2

VectB

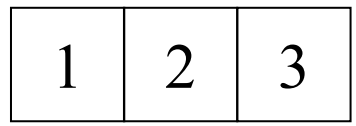
3 baiti



x x+2 x+4

VectW

6 baiti



x x+4 x+8

VectD

12 baiti

Informācija par elementu izmēru *Assembler* valodā: operators **Type**.

```
Mov   Ax, Type VectB    ;1
Mov   Ax, Type VectW    ;2
Mov   Ax, Type VectD    ;4
```

Tipa definēšana un rādītāju izveidošana: operators **Typedef**.

```
VectW     DW  1, 2, 3
PointerW  Typedef Ptr Word
Pw        PointerW VectW
...
Mov   Bx, Pw
Mov   Ax, [Bx] ;1
Add   Bx, 2
Mov   Ax, [Bx] ;2
```

Cikls **Loop**

```
    Mov  Cx, N     ; skaitītājs
S:
    ...
    Loop S
    ; 1. Cx := Cx-1
    ; 2. Cx=0 ?
```

Pārskaitļa (nepārskaitļa) pārbaude

```
Mov Ax, 5
Test Ax, 00000001B  ;Test Ax, 1
Jz  _Even
```

	?	?	?	?	?	?	?	1	
and									**=1**
	0	0	0	0	0	0	0	1	

	?	?	?	?	?	?	?	0	
and									**=0**
	0	0	0	0	0	0	0	1	

Cikli **Loope** (**Loopz**) un **Loopne** (**Loopnz**)

Loop while **E**qual/**Z**ero un **Loop** while **N**ot **E**qual/**Z**ero

Atrast pirmo skaitli, kas *nav nulle*.

```
    Mov Cx, N
    Mov Bx, -1
C:
    Inc Bx
    Cmp V[Bx], 0
```

```
LoopE C
Jne OK
```

Atrast pirmo skaitli, kas *ir* nulle.

```
    Mov Cx, N
    Mov Bx, -1
C:
    Inc Bx
    Cmp V[Bx], 0
```

```
LoopNE C
Je OK
```

Abos gadījumos spēkā ir `Loop` principi un ZF (*Zero Flag*) analīze.

Indeksa adresēšana

Elementu izvietošana akumulatorā: *baiti* un *vārdi*.

```
    Mov   Cx, N
    Xor   Bx, Bx
S:
    Mov   Al, VectB[Bx]
    Inc   Bx
    Loop  S
```

```
    Mov   Cx, N
    Xor   Bx, Bx
S:
    Mov   Ax, VectW[Bx]
    Add   Bx, 2
    Loop  S
```

Elementu izvietošana akumulatorā: *dubultvārdi*.

```
    ...
    Mov   EAx, VectD[Bx]
    Add   Bx, 4
```

Bāzes adresēšana

Elementu izvietošana akumulatorā: *baiti* un *vārdi*.

```
    Mov   Cx, N
    Lea   Bx, VectB
S:
    Mov   Al, [Bx]
    Inc   Bx
    Loop  S
```

```
    Mov   Cx, N
    Lea   Bx, VectW
S:
    Mov   Ax, [Bx]
    Add   Bx, 2
    Loop  S
```

Elementu izvietošana akumulatorā: *dubultvārdi*.

```
    ...
    Mov   EAx, [Bx]
    Add   Bx, 4
```

Iespējamie indeksa reģistri: `Bx, Si, Di, Bp`

Indeksa adresēšana **ar mērogošanu**

Elementu izvietošana akumulatorā: *baiti* un *vārdi*.

```
    Mov   Cx, N
    Xor   EDx, EDx
S:
    Mov   Al, VectB[EDx*1]
    Inc   EDx
    Loop  S
```

```
    Mov   Cx, N
    Xor   EDx, EDx
S:
    Mov   Ax, VectW[EDx*2]
    Inc   EDx
    Loop  S
```

Elementu izvietošana akumulatorā: *dubultvārdi*.

```
...
Mov   EAx, VectD[EDx*4]
Inc   EDx
```

Bāzes-indeksa adresēšana

Elementu izvietošana akumulatorā: *baiti* un *vārdi*.

```
  Mov   Cx, N
  Xor   Bx, Bx
  Xor   Si, Si
S:
  Mov   Al, VectB[Bx][Si]
  Inc   Bx
  Loop  S
```

```
  Mov   Cx, N
  Xor   Bx, Bx
  Xor   Si, Si
S:
  Mov   Ax, VectW[Bx][Si]
  Inc   Bx
  Inc   Si
  Loop  S
```

```
VectB[Bx][Si] = VectB[Si][Bx]
VectW[Bx][Si] = VectW[Si][Bx]
```