

Aritmetica de pointeri

În cadrul sistemului de adresare se efectuează operații cu adrese (pointeri). Care sunt operațiile ARITMETICE cu pointeri permise **ÎN INFORMATICĂ** ?...

Răspuns: Orice operație care are sens... aceasta însemnând orice operație ce exprimă ca rezultat o localizare în memorie corectă și utilă ca informație pt programator.

Aritmetica de pointeri/adrese (pointer arithmetic – operații ARITMETICE cu pointeri) = utilizarea de expresii aritmetice, care au ca operanzi adrese !

- adunări și scăderi de adrese ?
 - Adunare de adrese = ??? CE reprezintă ? NIMIC !!!!!
 - Scădere de adrese = ??? CE reprezintă ? $q-p$ = nr **octeți** dintre cele două adrese de memorie (niciodată nu depășim dim memoriei ; valoarea obținută este o CONSTANTA NUMERICA !!!!)
- adunări și scăderi de constante la o/dintr-o adresă = necesare și utile pt accesarea elementelor dintr-un array
- înmulțirea a două adrese ?- nepermisă (în majoritatea cazurilor valoarea obținută este dincolo de limita maximă a memoriei posibile a fi accesată).
- Înmulțirea cu o constantă (în majoritatea cazurilor valoarea obținută este dincolo de limita maximă a memoriei posibile a fi accesată). În plus, CE reprezintă valoarea obținută ?... Nimic util !!
- Împărțire ?... No way !
- Singura excepție de la regulile aritmeticii de pointeri o constituie formula de calcul a offsetului unui operand unde sunt permise adunări de valori de regiștri (NU adunări de pointeri!!)... În rest nu există excepții

$$a[7] = *(a+7) = *(7+a) = 7[a]$$

- atât în C cât și în asamblare !

Pointer arithmetic...? DOAR 3 operații sunt permise cu POINTERI:

1). Scăderea a doua adrese

Adresa – adresa = ok (q-p = scadere de 2 pointeri = sizeof(array) sau nr de elemente (in C)/octeti (asamblare) dintre doua adrese de memorie)

2) Adunarea unei constante numerice la o adresă

Adresa + constanta numerica (identificarea unui element prin indexare – a[7]), q+9

3). Scăderea unei constante numerice dintr-o adresă

Adresa – constanta numerica - a[-4], p-7

- scăderea a 2 pointeri – valoare SCALARA !!! (valoare numerică constantă imediată)
- adunarea unei constante la un pointer → POINTER !
- scăderea unei constante dintr-un pointer → POINTER !

(ultimele două sunt utile pt referirea de elemente dintr-un array/zonă de memorie)

ADUNAREA A DOI POINTERI NU ESTE PERMISA !!!!!

p+q = ??? (allowed in NASM...sometimes...!!!!!!) – dar nu inseamna ADUNARE DE POINTERI in cele din urma asa cum vom vedea...

V db 17

add edx, [EBX+ECX*2 + v -7] – OK !!!!

mov ebx, [EBX+ECX*2 - v-7] – Syntax error !!!! invalid effective address – impossible segment base multiplier

adc ecx, [EBX+ECX*2 + a+b-7] – Syntax error din cauza “a+b”; invalid effective address – impossible segment base multiplier

sub [EBX+ECX*2 + a-b-7], eax – ok, pt că a-b este o operație corectă cu pointeri !!!

[EBX+ECX*2 + v -7] – ok
SIB depl. const.

[EBX+ECX*2 + a-b-7]

const.

```
mov eax, [EBX+ECX*2+(-7)] – ok.
```

L-value; R-value. Valoare stângă vs. valoare dreaptă a unei atribuiri.

Atribuire: $i := i + 1$

LHS vs. RHS

(Adresa lui I \leftarrow valoarea lui I + 1)

$$\text{LHS}(i) = \text{adresa lui } I := \text{RHS}(i) = (\text{continutul de la adresa } I) + 1$$

LHS (valoare stanga a unei atribuirii este o L-value = adresa) := RHS

(valoarea dreapta a unei atribuirii = R-Value = CONTINUT !!)

Sintaxele majorității limbajelor de programare prevăd că:

Symbol := expression_value, adică Identificator := expresie

In fapt, sunt limbaje (C++, ASAMBLARE !!!) care permit mai general sintaxa:

Expresie_calcul_de_adresa := expresie

```
(mov [ebx+2*EDX+v-7], a+2)
```

Dereferențierea (extragerea valorii de la o adresa) este implicită în 99% din limbaje. Exemplu excepție – limbajul BLISS – unde dereferențierea trebuie menționată explicit întotdeauna; $i \leftarrow *i+1$
(+ unele situații în Algol 68)

Symbol := expression value (99% of the cases...)

$$Address_computation_Expression := expression_value$$

In C++ $f(a+3, b-2, 2) = x+y+z$

Variabilele “referință C++” (C++ reference variables) au 3 utilizări:

- 1) `Int& j = i; // j devine ALIAS pt i`
- 2) Transmiterea de variabile prin referință la apelul de subprograme
`float f(int&x, y);.....`
- 3) Returnarea de L-valori prin intermediul funcțiilor

`Int& f(x,i) {...return v[i];}` – Funcția f returnează o LHS (valoare stângă)
`F(a,7) = 79; înseamnă că v[7]=79 !!!`

De asemenea, separat de acestea se permite și utilizarea operatorului condițional ternar pe post de valoare stângă:

`(a+2?b:c) = x+y+z ; - correct`
`(a+2?1:c) = x+y+z; - syntax error !!! 1:=n !!!!`

Segment data

A db 17,-2
B dw -20345, “x”
Start2:
C dd 12345678h

Segment code

.....
Jmp start2 ; exemplu de salt intr-un data segment, deci pot exista etichete de cod
; intr-un segment de date !
.....
