Arhitectura Sistemelor de Calcul

Lect. Dr. Șotropa Diana
diana.sotropa@ubbcluj.ro

Facultatea de Matematică și Informatică
Universitatea Babeș-Bolyai

Aritmetica de pointeri

Aritmetica de pointeri

Care sunt operatiile ARITMETICE cu pointeri permise IN INFORMATICA ?

```
MOV EAX, [v1]; [] = operatorul de dereferențiere

a[7] = * (a+7); * = operatorul de dereferențiere
```

- Aritmetica de pointeri/adrese = utilizarea de expresii aritmetice, care au ca operanzi adrese

Aritmetica de pointeri

Care sunt operatiile ARITMETICE cu pointeri permise IN INFORMATICA ?

- adunări şi scăderi de adrese ?
  - Adunare de adrese = ? CE reprezinta ? NIMIC !!!!!
  - Scădere de adrese = ? CE reprezinta ? q-p = nr octeţi dintre cele două adrese de memorie (niciodată nu depăşim dim memoriei ; valoarea obţinută este o CONSTANTA NUMERICA)
- adunări şi scăderi de constante la o/dintr-o adresă = necesare şi utile pt accesarea elementelor dintr-un array
- înmulţirea a două adrese ? - nepermisă (in majoritatea cazurilor valoarea obtinuta este dincolo de limita maxima a memoriei posibile a fi accesata).
- Inmultirea cu o constanta ? - (in majoritatea cazurilor valoarea obtinuta este dincolo de limita maxima a memoriei posibile a fi accesata). In plus, CE reprezinta valoarea obtinuta ?... Nimic util !
- Impărţire ? ... No way !

Aritmetica de pointeri

- Singura exceptie de la regulile aritmeticii de pointeri o constituie **formula de calcul a offsetului unui operand** unde sunt permise adunări de valori de regiştri (NU adunări de pointeri!!)... In rest nu exista excepţii

```
MOV AX, a[7]
```

Ce e a[7]?

```
a[7]=*(a+7)=*(7+a)=7[a]; atât în C cât și în asamblare
```

Aritmetica de pointeri

- DOAR 3 operaţii sunt permise cu POINTERI:
  - **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
    => valoare SCALARA !!! (valoare numerică constantă imediată)
  - **Adunarea unei constante numerice la o adresă**
    `adresa + constanta numerica`
    identificarea unui element prin indexare – `a[7] , q+9`
    => POINTER
  - **Scăderea unei constante numerice dintr-o adresă**
    `adresa – constanta numerica - a[-4] , p-7`
    => POINTER

Aritmetica de pointeri

```
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! pentru că a-b
este o operaţie corectă cu pointeri
```

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

- Atribuire: i:=i+1

Q: i-ul din stanga este tot una cu i-ul din dreapta?

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

- Atribuire: i:=i+1
  `adresa lui i <- valoarea lui i + 1`

- LHS (valoarea stanga a unei atribuiri este o L-value = adresa)
- RHS (valoarea dreapta a unei atribuiri este o R-Value = continut)

- Sintaxele majorităţii 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 := valoare_expresie_aritmetică`

```
mov dword [ebx+2*EDX+v-7], a+2 ; este corect?
mov dword [ebx+2*EDX+v-7], [a+2] ; este corect?
```

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

- Atribuire: i:=i+1
  `adresa lui i <- valoarea lui i + 1`

- LHS (valoarea stanga a unei atribuiri este o L-value = adresa)
- RHS (valoarea dreapta a unei atribuiri este o R-Value = continut)

- Sintaxele majorităţii 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 := valoare_expresie_aritmetică`

```
mov dword [ebx+2*EDX+v-7], a+2 ; este corect? - DA
mov dword [ebx+2*EDX+v-7], [a+2] ; este corect? - NU
```

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

Q: Operatorul condițional ternar este L-value sau R-value?

- `(a + 2 ? b : c) = x + y + z ; este corect?`
- `(a + 2 ? 1 : c) = x + y + z ; este corect?`

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

Q: Operatorul condițional ternar este L-value sau R-value?

- `(a + 2 ? b : c) = x + y + z ;` este corect? - DA
- `(a + 2 ? 1 : c) = x + y + z ;` este corect? – NU! Syntax error

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

- `Address computation Expression := expression_value`
  In C++ `f(a+3, b-2, 2) = x+y+z`

- Variabilele “referinţă C++” au 3 utilizări:
  - `Int& j = i;` // j devine ALIAS pt i
  - Transmiterea de variabile prin referinţă la apelul de subprograme
    `float f(int&x, y)`
  - Returnarea de L-valori prin intermediul funcţiilor
    `Int& f(x,i) {….return v[i];}`
    Funcţia f returnează o LHS (valoare stângă)
    `f(a,7) = 79`; înseamnă că `v[7]=79`
- De asemenea, separat de acestea se permite şi utilizarea operatorului condiţional 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 !!!!

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
UNIVERSITATEA BABEȘ-BOLYAI

Str. Mihail Kogălniceanu nr. 1
Cluj-Napoca, Cluj, România

www.cs.ubbcluj.ro