# Pointer arithmetic

In the addressing system operations with pointers are performed. Which are the ARITHMETIC operations allowed with pointers in COMPUTER SCIENCE ?...

**Answer**: Any operation that makes sense... meaning any operation that expresses as a result a correct location in memory useful as an information for the programmer/processor.

- adding a constant value to a pointer a[7] = *(a+7) – useful for going into memory forth and back relative to a starting address
- subtracting ..... a[-4] , a(-4) ...
- ✗ multiplying 2 pointers ? – No way ... no practical usage !
- ✗ dividing 2 pointers ? - No way ... no practical usage !
- adding/subtracting 2 pointers ?
- ✗ ADDING 2 pointers doesn't make sense !! – it is not allowed
- SUBTRACTING 2 pointers !! does makes sense... q-p = nr. of elements (in C) = nr. of bytes between these 2 addresses in assembly (this can be very useful for determine the length of a memory area).

  a[7] = *(a+7) = *(7+a) = 7[a] - both in C and assembly !

POINTER ARITHMETIC OPERATIONS - ***Pointer arithmetic*** represents the set of arithmetic operations allowed to be performed with pointers, this meaning using arithmetic expressions which have addresses as operands.

Pointer arithmetic contains ONLY 3 operations that are possible:

1). Subtracting two addresses
Address – address = ok (q-p = subtraction of 2 pointers = sizeof(array) in C, the number of bytes between these 2 addresses in assembly)

2). Adding a numerical constant to a pointer
Address + numerical constant (identification of an element by indexing – a[7]) , q+9

3). Subtracting a numerical constant from a pointer
Adress - numerical constant - a[-4] , p-7;
*(a-4) - useful for reffering array elements

- subtraction of 2 pointers = SCALAR VALUE (constant)
- adding a constant to a pointer → a POINTER !!
- subtracting a constant from a pointer → a POINTER !!

! ADDING TWO POINTERS IS NOT ALLOWED !!!

p+q = ???? (allowed in NASM...sometimes...) – but it doesn't mean in the end as we shall see that this is "a pointer addition" !!!

How do we make in NASM the difference between the address of a variable and its contents ?

Var – invoked like that it is an address (offset) ; [var] – is its contents
[] = the dereferencing operator !! (like *p in C)

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

mov [EBX+ECX*2 + a+b-7], bx - not allowed ! syntax error ! because of "a+b" invalid effective address – impossible segment base multiplier

sub [EBX+ECX*2 + a-b-7], eax – ok, because a-b is a correct pointers operation !!!

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

[EBX+ECX*2 + a-b-7]
SIB const.

mov eax, [EBX+ECX*2+(-7)] – ok.

## L-value vs. R-value. LHS vs. RHS of an assignment.

Assignment: i:=i+1 (always a pointer, an address) LHS vs. RHS

Address of I ← value of I + 1

LHS(i) = Address of I := RHS(i) =(the contents from the address i) + 1

LHS (Left Hand Side of an assignment = L-value = address) := RHS
(Right Hand Side of an assignment = R-Value = CONTENTS !!)

---

The syntax of most programming languages define assignment as:

Symbol := expression_value
Identifier := expression (usually in 99% of the cases)

C++ and ASSEMBLY !!! allow a much more general syntax:

Address_computation_expression := expression (the most general)
(mov [ebx+2*EDX+v-7], a+2)

Dereferencing (extracting the value from a given address) is usually implicit (depending on the context !) in 99% of the cases. Exception: BLISS language, where dereferencing must always be explicitly specified; i←*i+1 (also we have some similar situations in Algol68)

---

C++ reference variables are a special type of variables offering 3 kinds of usages:

1) Int& j = i; // j becomes an ALIAS for i
2) Passing variables **by reference** at subprograms call:
   float f(int&x, y);…..
3) Returning L-values as a result of functions calls :

Int& f(x,i) {….return v[i];} – Function f returns a LHS (Left Value)
F(a,7) = 79;  meaning that v[7]=79 !!!

Independently of these, the conditional ternar operator may be used as a L-value:

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