Practical Foundations of Debugging

Chapter 3

Bytes, words, double words and pointers to memory

Designed, written and illustrated by Dmitry Vostokov

Using hexadecimal numbers

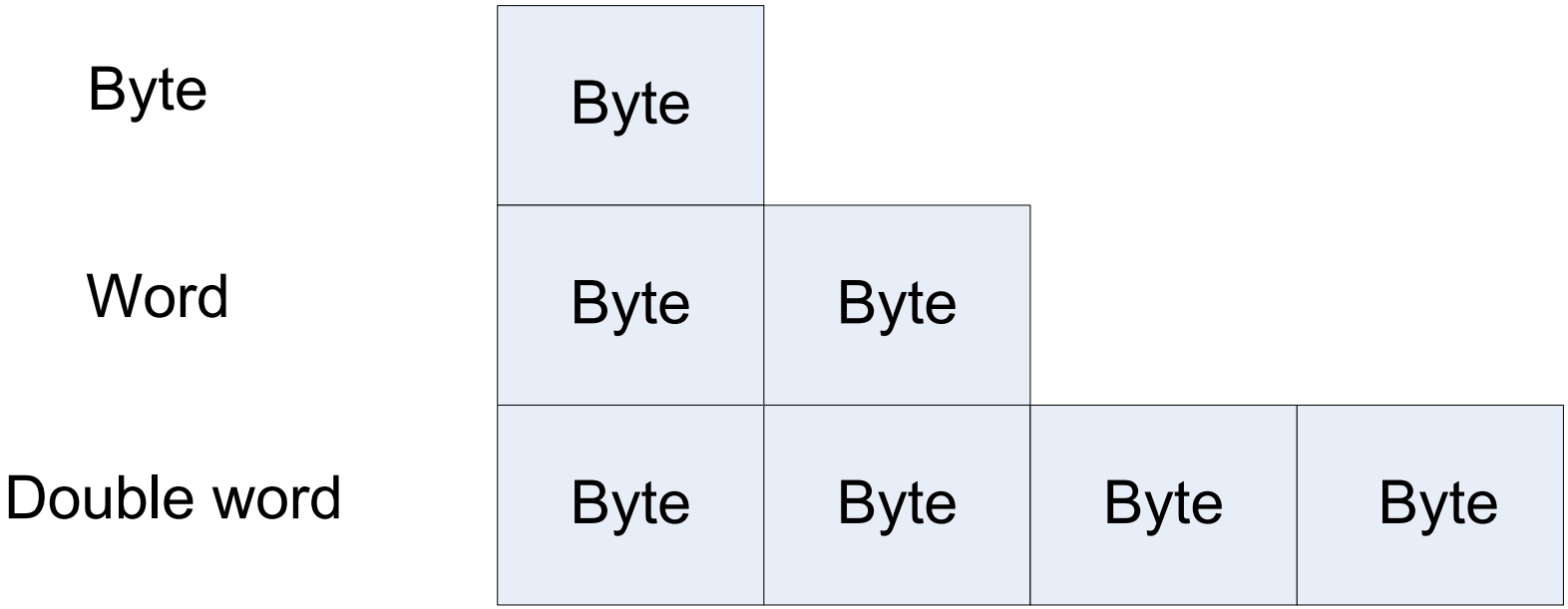
WinDbg disassembly:

- mov [a], 12
- mov [a], 0xC

In C language:

- a = 12;
- a = 0xC;

Bytes, words and double words



Bits, bytes, words and double words

- Bit

12_{dec} 1100_{bin} value = 0 or 1

- Byte (unsigned char)

8 bits 00001100_{bin} $0C_{hex}$

value 0_{dec} - 255_{dec} or 0_{hex} - FF_{hex}

- Word (unsigned short)

16 bits 0000000000001100_{bin} $000C_{hex}$

value 0_{dec} - 65535_{dec} or 0_{hex} - $FFFF_{hex}$

- Double word (unsigned int, unsigned)

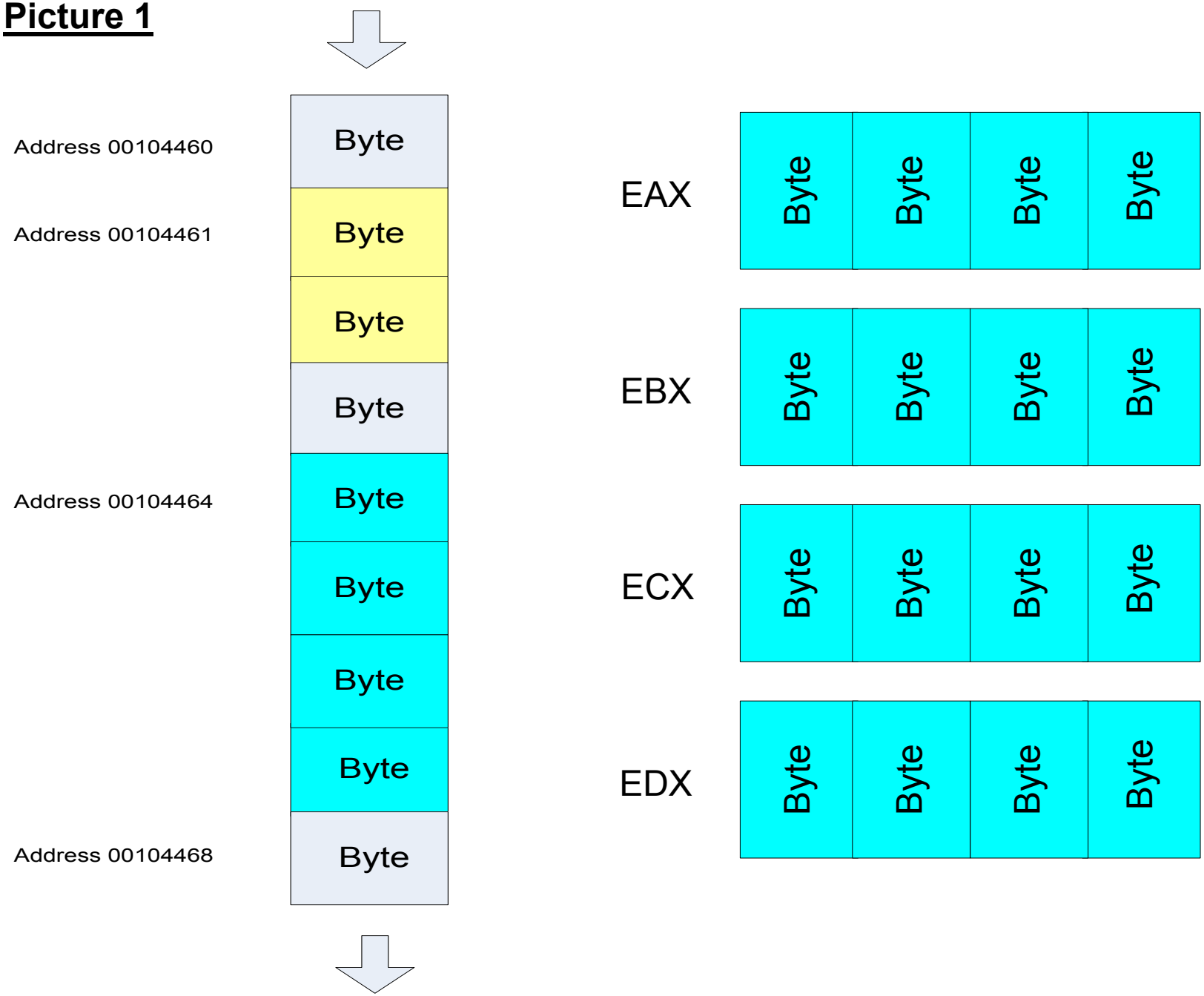
32 bits $00000000000000000000000000001100_{bin}$

value 0_{dec} - 4294967295_{dec} or 0_{hex} - $FFFFFFFF_{hex}$

Memory layout (Picture 1)

- Minimal addressable element is memory **byte**
- Maximum addressable element is **double word**
- All registers are 32-bits and can contain double word value

Picture 1


Address 00104460
Address 00104461
Address 00104464
Address 00104468
Byte
EAX
EBX
ECX
EDX

Pointers revisited

- Pointer is a memory cell or a register that contains the address of another memory cell. Has its own address (as any memory cell)
- Pointers are always 32-bit (Windows)
- Memory cell can be byte, word or double word
- Pointer to byte (byte ptr), pointer to word (word ptr), pointer to double word (dword ptr)

Addressing types (Picture 6)

- byte ptr [eax] (Pictures 2,3)

mov byte ptr [eax], 0xFF

- word ptr [eax]

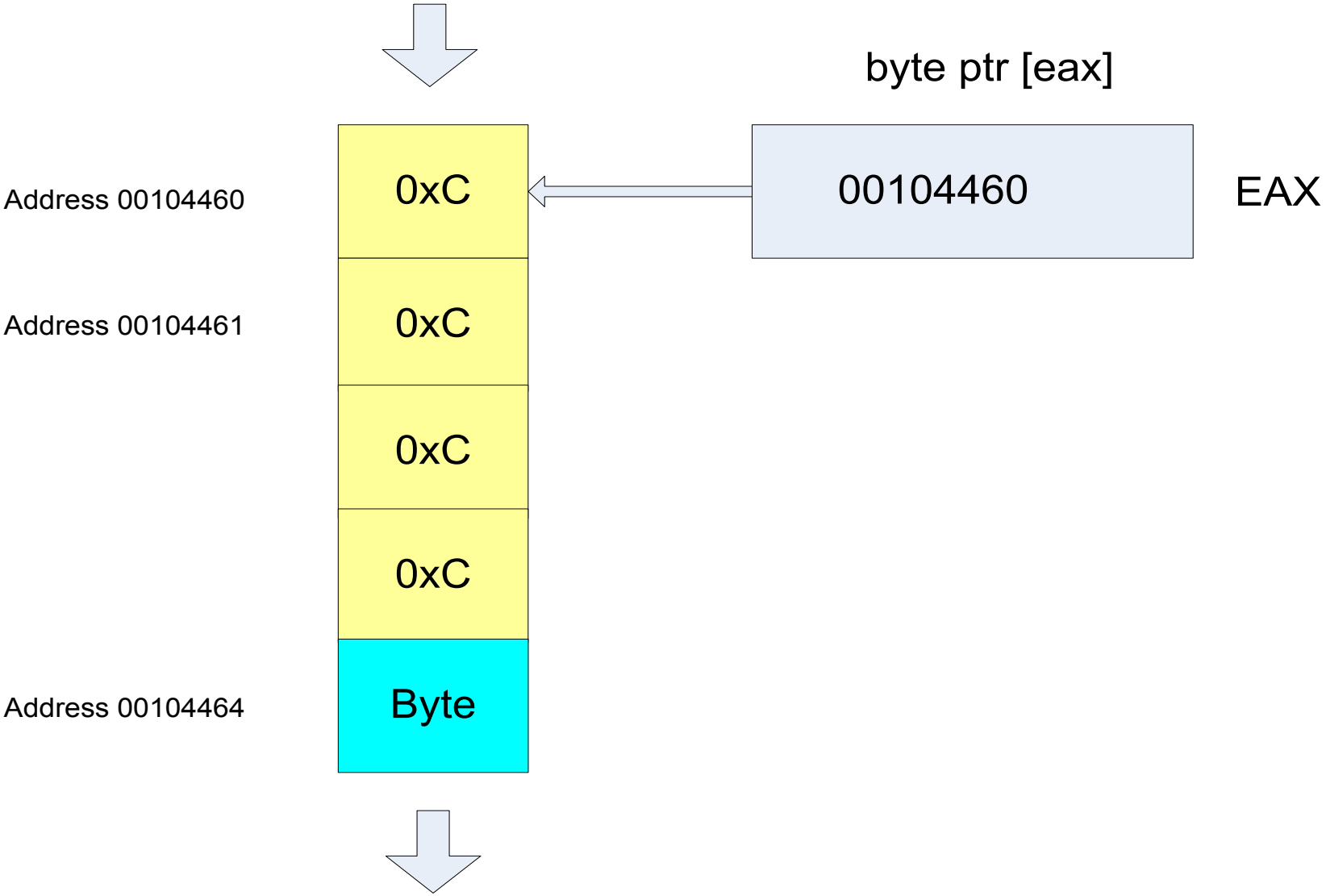
mov word ptr [eax], 0xFFFF

- dword ptr [eax] (Pictures 4,5)

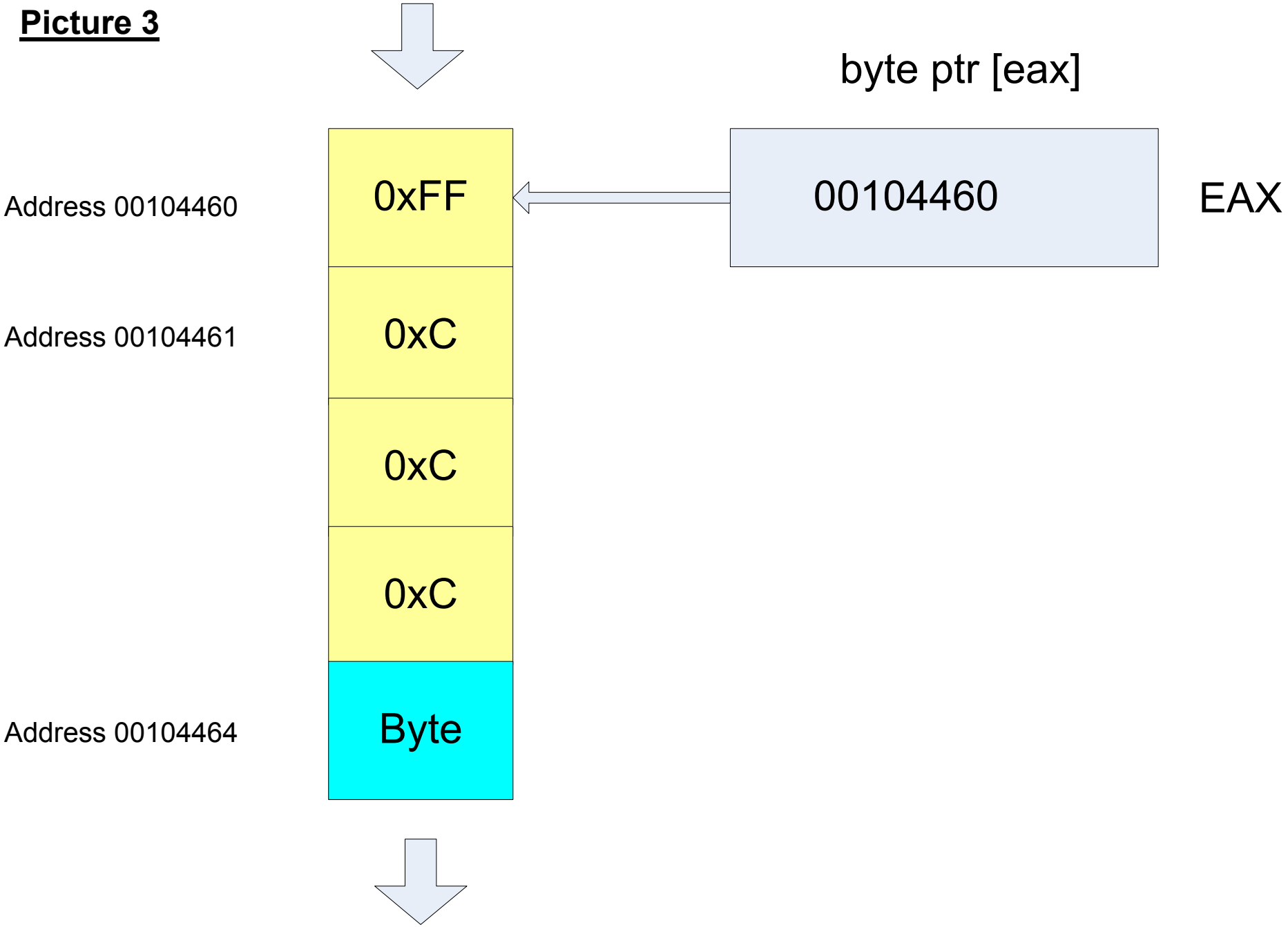
mov dword ptr [eax], 0xFFFFFFFF

Picture 2

mov byte ptr [eax], 0xFF

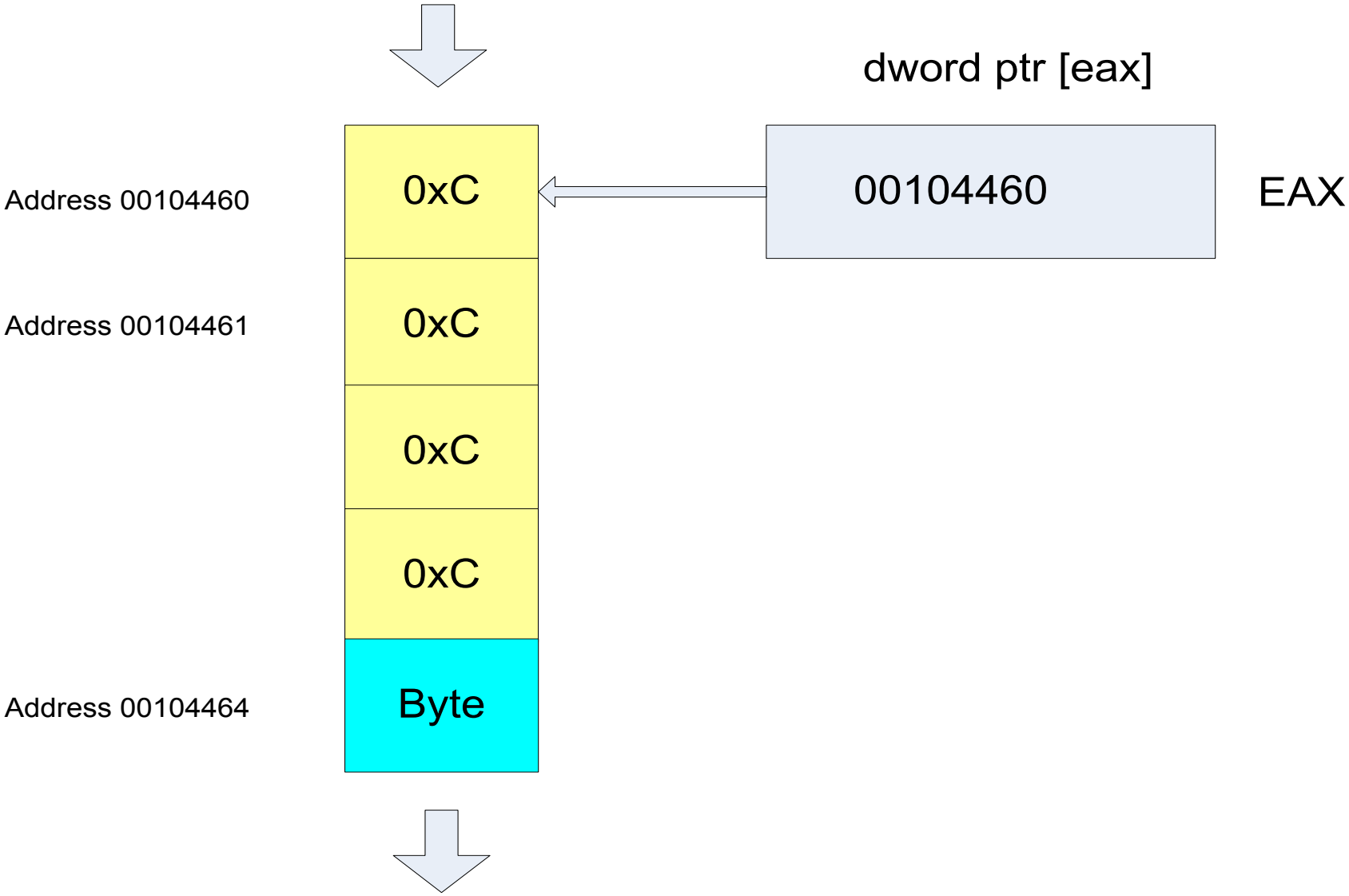


Picture 3

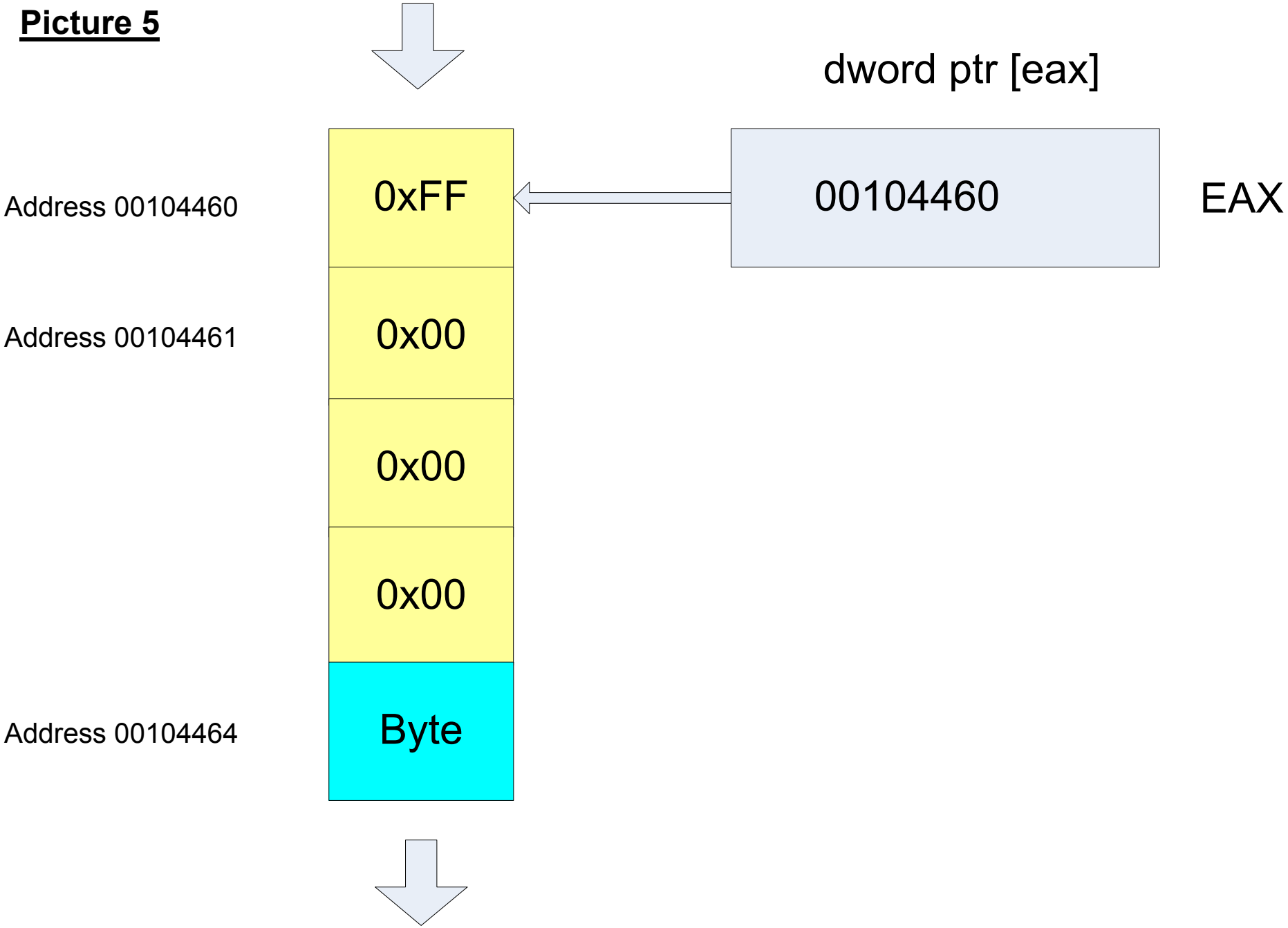

byte ptr [eax]
0xFF
00104460
EAX
Address 00104460
0xC
Address 00104461
0xC
0xC
Byte
Address 00104464

Picture 4

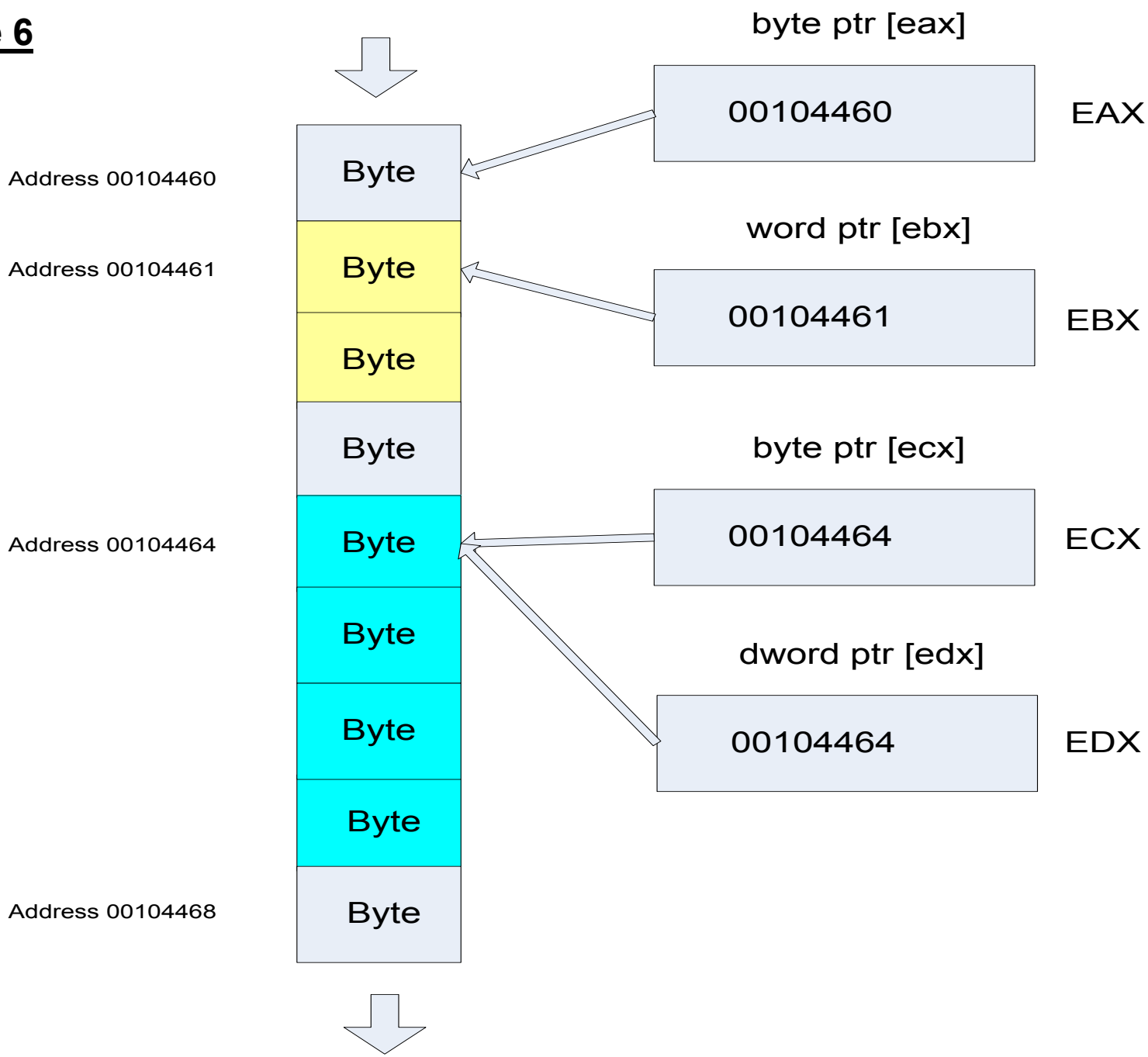
mov dword ptr [eax], 0xFF



Picture 5


dword ptr [eax]
Address 00104460
0xFF
00104460
EAX
Address 00104461
0x00
0x00
0x00
Address 00104464
Byte

Picture 6


byte ptr [eax]
00104460
EAX
Address 00104460
Byte
word ptr [ebx]
Address 00104461
Byte
00104461
EBX
Byte
Byte
byte ptr [ecx]
Address 00104464
Byte
00104464
ECX
Byte
dword ptr [edx]
Byte
00104464
EDX
Byte
Address 00104468
Byte

Registers revisited

EAX, **EBX**, **ECX**, **EDX** can be used as pointers

Additionally:

- **EAX** and **EDX** contains the multiplication result after executing **imul** instruction
- **ECX** is often used as a loop counter

for (int i = 0; i < N ; ++i)

NULL pointers

- Addresses 0x00000000 – 0x0000FFFF are specifically made inaccessible

```
mov eax,   0xF
mov [eax], 1     // Access violation
```

Invalid pointers

- NULL pointers
- Pointers to inaccessible memory
- Pointers pointing to “random” memory
- Uninitialized pointers
- Dangling pointers

“Pointers to memory” Project – Memory Layout and Registers

- Two memory addresses (locations):
  “a” and “b”

int a, b;

- Two memory addresses (locations):
  “pa” and “pb”

int *pa, *pb;

- **int *pa** is a pointer to **int** (memory cell **pa** contains the address of another memory cell that contains an integer value)

Pointer initialization

```
• int b;            // uninitialized variable
int *pb;            // uninitialized pointer
pb = &b;            // [pb] contains the address b

• int b = 12;       // initialized variable
int *pb = &b;       // initialized pointer
```

- Pointers are also variables

Memory segments

- Different places in memory (address ranges)
- .DATA

All initialized global and static variables (including pointers)

- .BSS (block storage space)

All uninitialized global and static variables (including pointers)

Why different memory segments? (Picture 7)

- int array[1000000]; // 4Mb

Program size on disk is 32Kb

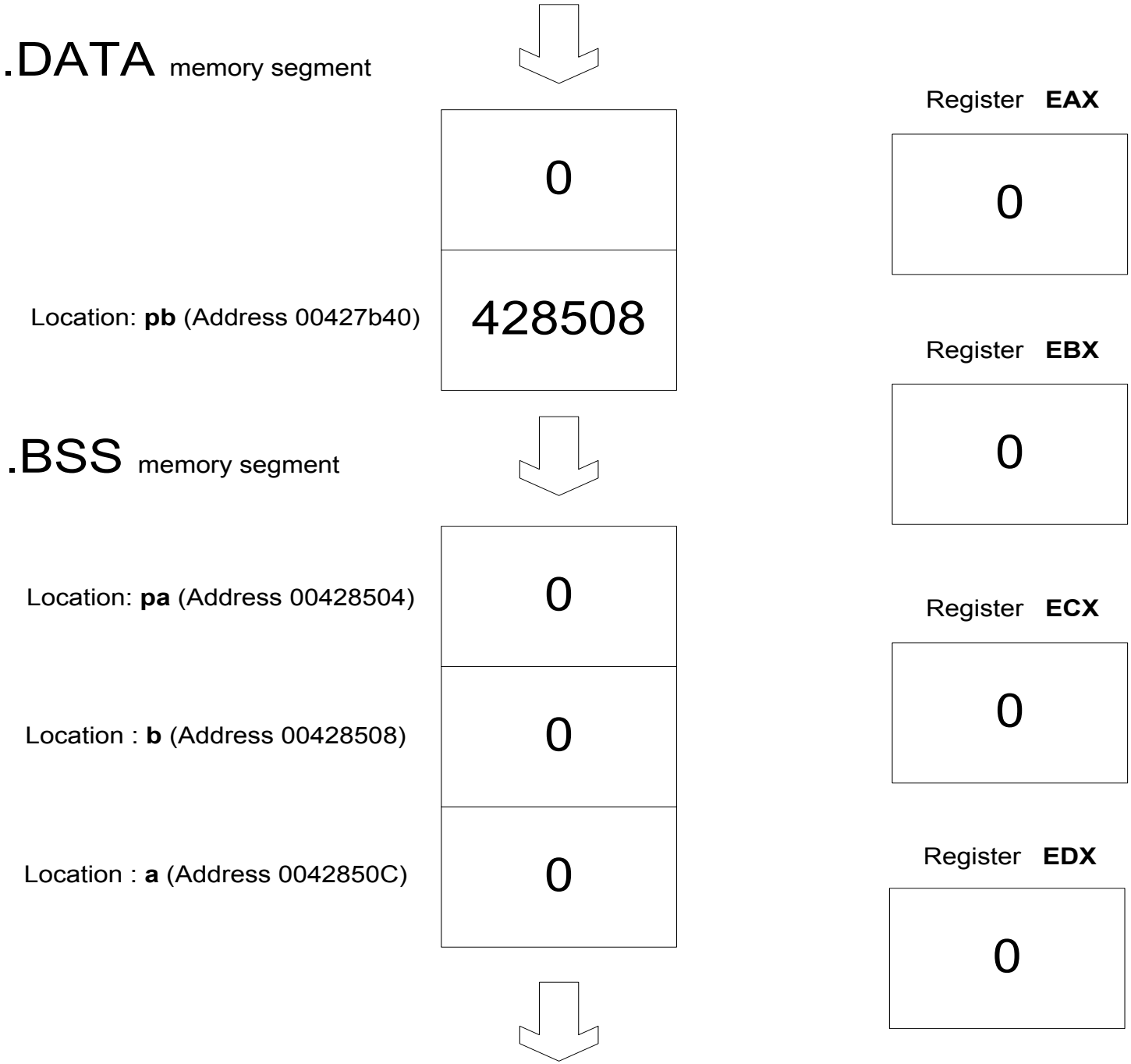
array is put into .BSS segment that contains only information about its size

- int array[1000000] = { 12 };

Program size on disk 3.84Mb

array is put into .DATA segment and contains { 12, 0, 0, 0, 0 … }

Picture 7


.DATA memory segment
0
Location: pb (Address 00427b40)
428508
.BSS memory segment
Location: pa (Address 00428504)
0
Location : b (Address 00428508)
0
Location : a (Address 0042850C)
0
Register EAX
0
Register EBX
0
Register ECX
0
Register EDX
0

More pseudo notation

- [a] means contents at the address a
- [eax] means contents at the address stored in eax

(eax is a pointer)

- *[pa] means contents at the address stored at the address pa, called dereferencing a pointer whose address is pa

```
int *pa = &a;
int b = *pa;
```

“Pointers” Project – Calculations (Picture 8)

[pa] := address a

*[pa] := 1 *; [a] = 1*

*[pb] := 1 *; [b] = 1*

*[pb] := *[pb] + *[pa]
; [b] = 2

```
lea eax,   a
mov [pa],  eax

mov eax,   [pa]
mov [eax], 1

mov ebx,   [pb]
mov [ebx], 1

mov ecx, [eax]
add ecx, [ebx]
mov [ebx], ecx
```

WinDbg output:

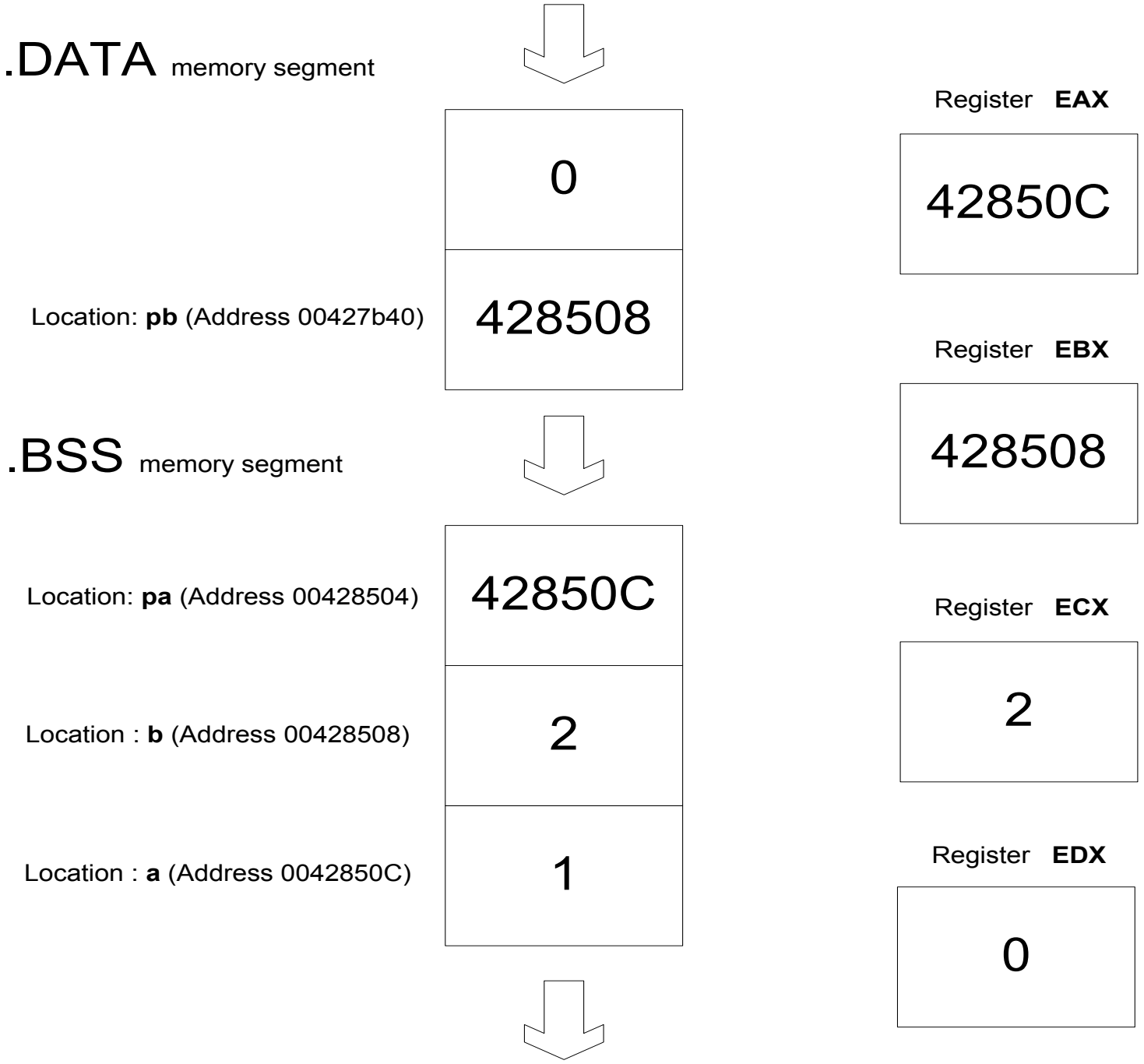
```
004126ce 8d057c854200     lea     eax,[MemoryPointers!a (0042857c)]
004126d4 a374854200       mov     [MemoryPointers!pa (00428574)],eax

004126d9 a174854200       mov     eax,[MemoryPointers!pa (00428574)]
004126de c60001           mov     byte ptr [eax],0x1

004126e1 8b1d407b4200     mov     ebx,[MemoryPointers!pb (00427b40)]
004126e7 c60301           mov     byte ptr [ebx],0x1

004126ea 8b08             mov     ecx,[eax]
004126ec 030b             add     ecx,[ebx]
004126ee 890b             mov     [ebx],ecx
```

Picture 8


.DATA memory segment
0
Location: pb (Address 00427b40)
428508
.BSS memory segment
Location: pa (Address 00428504)
42850C
Location : b (Address 00428508)
2
Location : a (Address 0042850C)
1
Register EAX
42850C
Register EBX
428508
Register ECX
2
Register EDX
0

What's next?

- More instructions
- Instruction pointer
- Disassemble a program written in C