Practical Foundations of Debugging

Chapter 1

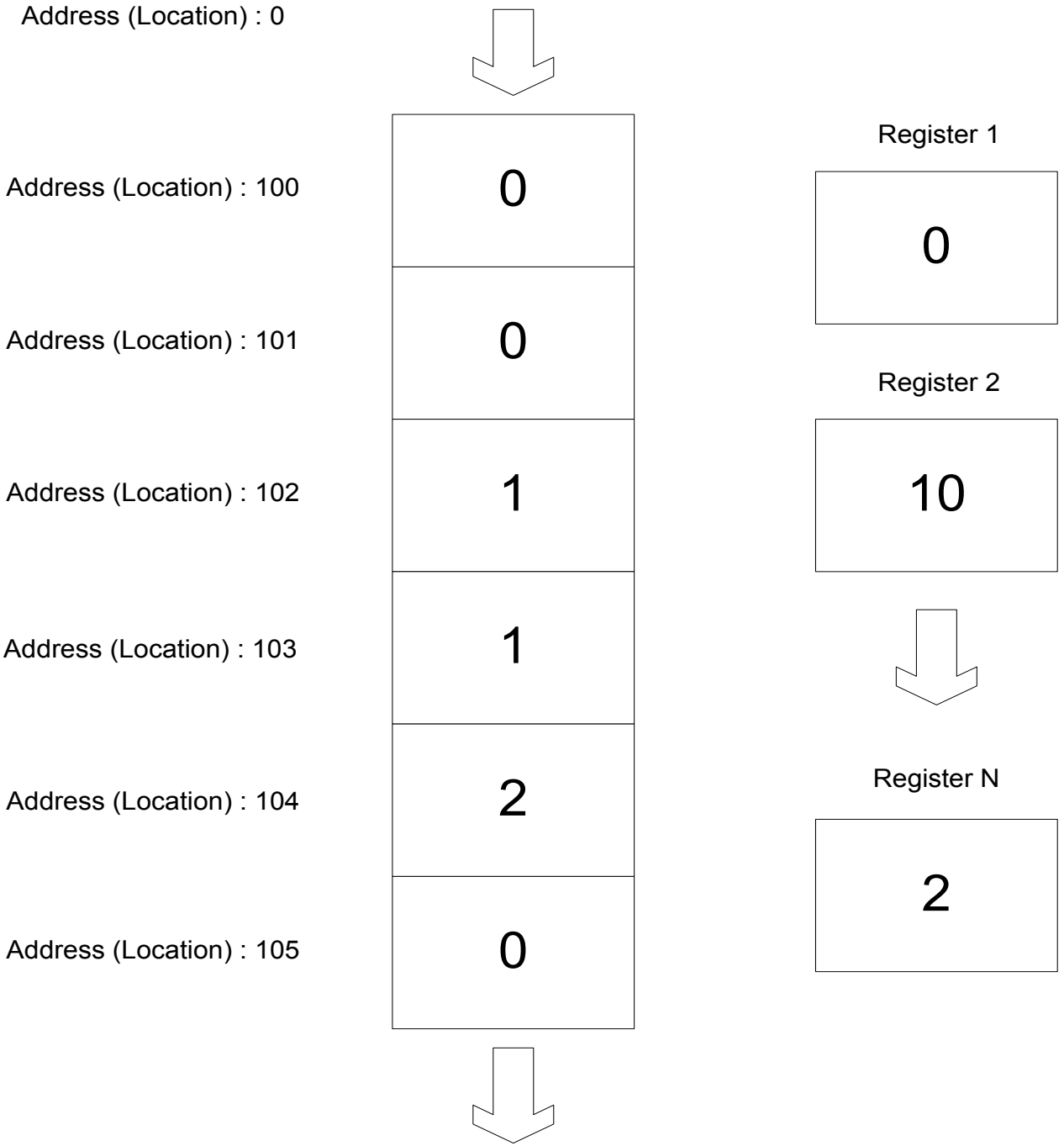
Memory, Registers and Simple Arithmetic

Designed, written and illustrated by Dmitry Vostokov

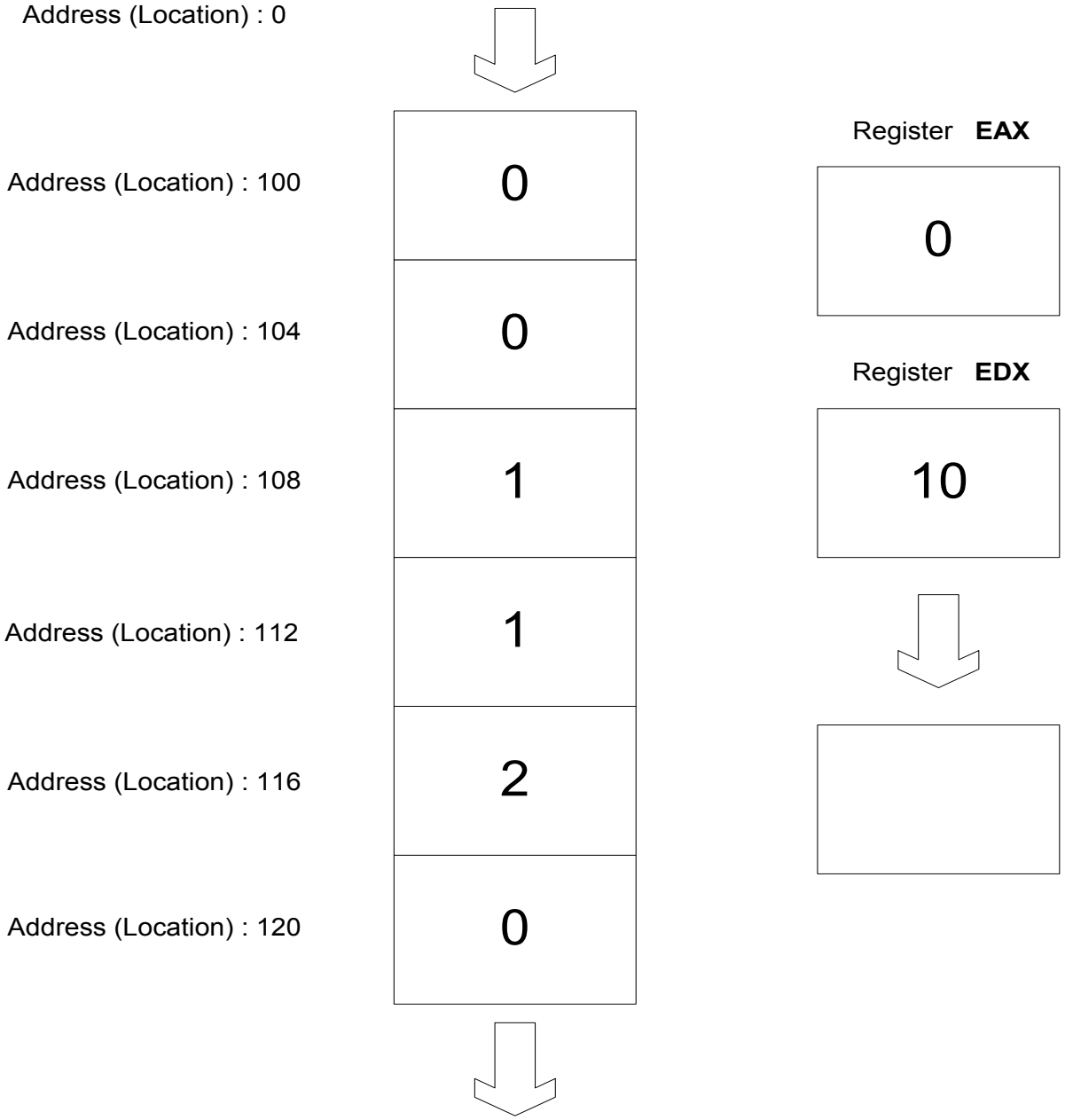
Memory and Registers inside idealized computer (Picture 1)

- Computer memory consists of a sequence of memory cells and each cell has unique address (location). Every cell contains a “number”. We refer to these “numbers” as contents at the address (location).
- Think of registers as the standalone memory cells. The name of the register is its address.

Picture 1


Address (Location) : 0
Address (Location) : 100
0
Address (Location) : 101
0
Address (Location) : 102
1
Address (Location) : 103
1
Address (Location) : 104
2
Address (Location) : 105
0
Register 1
0
Register 2
10
Register N
2

Memory and Registers inside Intel 32-bit PC

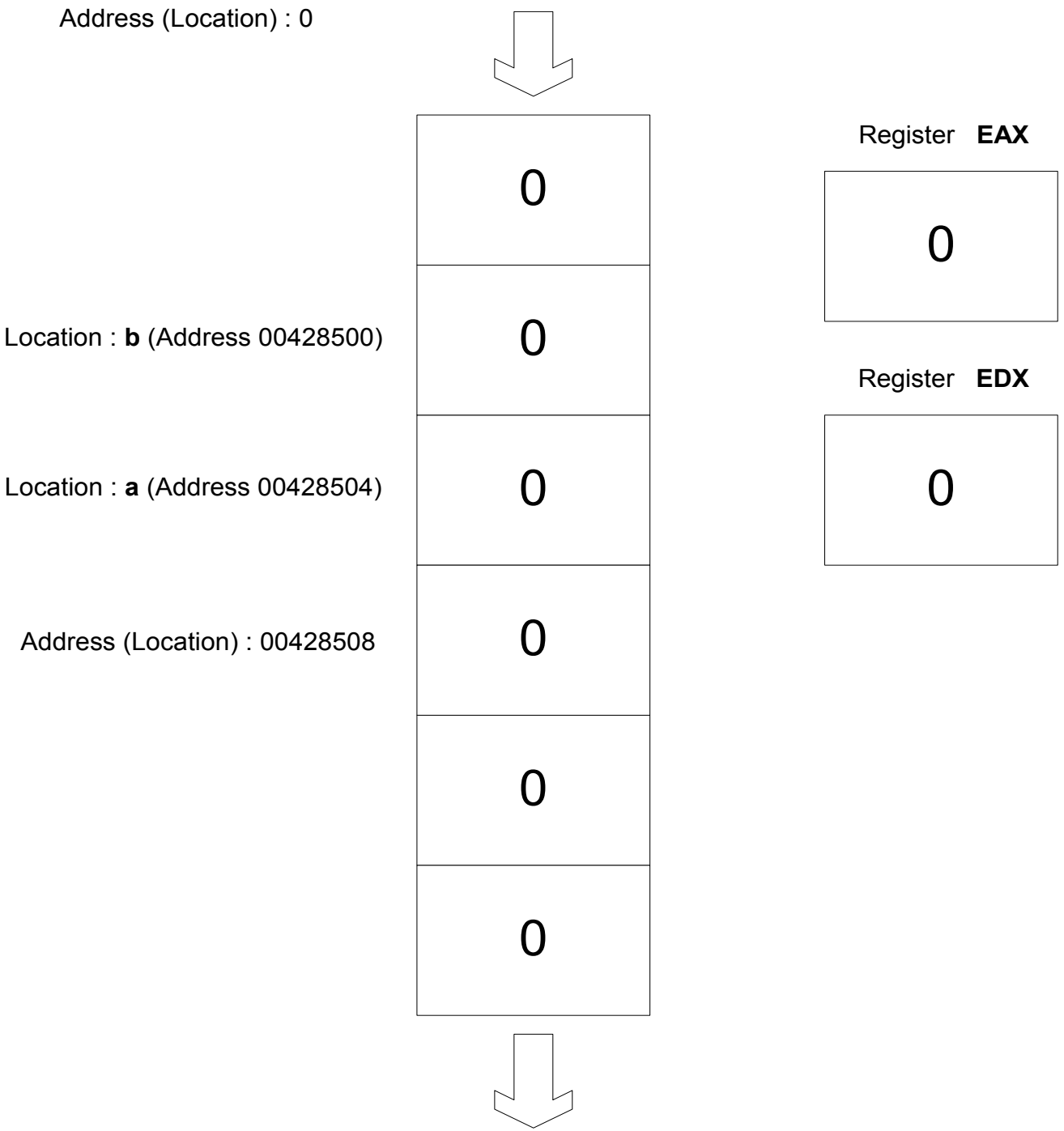


“Arithmetical” Project – Memory Layout and Registers (Picture 2)

- Two memory addresses (locations):
  “a” and “b”. We can think about “a” and “b” as names of addresses (locations)
- Notation [a] means contents at the memory address (location) “a”
- Registers *EAX* and *EDX.*
- In C we declare memory locations “a” and “b” as:

static int a, b;

Picture 2


Address (Location) : 0
0
Location : **b** (Address 00428500)
0
Location : **a** (Address 00428504)
0
Address (Location) : 00428508
0
0
0
Register **EAX**
0
Register **EDX**
0

“Arithmetical” Project - Calculations

[a] := 1

[b] := 1

[b] := [b] + [a] *; [b] = 2*

[b] := [b] * 2 *; [b] = 4*

Computer program

- We can think of a computer program as a sequence of instructions for manipulation of contents of memory cells and registers
- For example, addition: add the contents of memory cell №12 to the contents of memory cell №14.

  In notation [14] := [14] + [12]
- Because memory cells contain “numbers” we start with simple arithmetic

Assigning numbers to memory cells

- [a] := 1
- “a” means location (address) of the memory cell, the name of location (address) 00428504

  [a] means contents at the address “a”
- In C language “a” is called “variable” and we write:

a = 1;

- In Assembler we write:

mov [a], 1

- In WinDbg disassembly output we see:

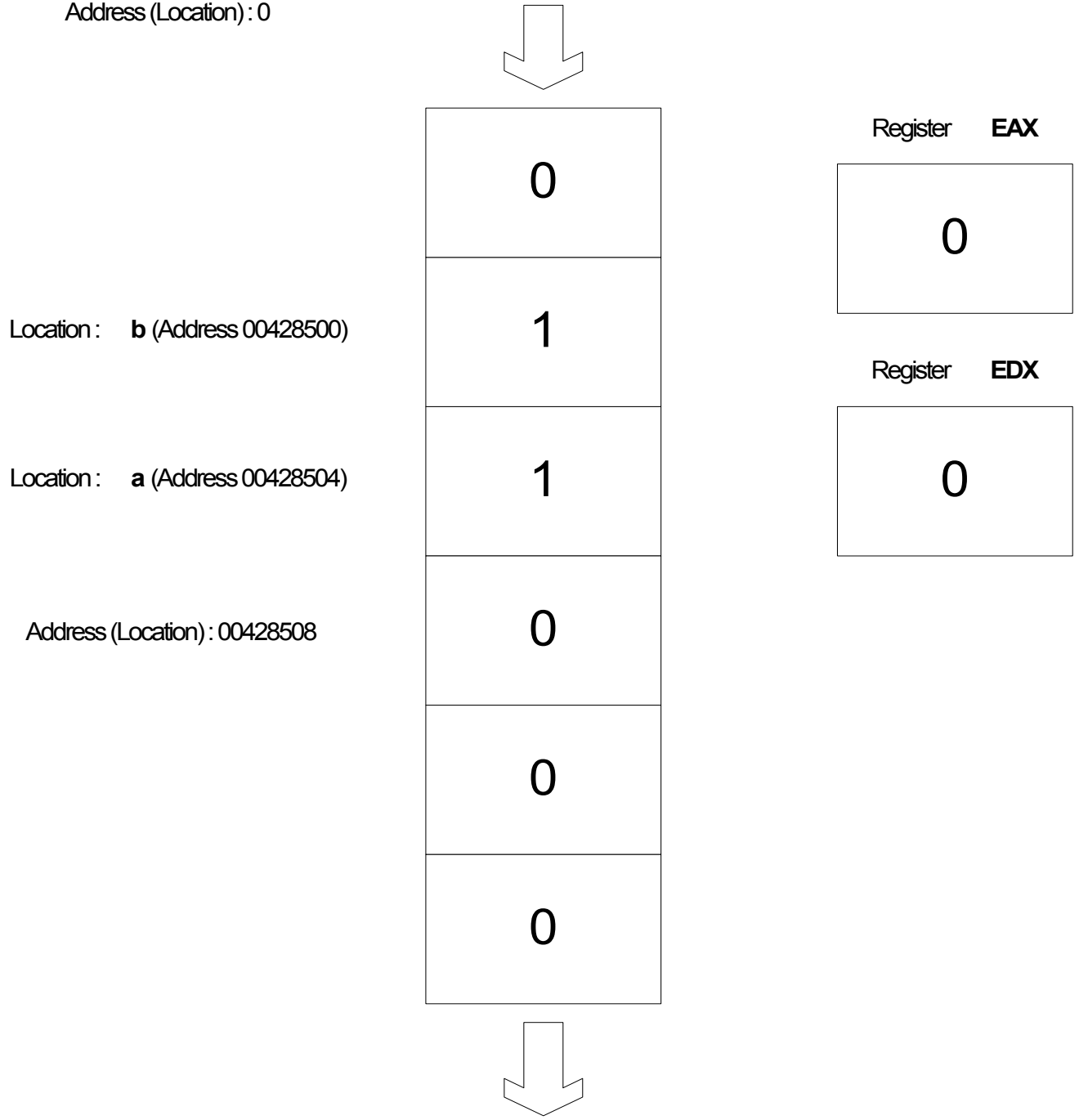
mov dword ptr [ArithmeticProject!a (00428504)], 1

“Arithmetical” Project – Calculations (Picture 3)

```
[a] := 1
[b] := 1
[b] := [b] + [a] ; b = 2
[b] := [b] * 2    ; b = 4
```

```
mov [a], 1
mov [b], 1
```

Picture 3


Address (Location) : 0
0
Location : b (Address 00428500)
1
Location : a (Address 00428504)
1
Address (Location) : 00428508
0
0
0
Register EAX
0
Register EDX
0

Assigning numbers to registers

- register := 1 or register := [a]
- We do not use brackets when refer to register contents
- Latter instruction means assign (copy) the number at the location (address) “a” to a register
- In Assembler we write:

mov eax, 1

mov eax, [a]

- In WinDbg disassembly output we see:

mov eax, [ArithmeticProject!a (00428504)]

Adding numbers to memory cells

- [b] := [b] + [a]
- “a” and “b” mean locations (addresses) “a” and “b”, names of locations (addresses) 00428504 and 00428500. [a] and [b] mean contents at the adresses “a” and “b”, simply some numbers
- In C language we write:

b = b + a;

- In Assembler we use instruction **add**
- We cannot use both memory addresses in one step (instruction): **add [b], [a]**
  We can only use **add [b], register**
  Here **register** is like a temporary memory cell:

register := [a]

[b] := [b] + register

- In Assembler we write:

mov eax, [a]

add [b], eax

- In WinDbg disassembly output we see:

mov eax,[ArithmeticProject!a (00428504)]

add [ArithmeticProject!b (00428500)],eax

“Arithmetical” Project – Calculations (Picture 4)

```
[a] := 1
[b] := 1
[b] := [b] + [a] ; [b] = 2
                 ; eax = 1
[b] := [b] * 2   ; [b] = 4
```

```
mov [a], 1
mov [b], 1
mov eax, [a]
add [b], eax
```

Picture 4


Address (Location) : 0
0
2
1
0
0
0
Location : **b** (Address 00428500)
Location : **a** (Address 00428504)
Address (Location) : 00428508
Register **EAX**
1
Register **EDX**
0

Incrementing (Decrementing) numbers in memory cells and registers

- [a] := [a] + 1 ([a] := [a] – 1)
- Means increment (decrement) number at location (address) “a”
- In C language we write:

a = a + 1; or **++a;** or **a++;**

b = b – 1; or **--b;** or **b--;**

- In Assembler we use instructions **inc** and **dec**
- In Assembler we write:

```
inc     [a]
Inc     eax
dec     [a]
dec     eax
```

- In WinDbg disassembly output we see:

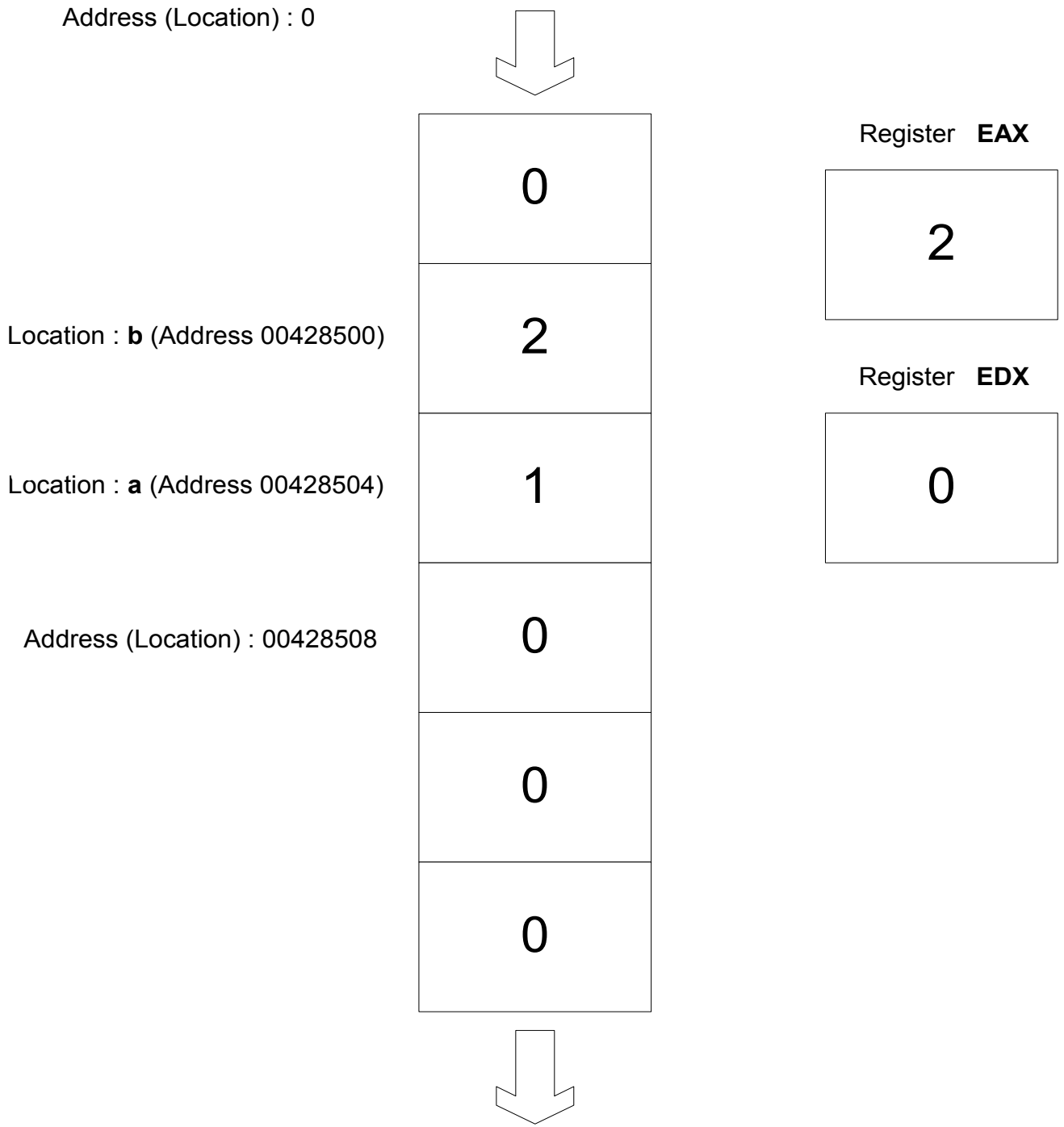
```
inc     eax
```

“Arithmetical” Project – Calculations (Picture 5)

```
[a] := 1
[b] := 1
[b] := [b] + [a]  ; b = 2
                  ; eax = 1
eax := eax + 1
                  ; eax = 2
[b] := [b] * 2    ; [b] = 4
```

```
mov [a], 1
mov [b], 1
mov eax, [a]
add [b], eax
inc eax
```

Picture 5


Address (Location) : 0
0
2
1
0
0
0
Location : **b** (Address 00428500)
Location : **a** (Address 00428504)
Address (Location) : 00428508
Register **EAX**
2
Register **EDX**
0

Multiplying numbers

- [b] := [b] * 2
- Means multiply the number at the location (address) “b” by 2
- In C language we write:

b = b * 2; or **b *= 2;**

In Assembler we use instruction **imul** (integer multiply)

- In Assembler we write:

imul [b]

mov [b], eax

Means [b] := [b] * eax, so we have to put 2 into eax, but we already have 2 in eax

Result of multiplication is put into registers eax and edx

- In WinDbg disassembly output we see:

imul dword ptr [ArithmeticProject!b (00428500)]

mov [ArithmeticProject!b (00428500)],eax

Multiplication and registers

- Why the result of multiplication occupies two registers eax and edx?
- Each register or memory cell can contain the number between -2147483648 and 2147483647
- If we multiply 2 by 2 the result can be put into one register eax
- If we multiply 2147483647 by 2147483647 we get 4611686014132420609. The result is too big to fit into one register or memory cell.
- We can think of edx:eax pair as two memory cells joined together to hold the multiplication result.

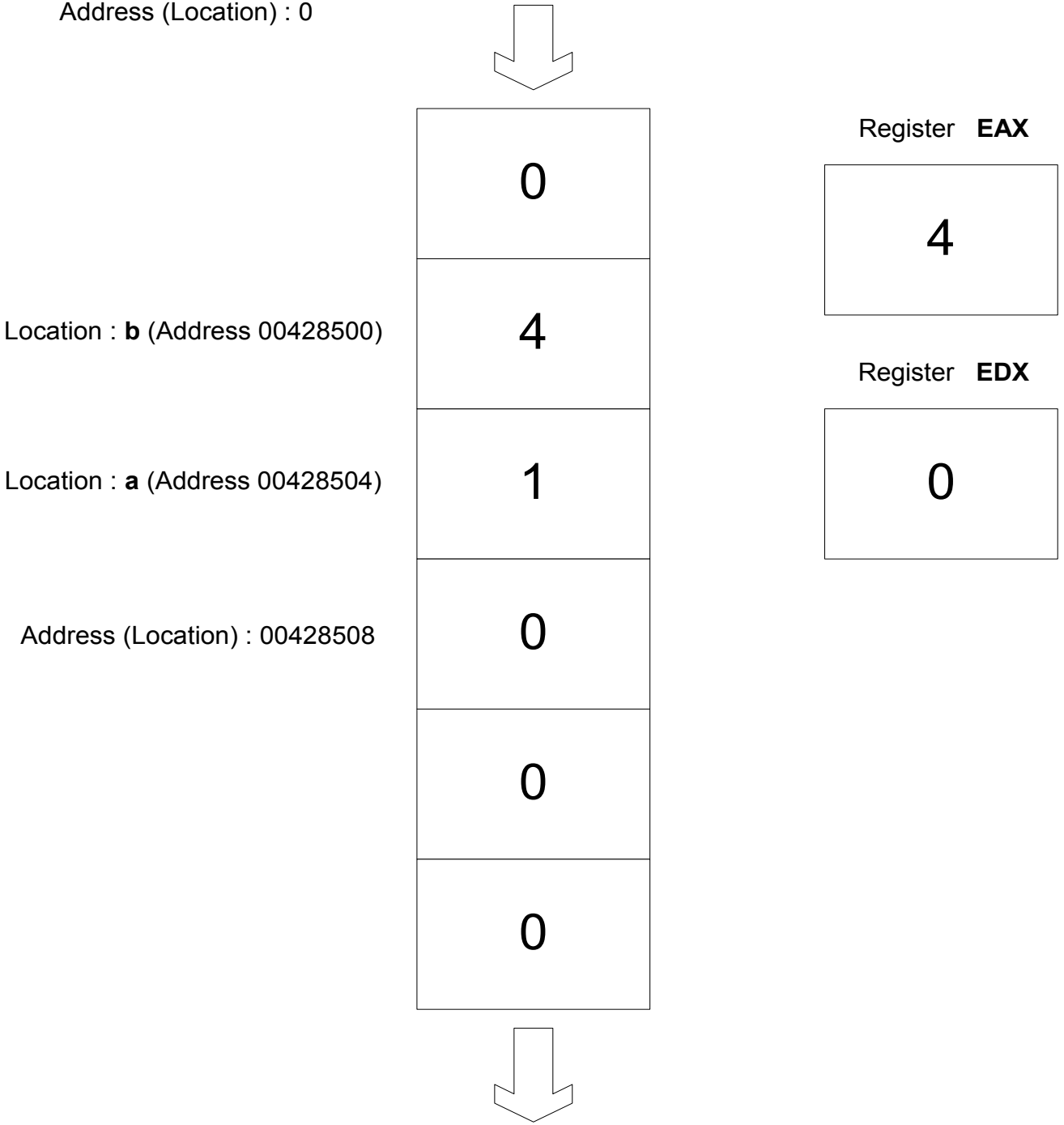
“Arithmetical” Project – Calculations (Picture 6)

```
[a] := 1
[b] := 1
[b] := [b] + [a] ; [b] = 2
                   ; eax = 1
eax := eax + 1
                   ; eax = 2
[b] := [b] * 2     ; [b] = 4
```

```
mov [a], 1
mov [b], 1
mov eax, [a]
add [b], eax
inc eax

imul [b]
mov [b], eax
```

Picture 6


Address (Location) : 0
0
4
1
0
0
0
Location : **b** (Address 00428500)
Location : **a** (Address 00428504)
Address (Location) : 00428508
Register **EAX**
4
Register **EDX**
0

What's next?

- We will review today's material by looking at the disassembly output of the very simple program written in C.
- Representation of numbers. Hexadecimal notation.
- The concept of the "pointer".
- We will rewrite our "arithmetical" project using pointers.