Smith College
Computer Science

CSC231—Assembly

Week #8 — Spring 2017

Dominique Thiébaut
dthiebaut@smith.edu

The **LOOP** Instruction

loop

loop label

```
loop        label
```

```
x       dd      1
sum     dd      0

        mov     ecx, 10
addUp:  mov     eax, dword[x]
        add     dword[sum], eax
        inc     dword[x]
        loop    addUp                   ;ecx<—ecx-1
                                        ;if ecx!=0,
                                        ; goto addUp
```

loop

loop label

```
loop         label
```

```
x         dd        1
sum       dd        0

          mov       ecx, 10
addUp:    mov       eax, dword[x]
          add       dword[sum], eax
          inc       dword[x]
          loop      addUp               ;ecx<—ecx-1
                                        ;if ecx!=0,
                                        ; goto addUp
```

Label

loop

loop label

```
loop      label
 ;ecx <— ecx-1
 ;if ecx!= 0,
 ;   goto label
 ;else continue
```

```
x       dd      1
sum     dd      0
```

Label

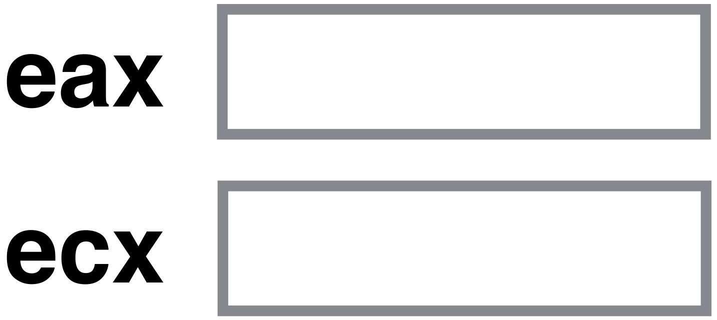
```
        mov     ecx, 10
addUp:  mov     eax, dword[x]
        add     dword[sum], eax
        inc     dword[x]
        loop    addUp             ;ecx<—ecx-1
                                  ;if ecx!=0,
                                  ; goto addUp
```

Labels

```
_start:     mov     eax, 4

            mov     ecx, 10
for1:       …
            …
            loop    for1

for2:       …
            …
            loop    for2
```

- Start with a **letter**
- End with a **colon** (when declared)
- Represent an **address** in the code section
- Must be **unique** in program

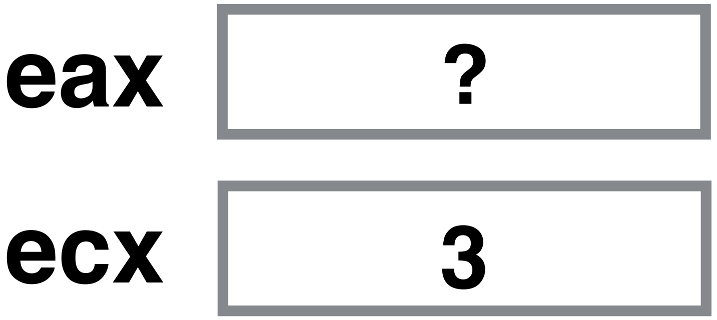
Tracing
One Example



eax

ecx

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for                 ;ecx<—ecx-1
                                    ;if ecx!=0,
                                    ;  goto for
```



eax ?

ecx 3

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for              ;ecx<—ecx-1
                                 ;if ecx!=0,
                                 ; goto for
```

eax	1
ecx	3

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for             ;ecx<—ecx-1
                                ;if ecx!=0,
                                ;  goto for
```

1

eax	1
ecx	3

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for                ;ecx<—ecx-1
                                   ;if ecx!=0,
                                   ;  goto for
```

```
1
```

eax	~~1~~ 2
ecx	3

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for          ;ecx<—ecx-1
                             ;if ecx!=0,
                             ; goto for
```

1

eax ~~1~~ 2

ecx ~~3~~ 2

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for          ;ecx<—ecx-1
                             ;if ecx!=0,
                             ; goto for
```

12

eax ~~1~~ 2

ecx ~~3~~ 2

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for                 ;ecx<—ecx-1
                                    ;if ecx!=0,
                                    ; goto for
```

12

eax ~~1~~ 2

ecx ~~3~~ 2

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for                 ;ecx<—ecx-1
                                    ;if ecx!=0,
                                    ; goto for
```

12

eax	~~1~~ ~~2~~ 3
ecx	~~3~~ 2

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for                 ;ecx<—ecx-1
                                    ;if ecx!=0,
                                    ; goto for
```

12

eax ~~1~~ ~~2~~ 3

ecx ~~3~~ ~~2~~ 1

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for          ;ecx<—ecx-1
                             ;if ecx!=0,
                             ; goto for
```

123

eax	~~1~~ ~~2~~ 3
ecx	~~3~~ ~~2~~ 1

```
           mov     ecx, 3
           mov     eax, 1
for:       call    _printDec
           inc     eax
           loop    for                    ;ecx<—ecx-1
                                          ;if ecx!=0,
                                          ; goto for
```

123

eax	~~1~~ ~~2~~ ~~3~~ 4
ecx	~~3~~ ~~2~~ 1

```
          mov     ecx, 3
          mov     eax, 1
for:      call    _printDec
          inc     eax
          loop    for               ;ecx<—ecx-1
                                    ;if ecx!=0,
                                    ;  goto for
```

```
123
```

eax ~~1~~ ~~2~~ ~~3~~ 4

ecx ~~3~~ ~~2~~ ~~1~~ 0

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for             ;ecx<—ecx-1
                                ;if ecx!=0,
                                ; goto for
```

```
123
```

eax ~~1~~ ~~2~~ ~~3~~ 4

ecx ~~3~~ ~~2~~ ~~1~~ 0

```
        mov     ecx, 3
        mov     eax, 1
for:    call    _printDec
        inc     eax
        loop    for                 ;ecx<—ecx-1
        ????                        ;if ecx!=0,
                                    ; goto for
```

Example 1
Sum of 1..10

```
; computes sum(1,2, …10)
x       dd      1
sum     dd      0

        mov     ecx, 10
        mov     eax, dword[x]
addUP:  add     dword[sum], eax
        inc     eax
        loop    addUp           ;ecx<—ecx-1
                                ;if ecx!=0,
                                ; goto addUp
        mov     dword[x], eax
```

Do we need eax?

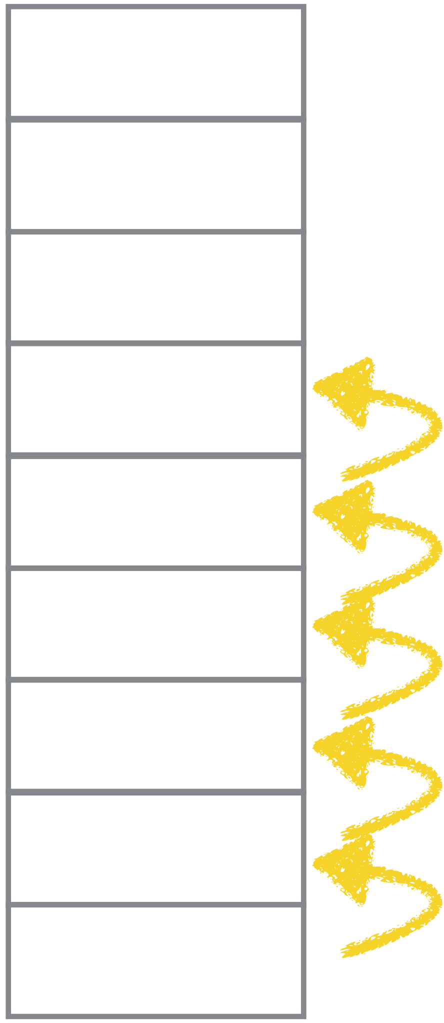
```
; computes sum(1,2, …10)
x       dd      1
sum     dd      0

        mov     ecx, 10
        mov     eax, dword[x]
addUP:  add     dword[sum], eax
        inc     eax
        loop    addUp               ;ecx<—ecx-1
                                    ;if ecx!=0,
                                    ; goto addUp

        mov     dword[x], eax
```

Example 2
Fibonaccis

http://i.dailymail.co.uk/i/pix/2016/07/17/13/1A32203E000005DC-3694326-image-m-17_1468760305397.jpg

```
_start:
        mov   eax, 1    ; fibn
        mov   ebx, 1    ; fibn-1
        call _printDec
        call _println

        mov   ecx, 10-1 ; we printed 1, 9 more to go

for:    mov   edx, ebx
        mov   ebx, eax
        add   eax, edx
        call _printDec
        call _println
        loop for
```

getcopy fib.asm

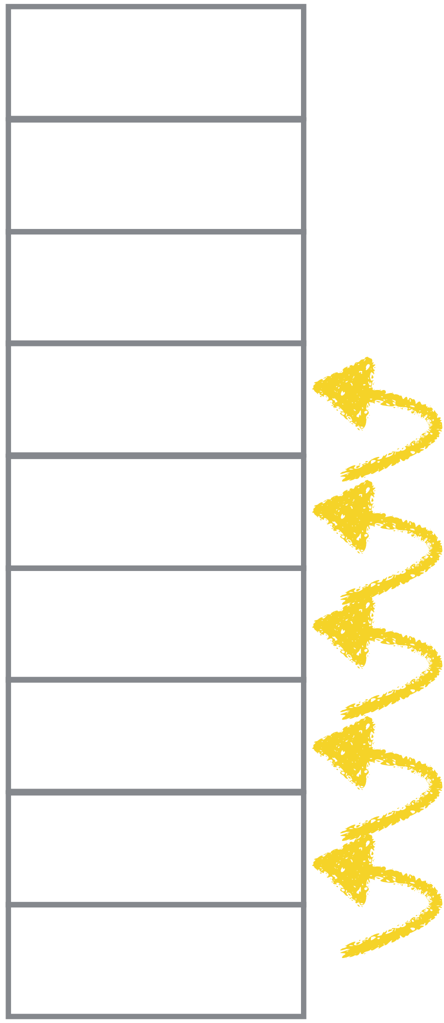


Looping Through Arrays

LOOP INSTRUCTION

Looping Through Arrays

INDIRECT ADDRESSING MODE



Indirect *Addressing Mode*

The **addressing mode** refers to the way the operand of an instruction is generated. We already know *register mode, immediate mode,* and *direct mode.*

Tracing One Example of **Indirect Addressing** (*Base Addressing*)

ebx	???
al	?

Memory

0x1104C	12
0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	1

```
                section .data
A               db    1,3,0xF0,0x3E,0x56
B               db    0x78,0x33,0x12


                 section .text
_start: mov   al, 'z'
        mov   ebx, A
        mov   byte[ebx], 0

        mov   ebx, B
        mov   byte[ebx], al
```

ebx	???
al	'z'

Memory

0x1104C	12
0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	1

```
          section .data
A         db    1,3,0xF0,0x3E,0x56
B         db    0x78,0x33,0x12


          section .text
_start:   mov   al, 'z'
          mov   ebx, A
          mov   byte[ebx], 0

          mov   ebx, B
          mov   byte[ebx], al
```

ebx 11045

al 'z'

Memory

0x1104C	12
0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	1

```
            section .data
A           db    1,3,0xF0,0x3E,0x56
B           db    0x78,0x33,0x12


            section .text
_start: mov   al, 'z'
        mov   ebx, A
        mov   byte[ebx], 0

        mov   ebx, B
        mov   byte[ebx], al
```

ebx 11045

al 'z'

Memory

0x1104C	12
0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	~~1~~0

```
          section .data
A         db    1,3,0xF0,0x3E,0x56
B         db    0x78,0x33,0x12


          section .text
_start: mov   al, 'z'
        mov   ebx, A
        mov   byte[ebx], 0

        mov   ebx, B
        mov   byte[ebx], al
```

ebx	**1104A**
al	**'z'**

Memory

0x1104C	12
0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	~~1~~0

```
          section .data
A         db   1,3,0xF0,0x3E,0x56
B         db   0x78,0x33,0x12


          section .text
_start: mov  al, 'z'
        mov  ebx, A
        mov  byte[ebx], 0

        mov  ebx, B
        mov  byte[ebx], al
```

ebx	**1104A**
al	**'z'**

Memory

0x1104C	12
0x1104B	33
0x1104A	~~78~~ 'z'
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	~~1~~ 0

```
               section .data
A              db    1,3,0xF0,0x3E,0x56
B              db    0x78,0x33,0x12


               section .text
_start:        mov   al, 'z'
               mov   ebx, A
               mov   byte[ebx], 0

               mov   ebx, B
               mov   byte[ebx], al
```

Example 2:
Setting an Array
to All 0s

```
; Array Table contains 10 words
Table dw     1,2,3,4,5,6
      dw     7,8,9,10

       mov    ecx, ____      ;# of elements
       mov    ebx, ____      ;address of
                             ;Table
clear: mov    word[ebx], ____;value to store
       add    ebx,____       ;make ebx point
                             ;to next word
       loop   clear          ;ecx<—ecx-1
                             ;if ecx!=0,
                             ; goto clear
```

Exercises

Problem #1:
Store the first 10 Fibonacci terms in an array of ints (32 bits)

Problem #2:
Given a DNA sequence of 1,000,000 characters stored in an array of bytes, and all characters in uppercase, transform it into its lowercase equivalent. The characters are A, C, G, T and N.

We stopped here last time…

1,000,000 DNA Bases: How fast?

```
        section .data
DNA     db      "AGCTANATTTTAGC…   "
        db      "GGTC… "
        …
        db      "GCCCTTTTAAAA"
N       equ     1000000

        mov     ebx, DNA                ; ebx points to DNA
        mov     ecx, N                  ; ready to loop N times

for:    add     byte[ebx], -'A'+'a'     ; transform char to lowercase
        inc     ebx                     ; ebx points to next byte
        loop    for                     ; loop N times
```

1,000,000 DNA Bases: How fast?

```
N       equ     1000000

        section .bss
DNA     resb    N

        section .text
; some code goes here to fill DNA with actual letters…

        mov     ebx, DNA                ; ebx points to DNA
        mov     ecx, N                  ; ready to loop N times

for:    add     byte[ebx], -'A'+'a'     ; transform char to lowercase
        inc     ebx                     ; ebx points to next byte
        loop    for                     ; loop N times
```

1,000,000 DNA Bases: How fast?

```
    DNA     db      "AGCTANATTTTAGC…   "
            db      "GGTC… "
            …
            db      "GCCCTTTTAAAA"
    N       equ     1000000

1           mov     ebx, DNA                ; ebx points to DNA
1           mov     ecx, N                  ; ready to loop N times

1   for:    add     byte[ebx], -'A'+'a'     ; transform char to lowercase
1           inc     ebx                     ; ebx points to next byte
1           loop    for                     ; loop N times
```

Total # cycles = 2 + 3*1,000,000 = 3,000,002 cycles
Assuming frequency of 1GHz, 1 cycle = 1ns
3,000,0002 ns = 0.003 sec

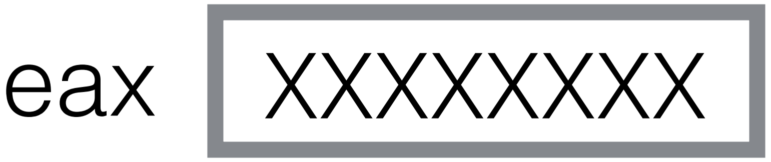
Addressing Modes

- Immediate
- Direct
- Indirect
- Indirect plus Displacement
- Indirect Indexed
- Indirect Indexed plus Displacement

- **Immediate**
- Direct
- Indirect
- Indirect plus Displacement
- Indirect Indexed
- Indirect Indexed plus Displacement

Immediate

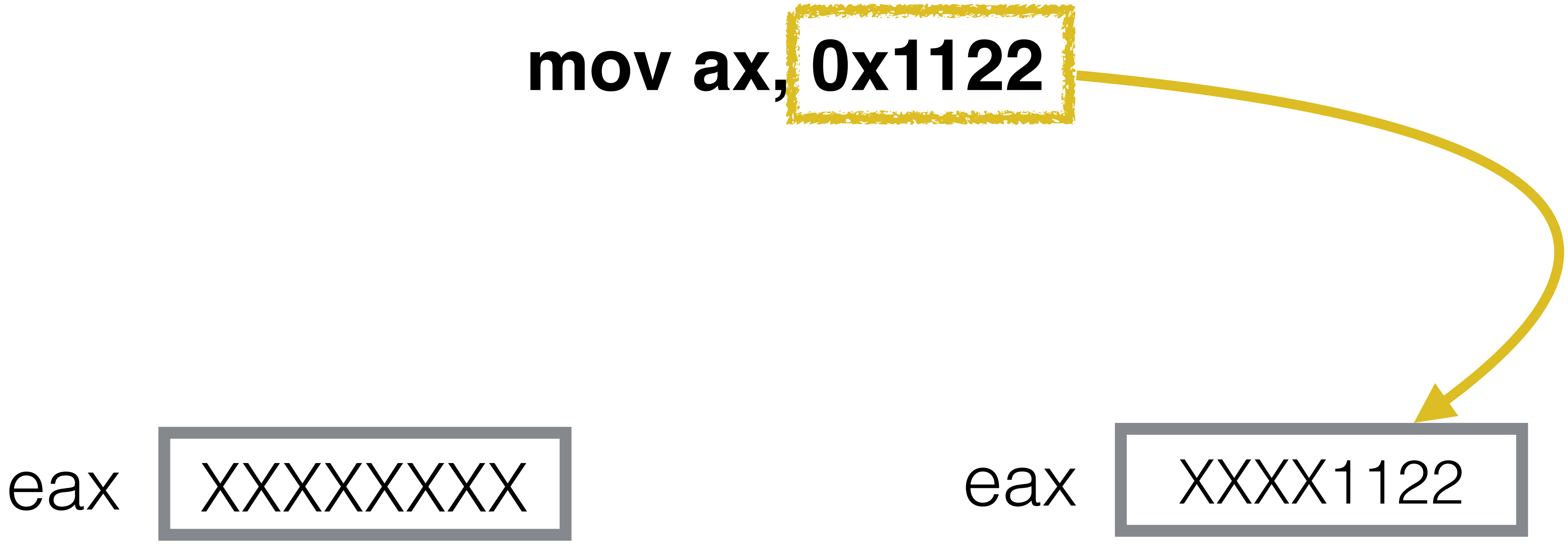
mov ax, 0x1122



eax XXXXXXXX

Before…

Immediate



mov ax, 0x1122

eax XXXXXXXX

eax XXXX1122

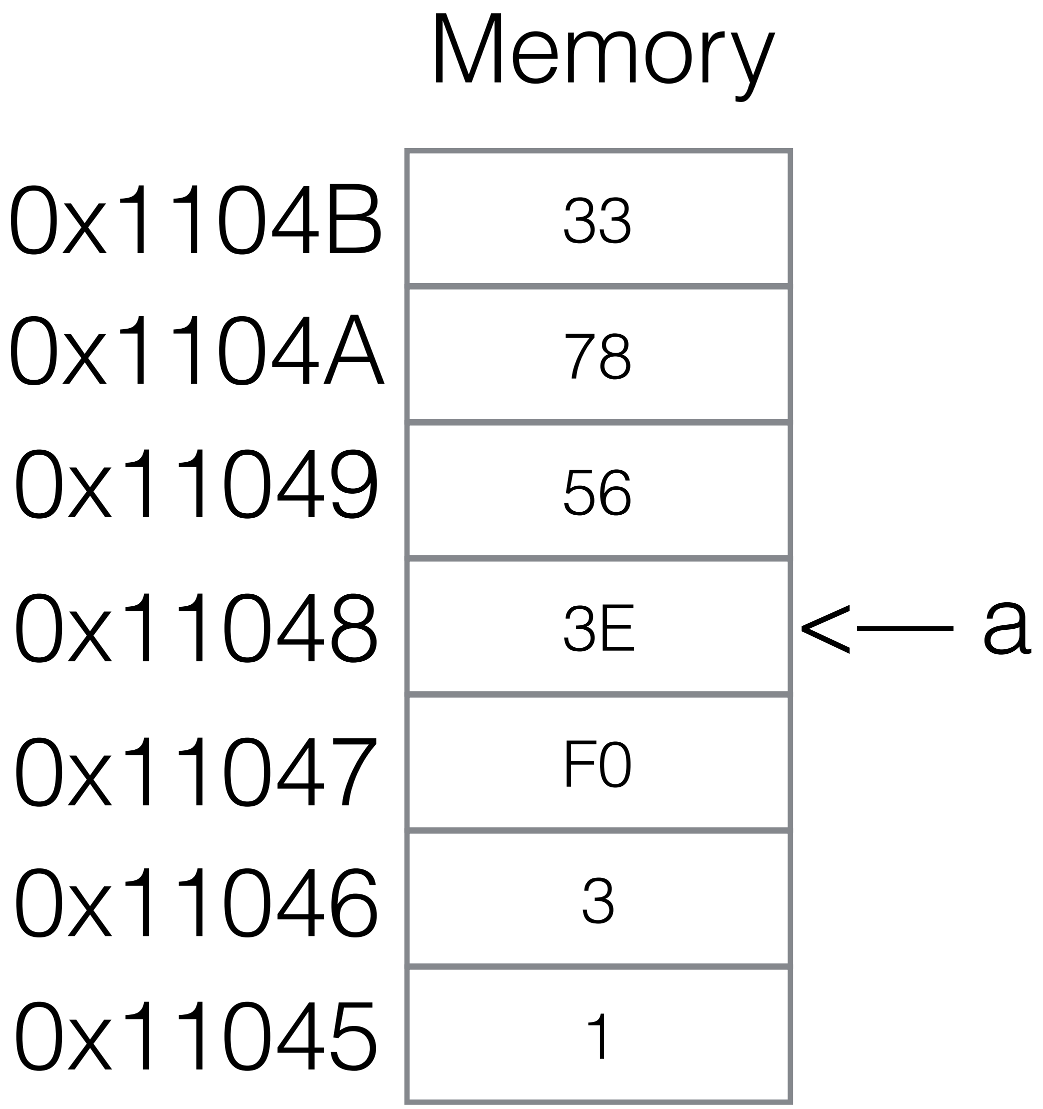
Before…

After…

- **Immediate**
- **Direct**
- Indirect
- Indirect plus Displacement
- Indirect Indexed
- Indirect Indexed plus Displacement

Direct

```
mov eax, dword[a]
```



Memory

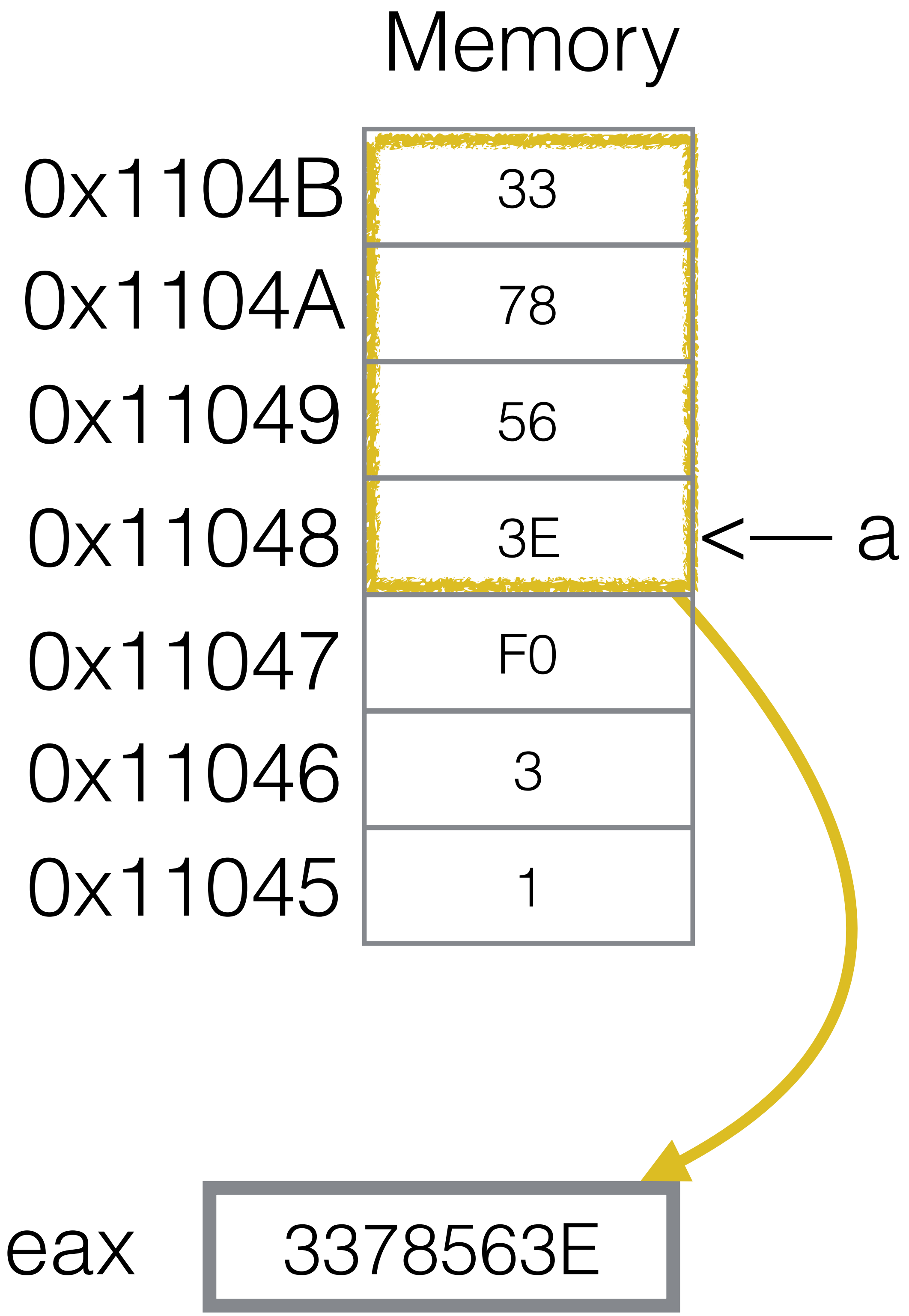
0x1104B	33	
0x1104A	78	
0x11049	56	
0x11048	3E	<— a
0x11047	F0	
0x11046	3	
0x11045	1	

eax 00000000

Before…

Direct

mov eax, dword[a]



Memory

0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	1

eax 00000000

Before…

eax 3378563E

After…

- **Immediate**
- **Direct**
- **Indirect**
- Indirect plus Displacement
- Indirect Indexed
- Indirect Indexed plus Displacement

Indirect

```
mov ebx, a
mov eax, dword[ebx]
```

Memory

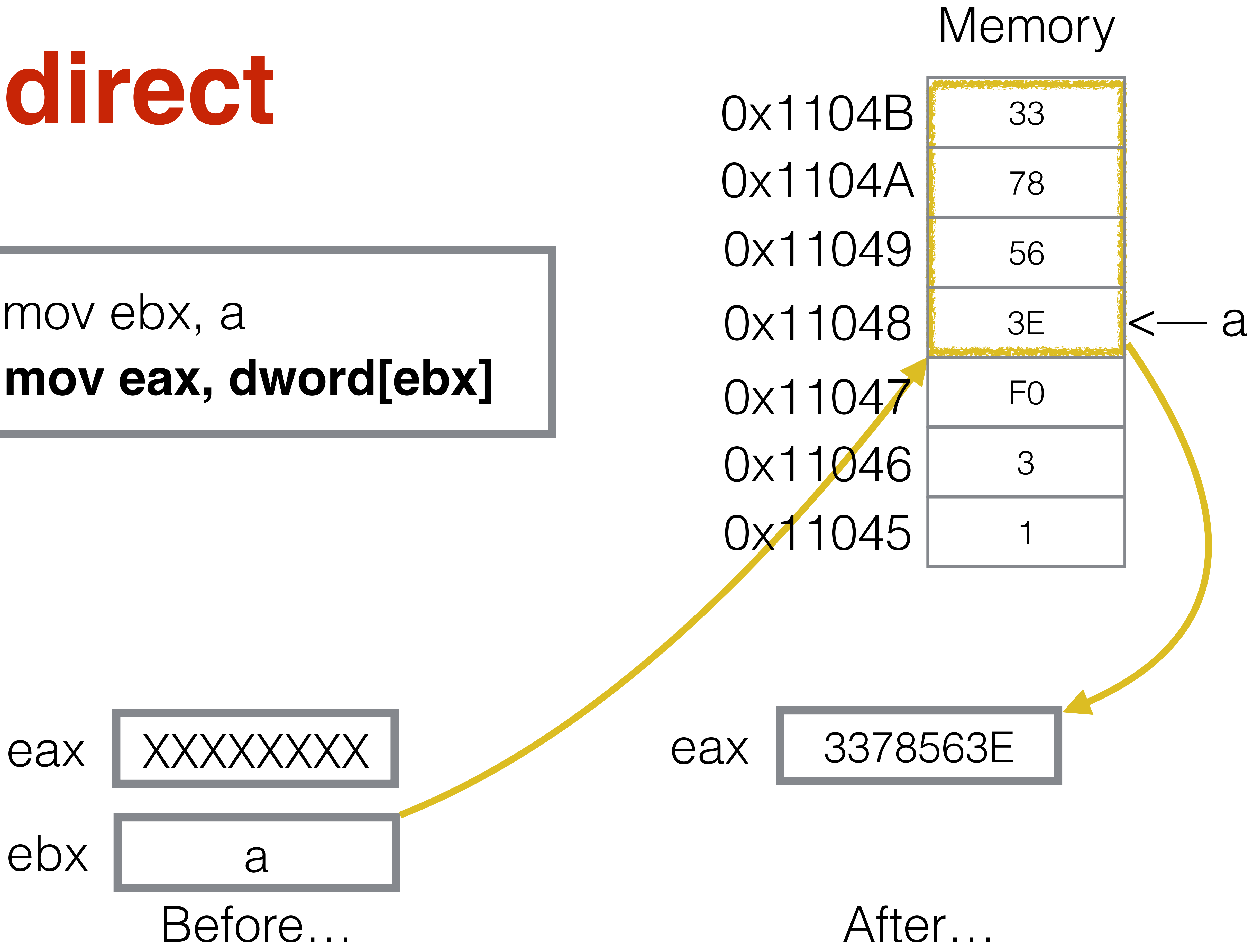
0x1104B	33	
0x1104A	78	
0x11049	56	
0x11048	3E	<— a
0x11047	F0	
0x11046	3	
0x11045	1	

eax	XXXXXXXX
ebx	a

Before…

Indirect

```
mov ebx, a
mov eax, dword[ebx]
```



Memory

0x1104B	33	
0x1104A	78	
0x11049	56	
0x11048	3E	<— a
0x11047	F0	
0x11046	3	
0x11045	1	

eax	XXXXXXXX
ebx	a

Before…

eax	3378563E

After…

- Immediate
- Direct
- Indirect
- **Indirect plus Displacement**
- Indirect Indexed
- Indirect Indexed plus Displacement

Indirect plus Dispt.

```
mov ebx, a
mov eax, dword[ebx+3]
```

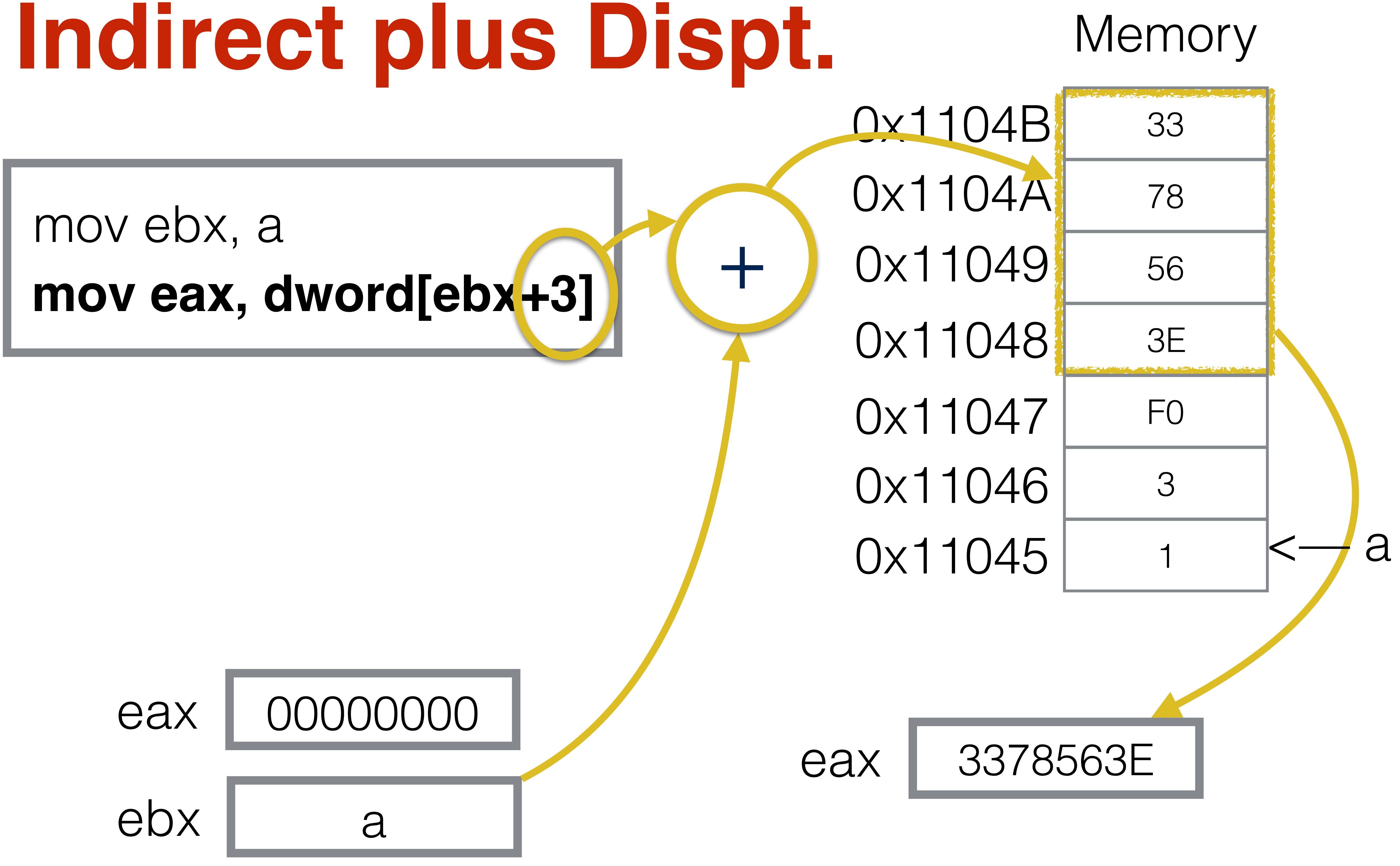
Memory

0x1104B	33	
0x1104A	78	
0x11049	56	
0x11048	3E	
0x11047	F0	
0x11046	3	
0x11045	1	<— a

eax	00000000
ebx	a

Before…

Indirect plus Dispt.



```
mov ebx, a
mov eax, dword[ebx+3]
```

Memory

0x1104B	33	
0x1104A	78	
0x11049	56	
0x11048	3E	
0x11047	F0	
0x11046	3	
0x11045	1	<— a

eax	00000000
ebx	a

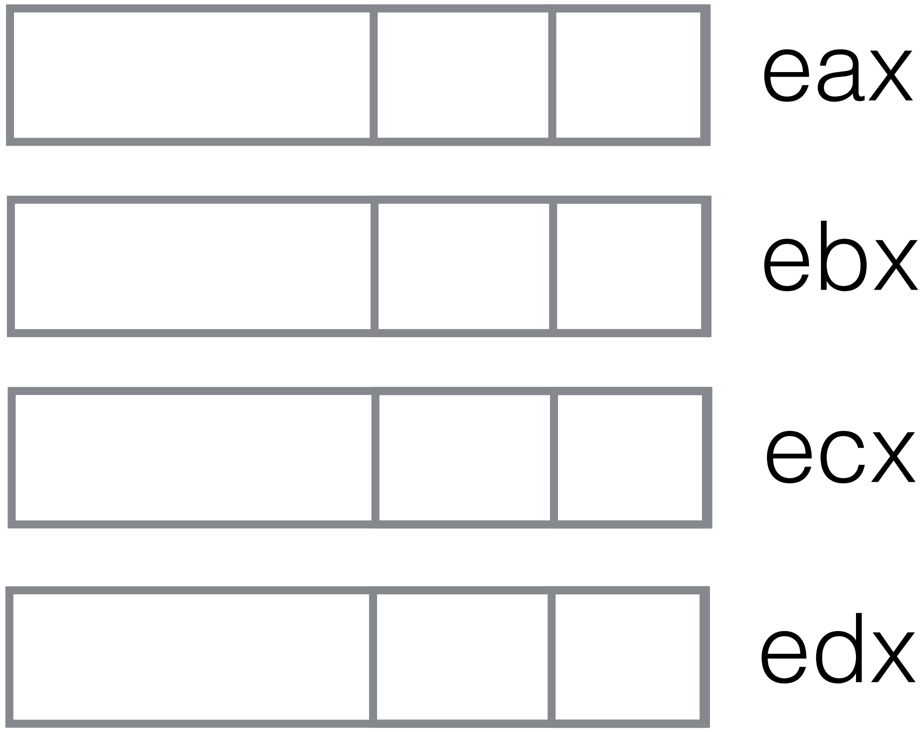
Before…

eax	3378563E

After…

- Immediate
- Direct
- Indirect
- Indirect plus Displacement
- **Indirect Indexed**
- Indirect Indexed plus Displacement

2 New Registers!



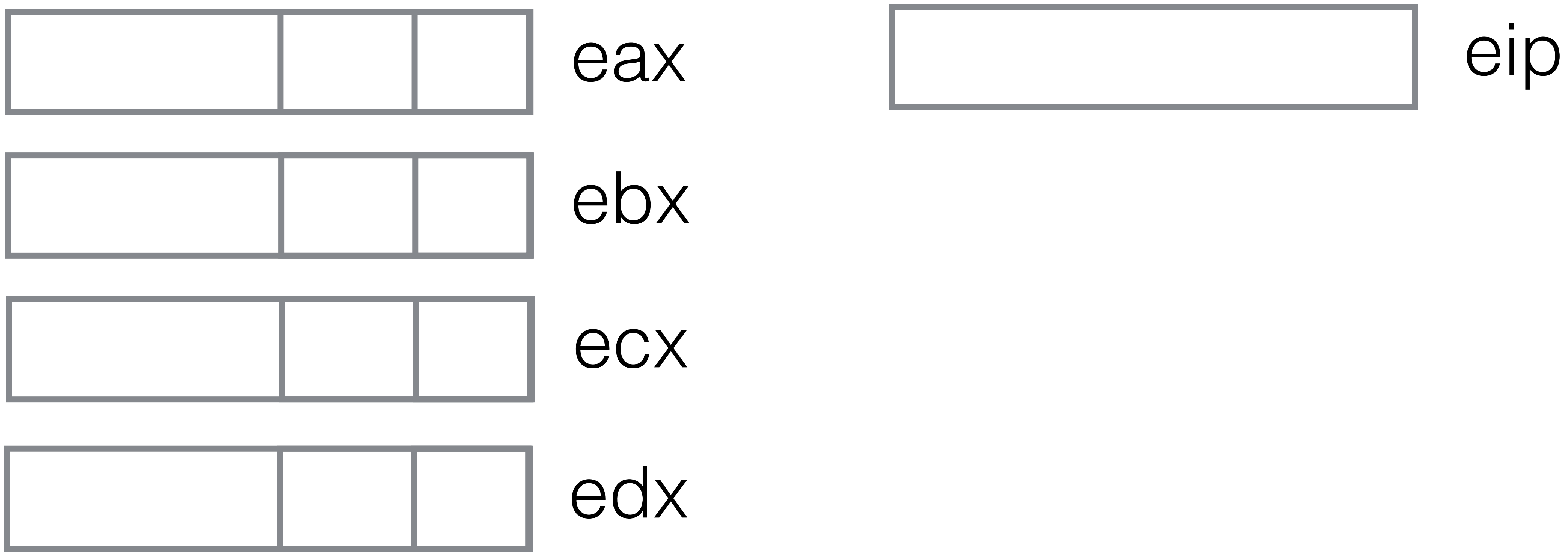
eax

ebx

ecx

edx

2 New Registers!



eax

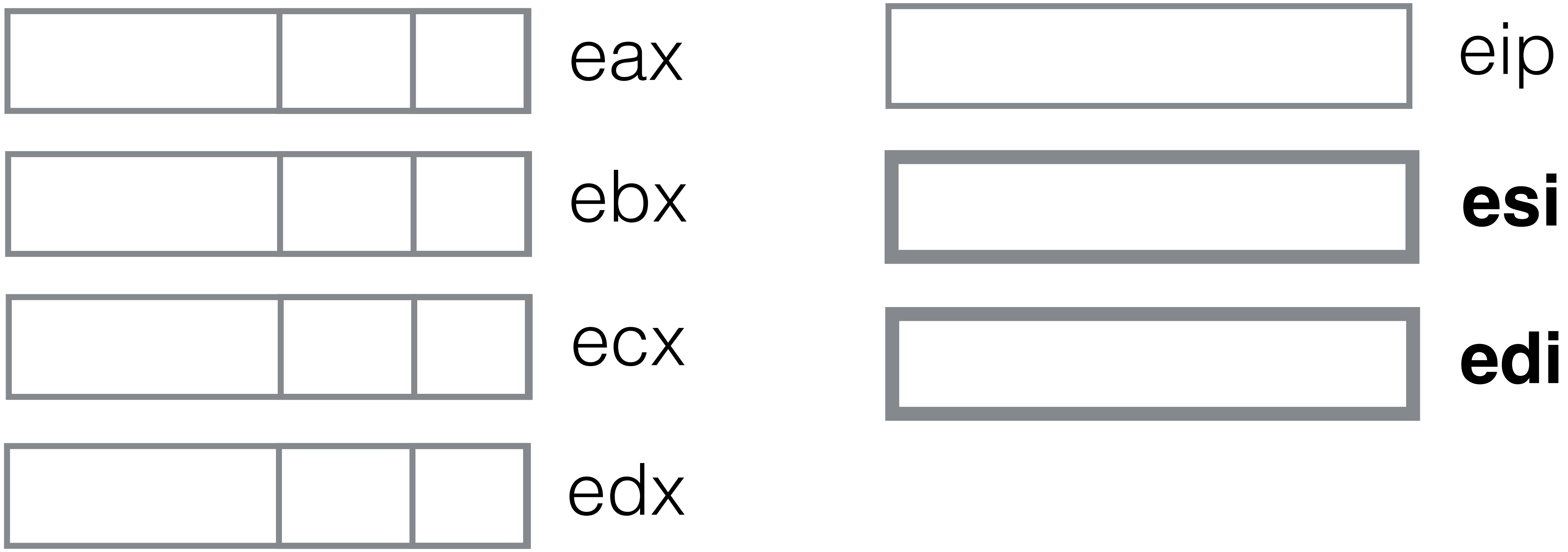
ebx

ecx

edx

eip

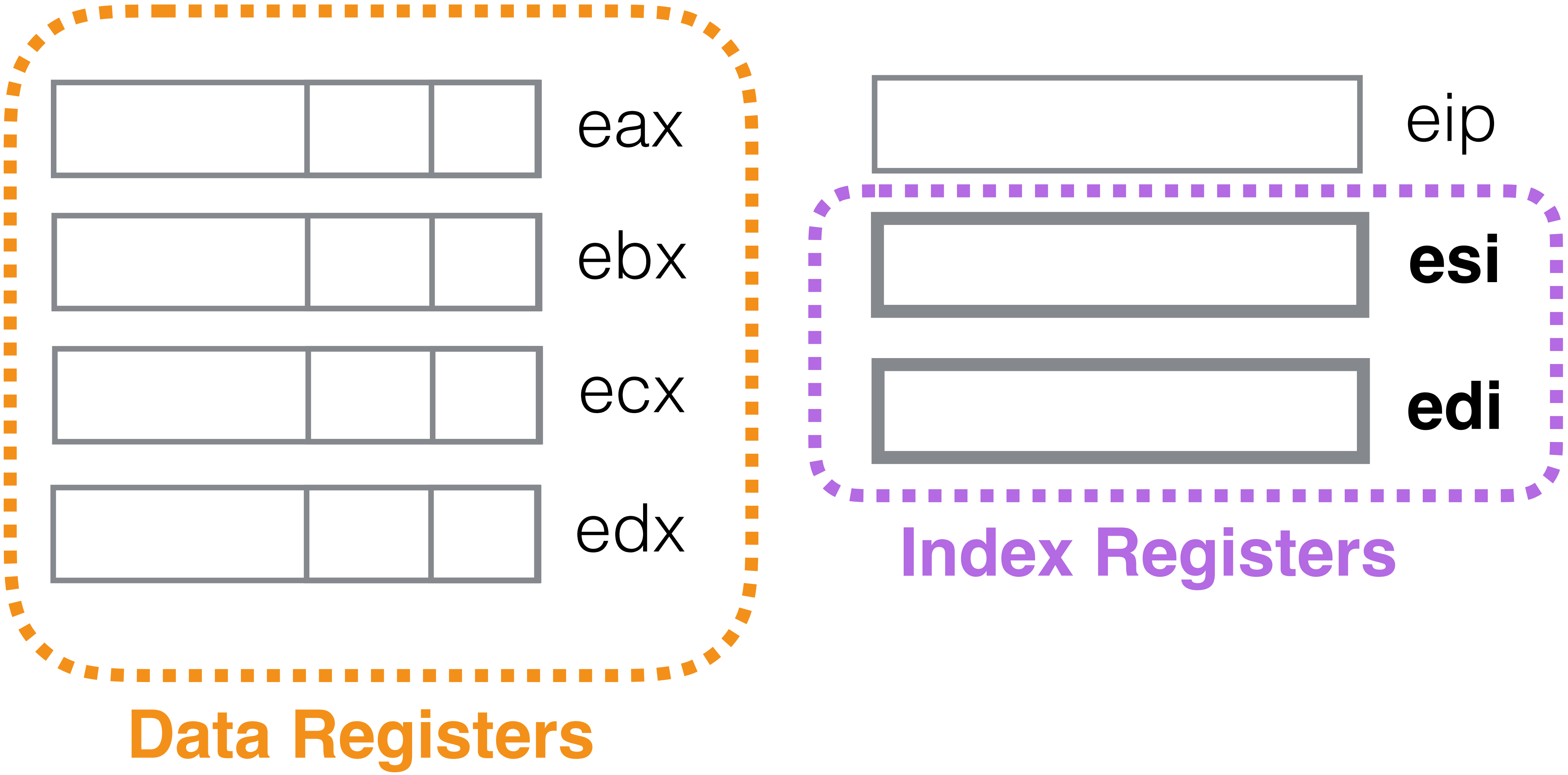
2 New Registers!



"i" in esi, edi for **index**

"s" for **source**, "d" for **destination**

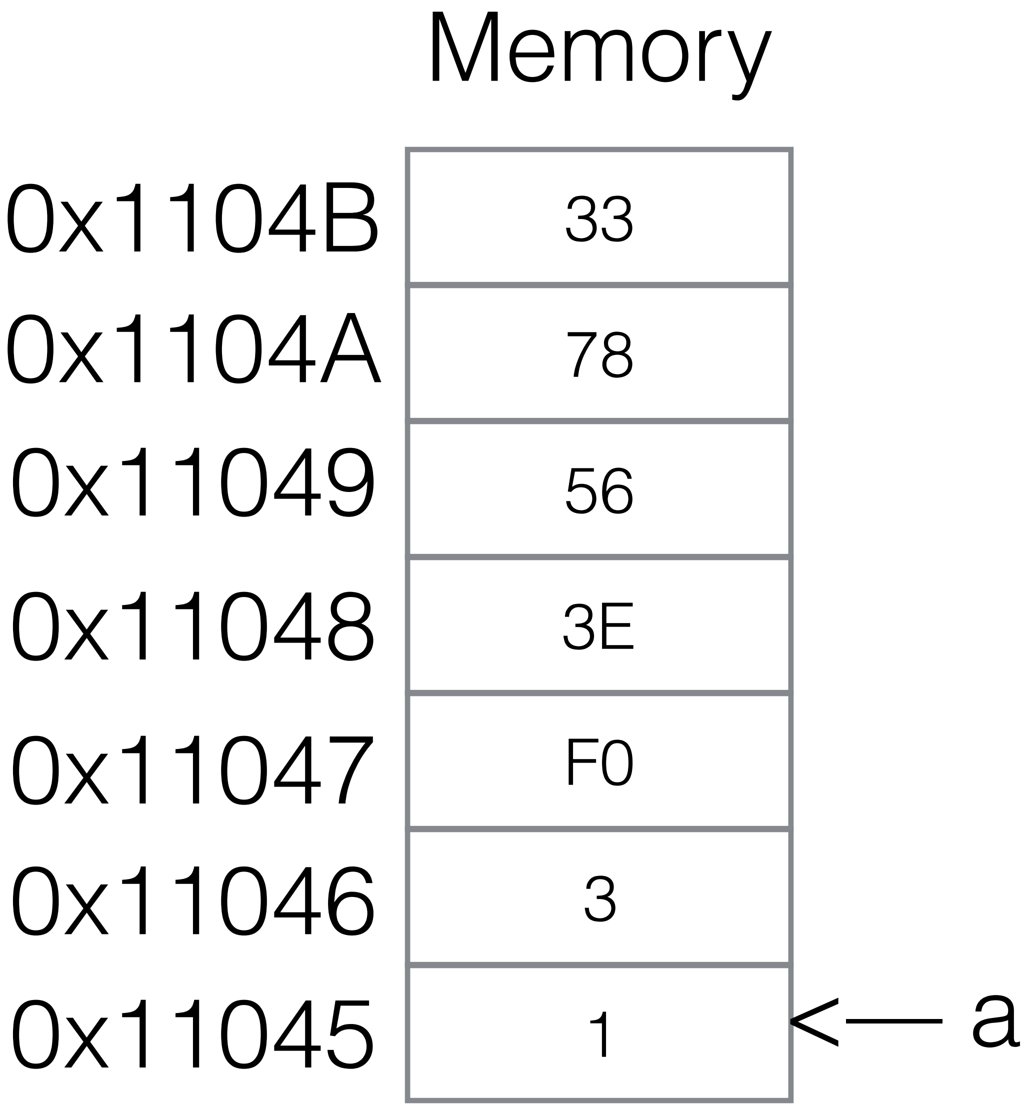
2 New Registers!



- Immediate
- Direct
- Indirect
- Indirect plus Displacement
- **Indirect Indexed**
- Indirect Indexed plus Displacement

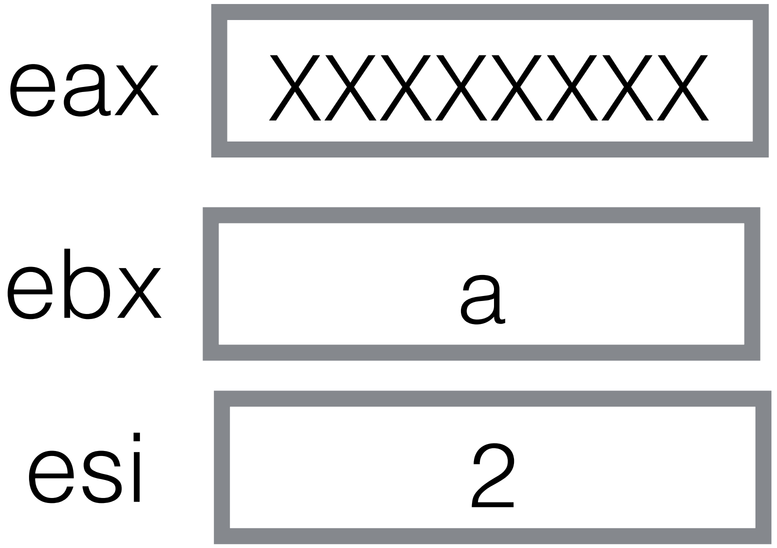
Indirect Indexed

```
mov ebx, a
mov esi, 2
mov ax, word[ebx+esi]
```



Memory

0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	1

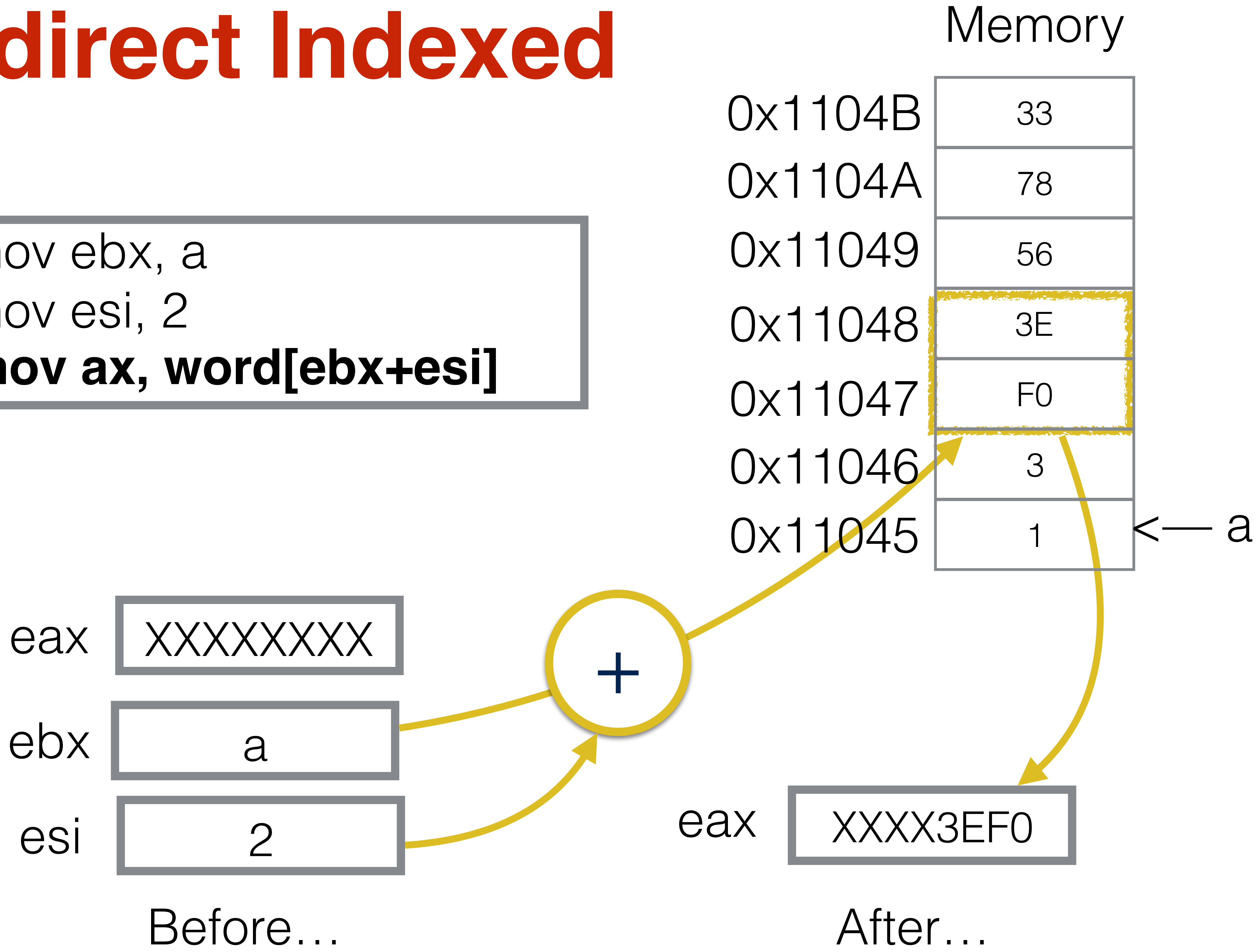


eax	XXXXXXXX
ebx	a
esi	2

Before…

Indirect Indexed

```
mov ebx, a
mov esi, 2
mov ax, word[ebx+esi]
```



Memory

0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	1

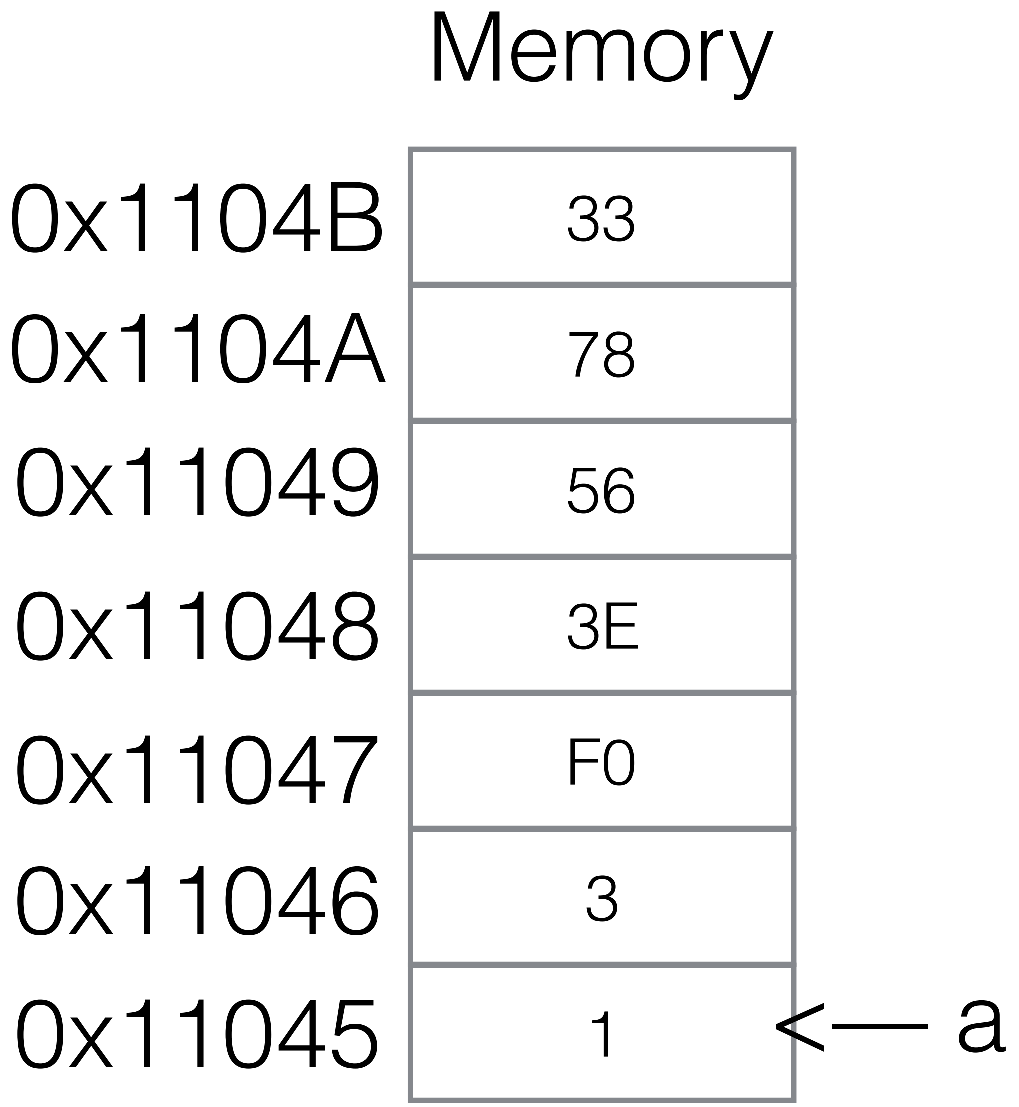
Before…

After…

- Immediate
- Direct
- Indirect
- Indirect plus Displacement
- Indirect Indexed
- **Indirect Indexed plus Displacement**

Indirect Indexed plus Displacement

```
mov ebx, a
mov esi, 2
mov ax, word[ebx+esi+1]
```



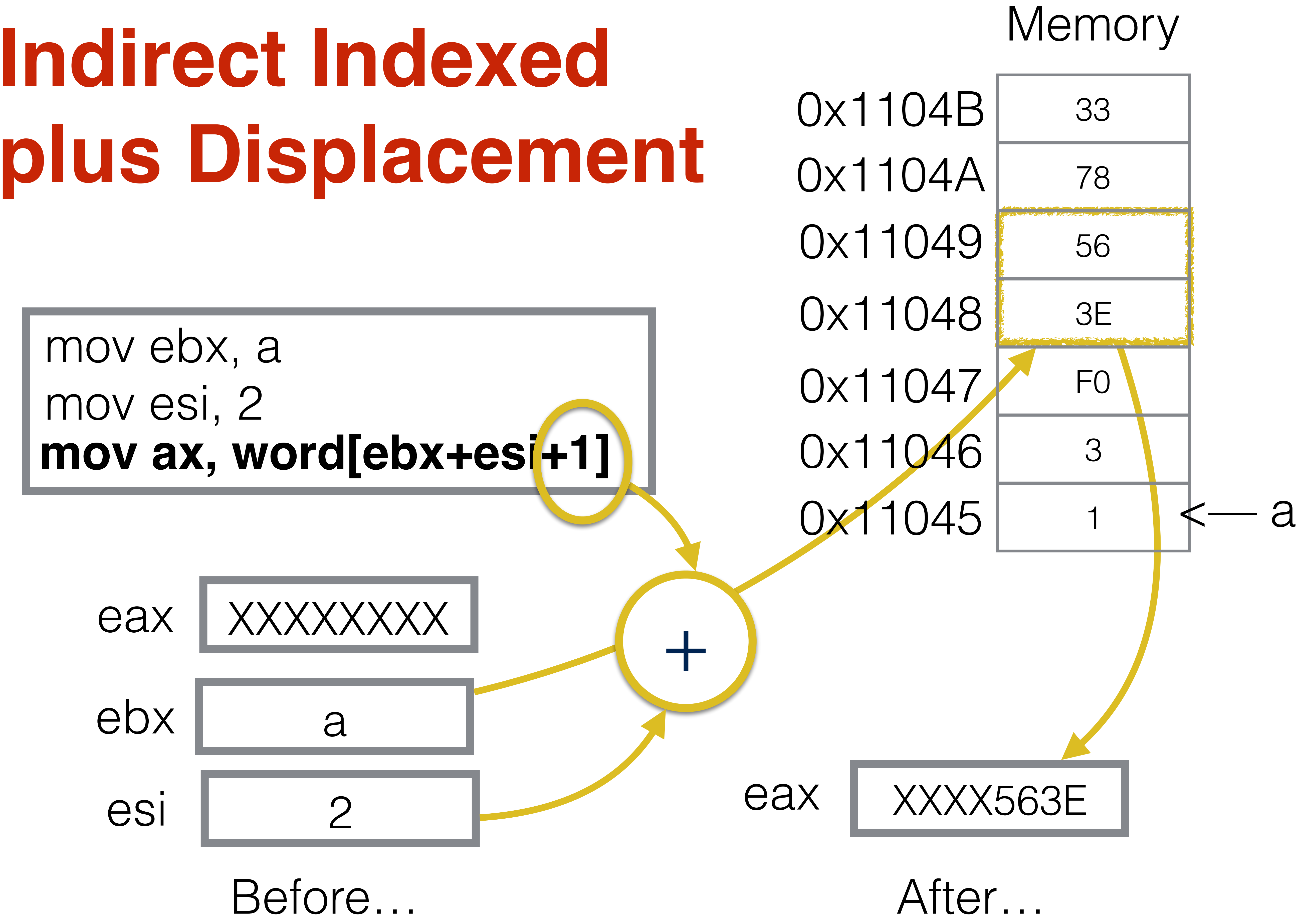
Memory

0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	1 <— a

eax	XXXXXXXX
ebx	a
esi	2

Before…

Indirect Indexed plus Displacement



```
mov ebx, a
mov esi, 2
mov ax, word[ebx+esi+1]
```

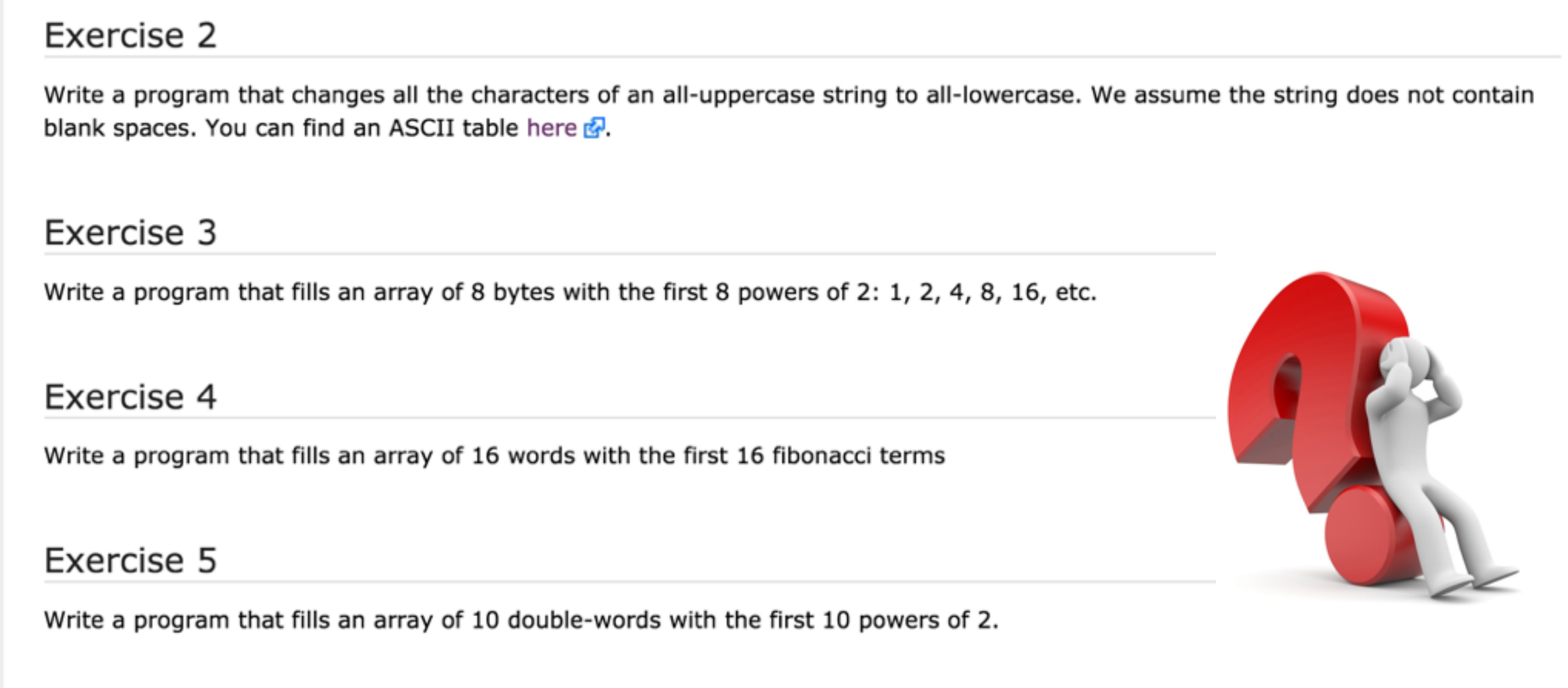
Memory

0x1104B	33
0x1104A	78
0x11049	56
0x11048	3E
0x11047	F0
0x11046	3
0x11045	1

Before…	
eax	XXXXXXXX
ebx	a
esi	2

After…	
eax	XXXX563E

```
;;;  --------------------------------------------------------------
;;;  Identify possible errors in the instructions below, and
;;;  indicate the addressing mode for each one.
;;;  --------------------------------------------------------------

                section .data
a               db      3
b               db      0x12345678
c               dw      0
x               dd      30
array           dd      1,2,3,4,5,6,7,8,9,10

                section .text
                global  _start

_start:         mov     eax, a
                mov     eax, dword[a] ; is it an error?
                mov     ebx, array
                mov     eax, dword[ebx]
                mov     esi, 0
                mov     dword[ebx+esi], 0
                mov     dword[ebx+esi+4], eax
                mov     edi, b
                mov     byte[edi], 'Z'
                add     al, 'z'-'Z'
                mov     ecx, 10
for:            inc     ecx
                loop    for

;;;  exit()

                mov     eax,1
                mov     ebx,0
                int     0x80    ; final system call
```



Exercise 2

Write a program that changes all the characters of an all-uppercase string to all-lowercase. We assume the string does not contain blank spaces. You can find an ASCII table here.

Exercise 3

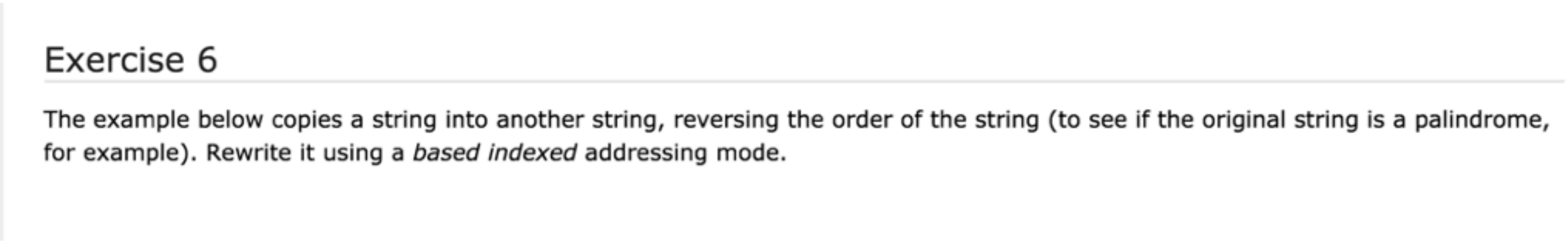
Write a program that fills an array of 8 bytes with the first 8 powers of 2: 1, 2, 4, 8, 16, etc.

Exercise 4

Write a program that fills an array of 16 words with the first 16 fibonacci terms

Exercise 5

Write a program that fills an array of 10 double-words with the first 10 powers of 2.



Exercise 6

The example below copies a string into another string, reversing the order of the string (to see if the original string is a palindrome, for example). Rewrite it using a *based indexed* addressing mode.

```
msg1    db      "A man, a plan, a canal, Panama"
msg2    db      "                              "
MSGLEN  equ     $-msg2

        mov     esi, msg1
        mov     edi, msg2+MSGLEN-1
        mov     ecx, MSGLEN

for     mov     al, byte[esi]
        mov     byte[edi], al
        inc     esi
        dec     edi
        loop    for
```