Smith College

Computer Science

CSC231—Assembly

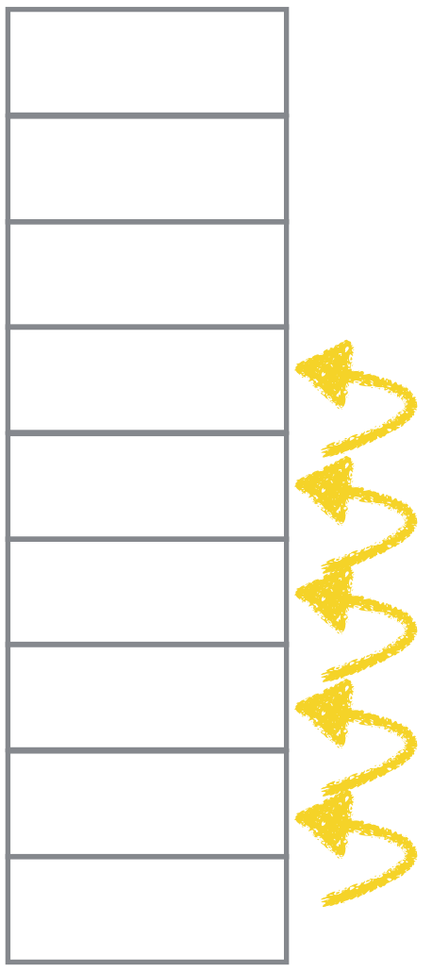
Week #9 — Fall 2017

Dominique Thiébaut
dthiebaut@smith.edu

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 sequence 1,2, 3, 4, … 10 into an array of 10 ints using a loop.

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.
How long would this take on a 1GHz processor?

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
```

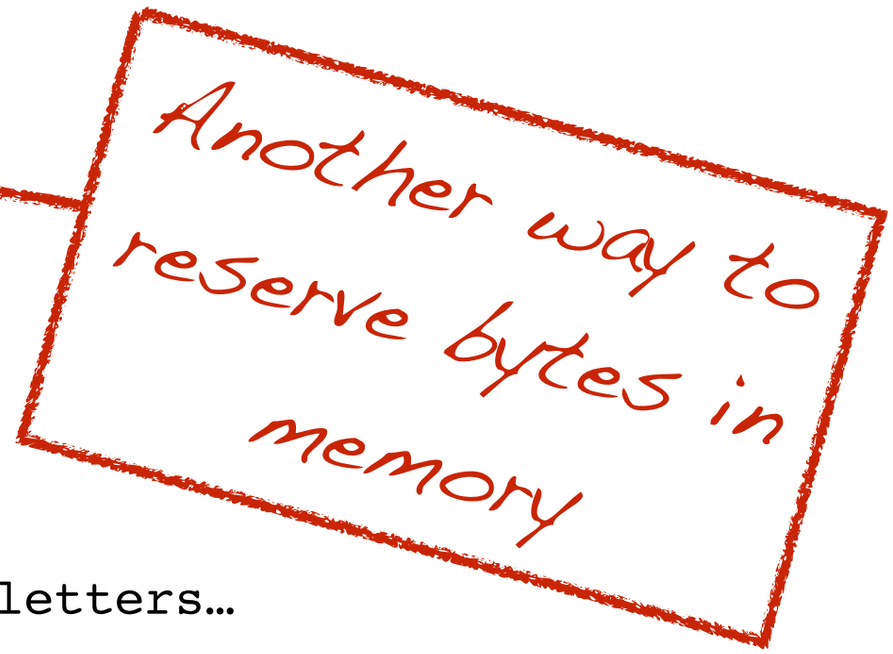
1,000,000 DNA Bases: How fast?

```
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

1,000,000 DNA Bases: How fast?

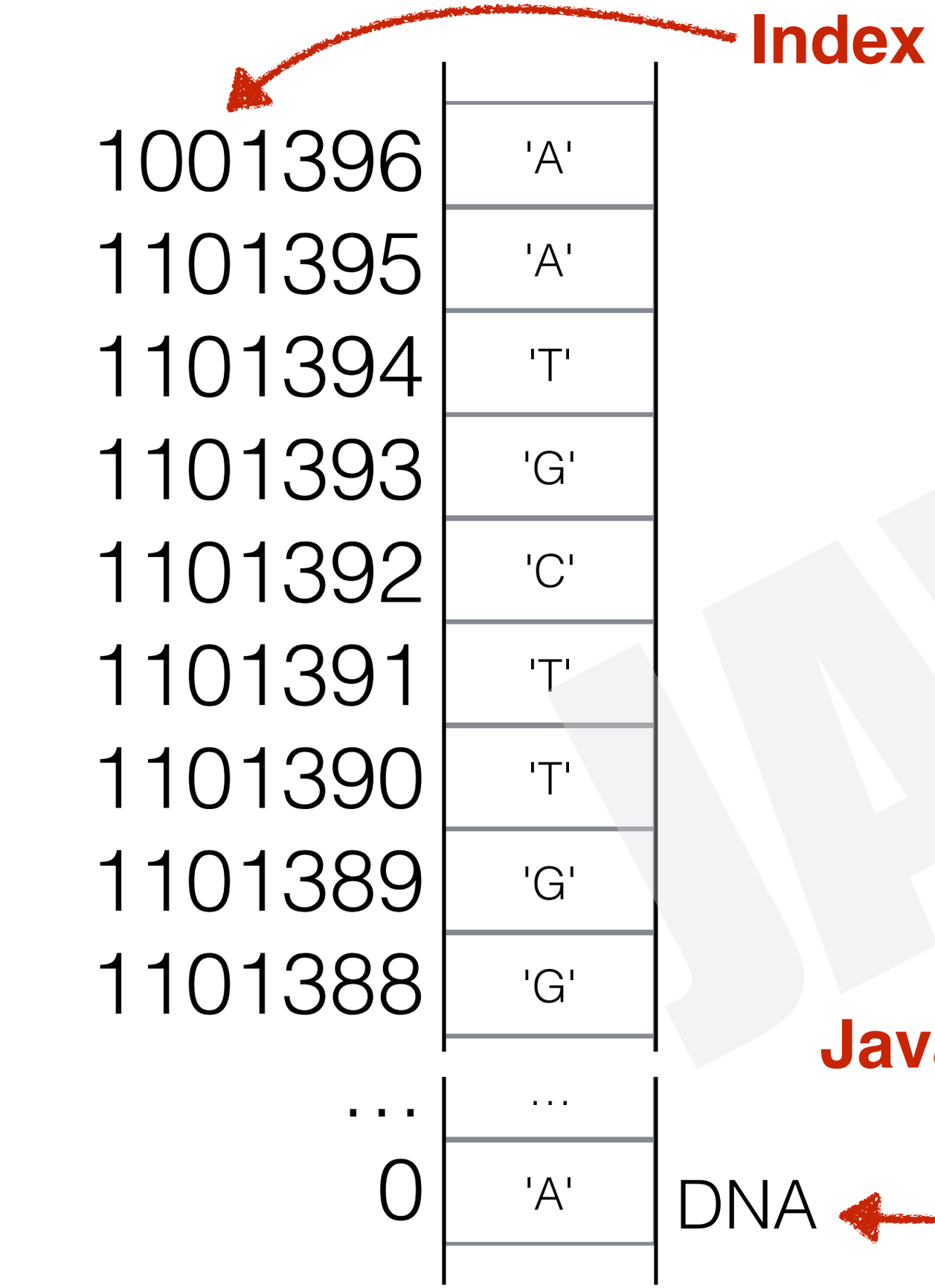
```
N       equ     1000000

        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
```

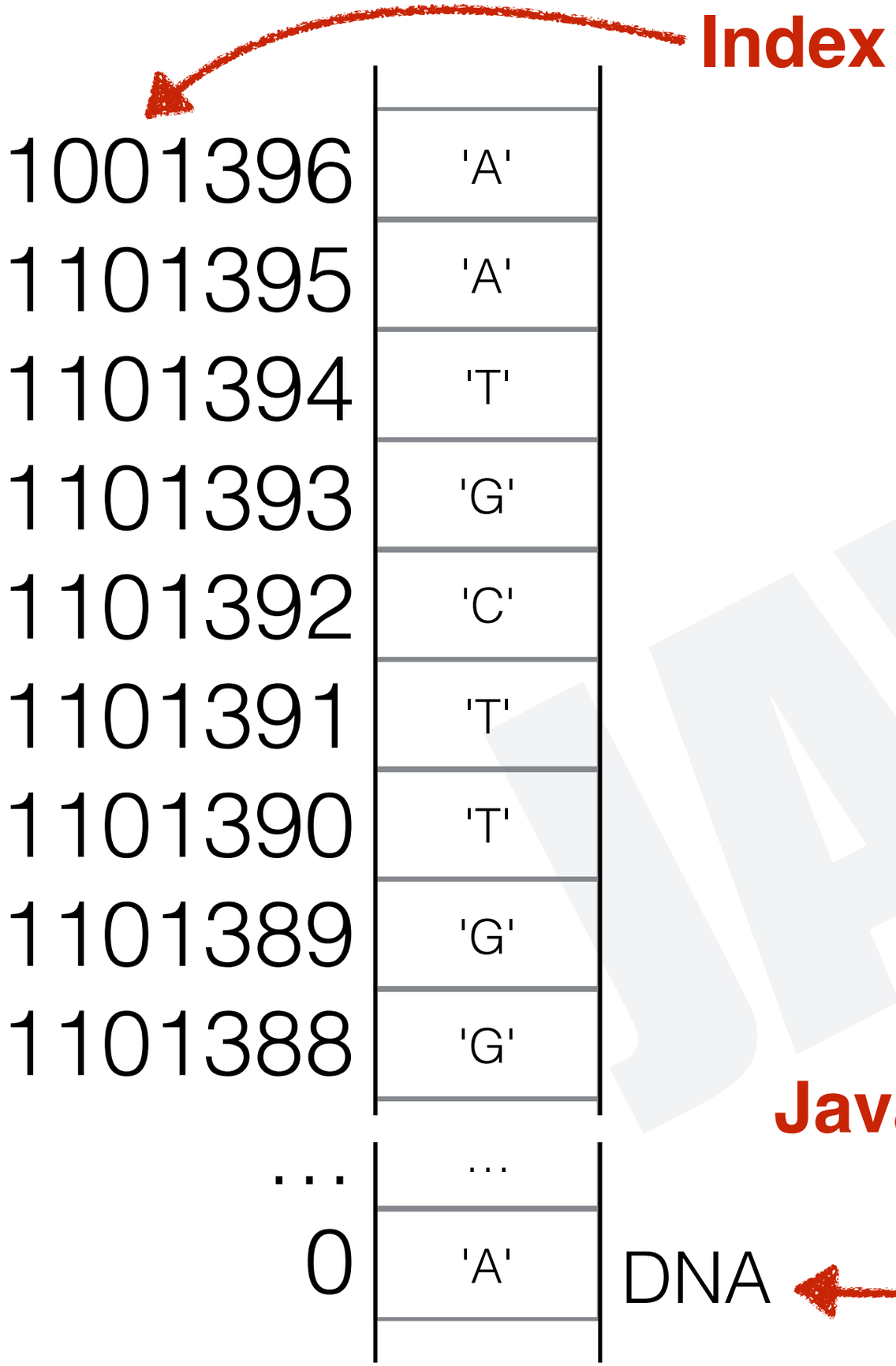


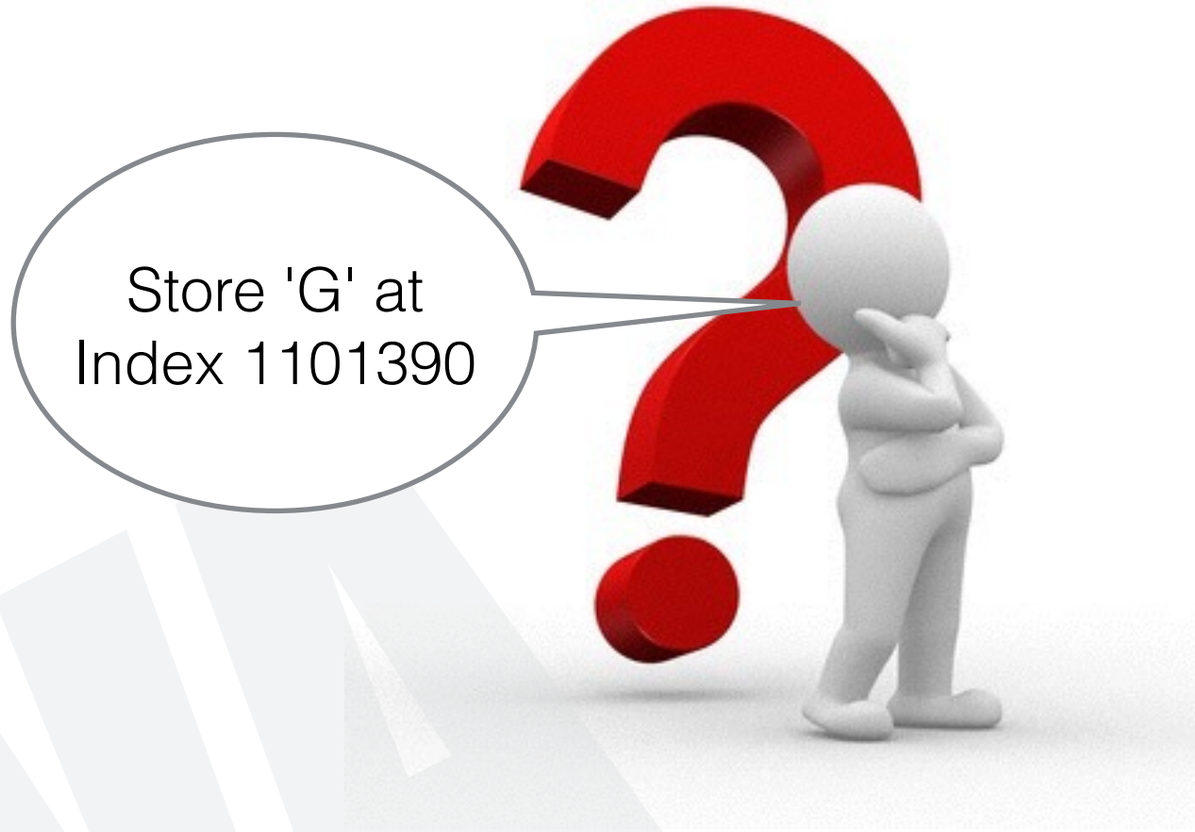
We stopped here last time…

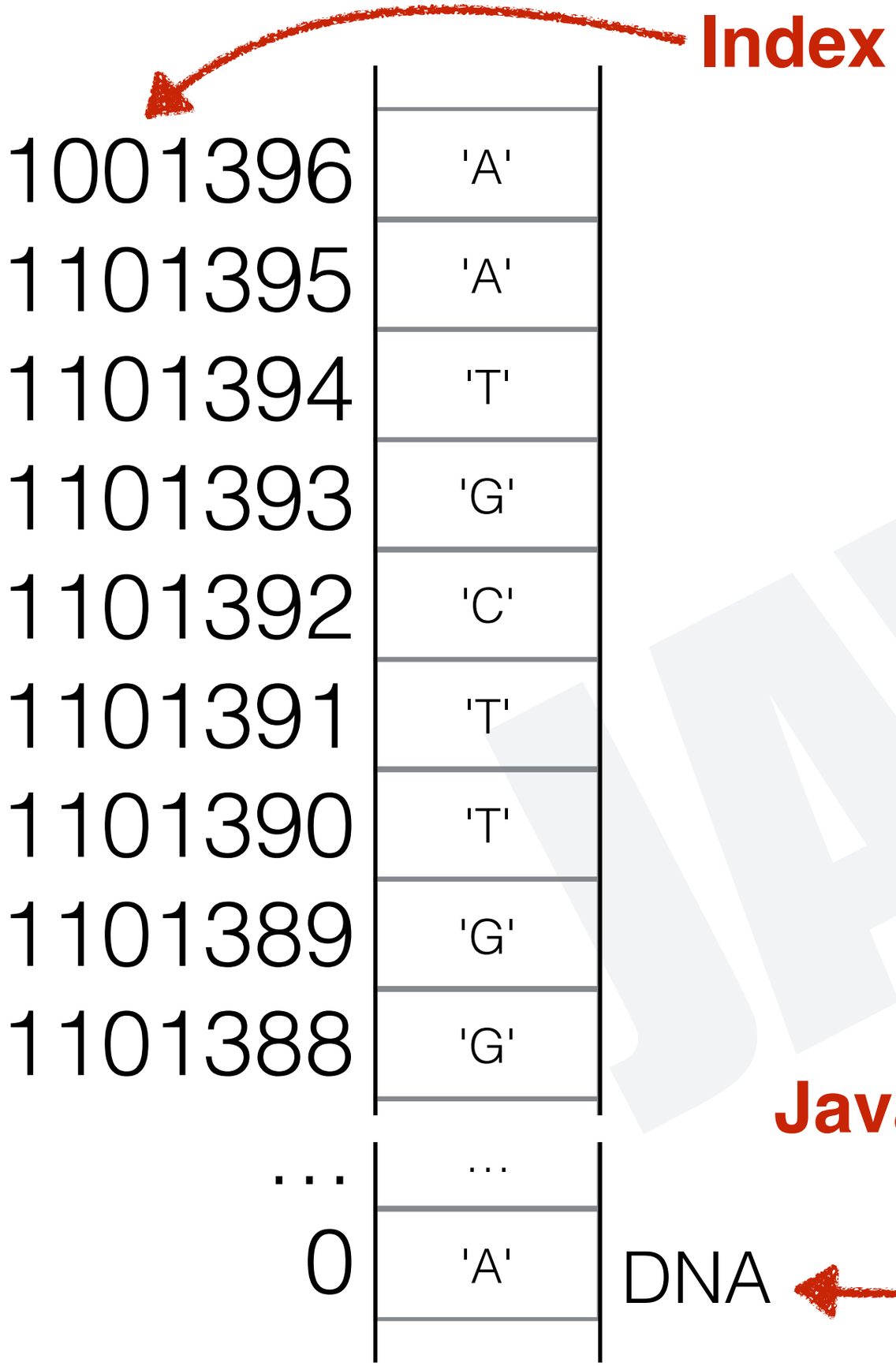
Friday Lab: Video Introduction

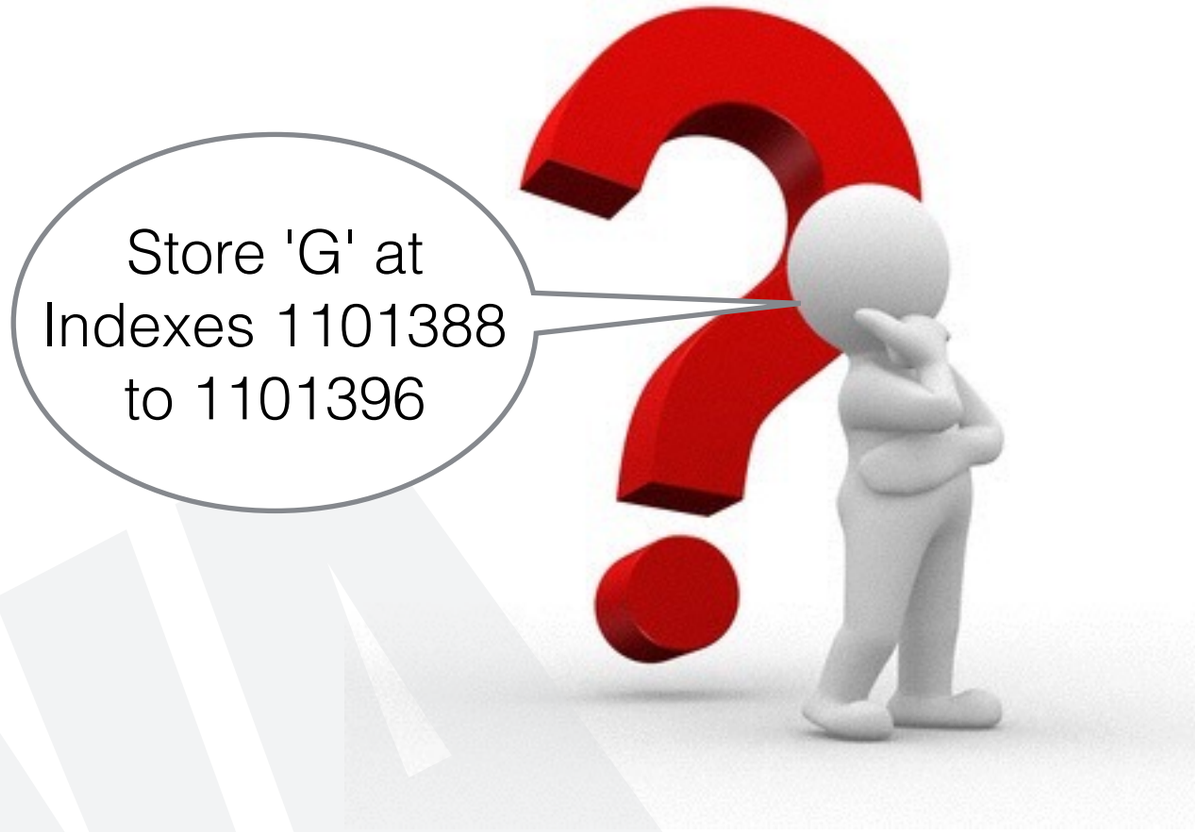
Understanding Indirect Addressing


Index
1001396
'A'
1101395
'A'
1101394
'T'
1101393
'G'
1101392
'C'
1101391
'T'
1101390
'T'
1101389
'G'
1101388
'G'
...
...
0
'A'
DNA
Java Array

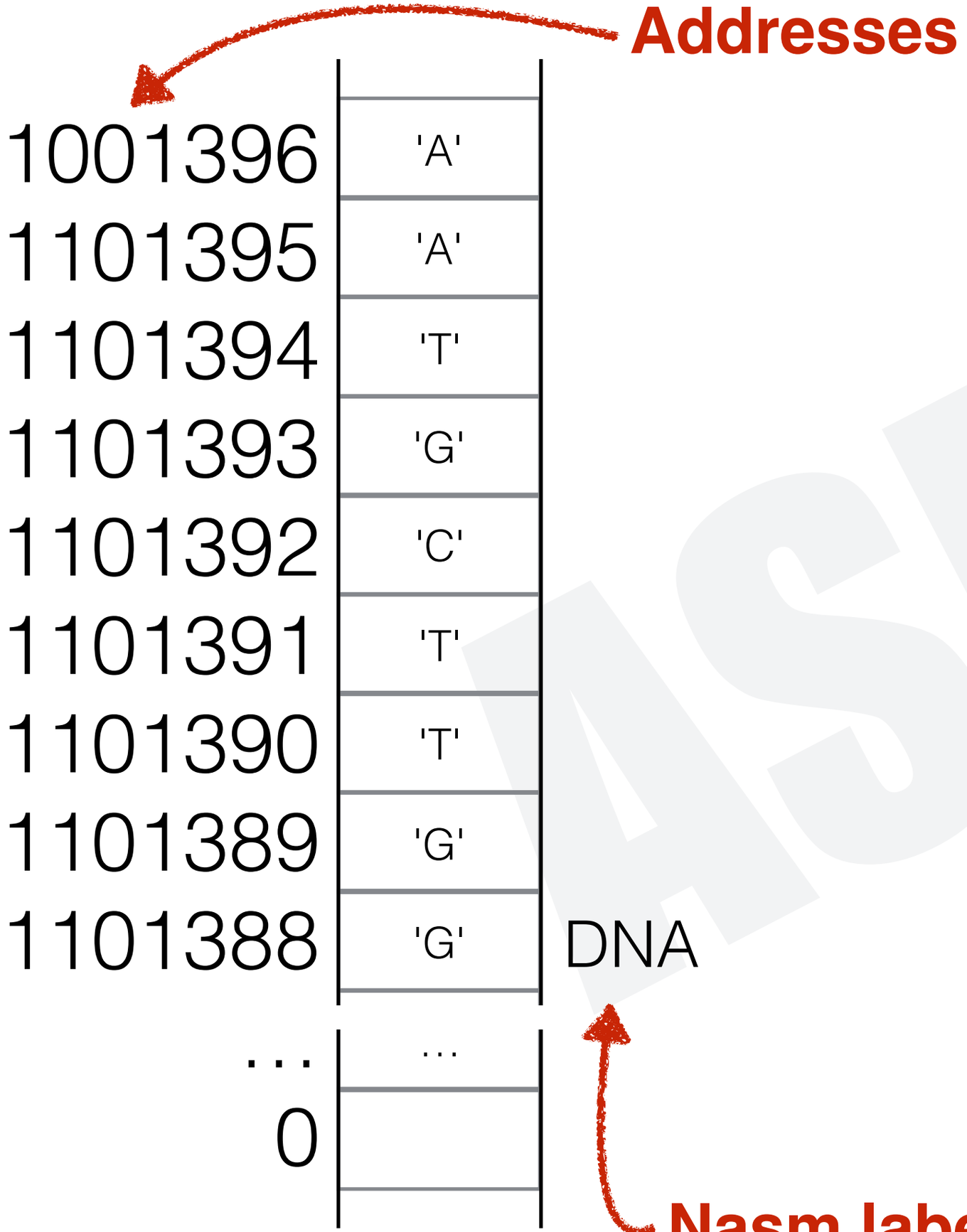

Index
1001396
'A'
1101395
'A'
1101394
'T'
1101393
'G'
1101392
'C'
1101391
'T'
1101390
'T'
1101389
'G'
1101388
'G'
...
...
0
'A'
DNA
Java Array


Store 'G' at
Index 1101390


Index
1001396
'A'
1101395
'A'
1101394
'T'
1101393
'G'
1101392
'C'
1101391
'T'
1101390
'T'
1101389
'G'
1101388
'G'
...
...
0
'A'
DNA
Java Array


Store 'G' at
Indexes 1101388
to 1101396

JAVA ASM


Addresses
1001396
'A'
1101395
'A'
1101394
'T'
1101393
'G'
1101392
'C'
1101391
'T'
1101390
'T'
1101389
'G'
1101388
'G'
DNA
...
...
0
Nasm label

Addresses

1001396	'A'
1101395	'A'
1101394	'T'
1101393	'G'
1101392	'C'
1101391	'T'
1101390	'T'
1101389	'G'
1101388	'G'
...	...
0	

DNA

Nasm label

Addresses

1001396	'A'
1101395	'A'
1101394	'T'
1101393	'G'
1101392	'C'
1101391	'T'
1101390	'T'
1101389	'G'
1101388	'G'
...	...
0	

DNA

Nasm label

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

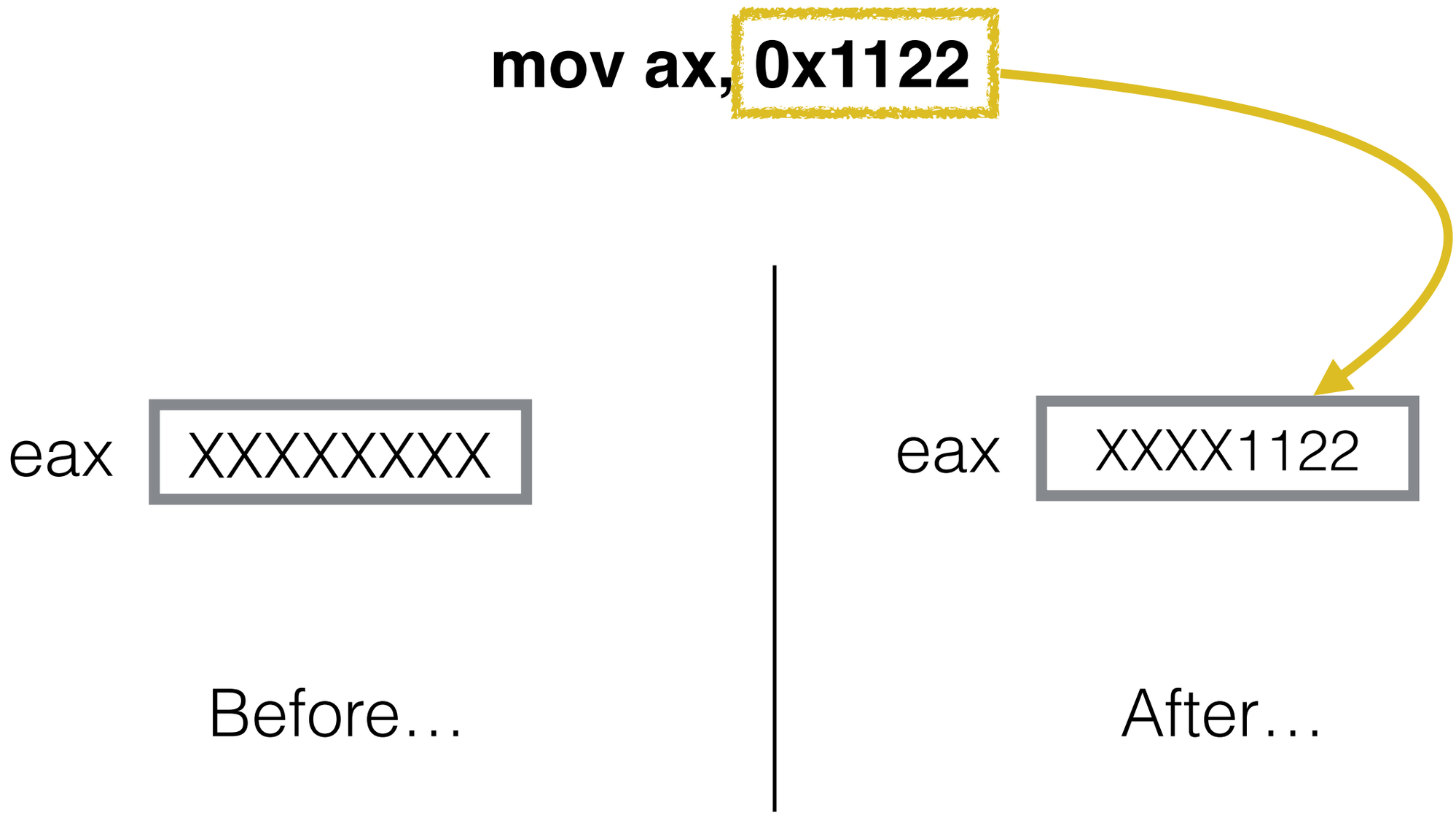
Immediate

mov ax, 0x1122

eax | XXXXXXXX

Before…

Immediate



thiebaut — ssh — 102×25

Python × ssh +

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 03 00 01 00 00 00  80 80 04 08 34 00 00 00  |............4...|
00000020  e8 00 00 00 00 00 00 00  34 00 20 00 02 00 28 00  |........4. ...(.|
00000030  06 00 03 00 01 00 00 00  00 00 00 00 00 80 04 08  |................|
00000040  00 80 04 08 a2 00 00 00  a2 00 00 00 05 00 00 00  |................|
00000050  00 10 00 00 01 00 00 00  a4 00 00 00 a4 90 04 08  |................|
00000060  a4 90 04 08 1c 00 00 00  1c 00 00 00 06 00 00 00  |................|
00000070  00 10 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  b8 04 00 00 00 bb 01 00  00 00 b9 a4 90 04 08 ba  |................|
00000090  1c 00 00 00 cd 80 b8 01  00 00 00 bb 00 00 00 00  |................|
000000a0  cd 80 00 00 0a 68 65 6c  6c 6f 20 77 6f 72 6c 64  |.....hello world|
000000b0  21 0a 2a 2a 2a 2a 2a 2a  2a 2a 2a 2a 2a 2a 0a 0a  |!.**************|
000000c0  00 2e 73 79 6d 74 61 6
000000d0  00 2e 73 68 73 74 72 7
000000e0  00 2e 64 61 74 61 00 0
000000f0  00 00 00 00 00 00 00 0
00000100  00 00 00 00 00 00 00 0
00000110  1b 00 00 00 01 00 00 0
00000120  80 00 00 00 22 00 00 0
00000130  10 00 00 00 00 00 00 0
00000140  03 00 00 00 a4 90 04 0
00000150  00 00 00 00 00 00 00 0
00000160  11 00 00 00 03 00 00 0
00000170  c0 00 00 00 27 00 00 0
00000180  01 00 00 00 00 00 00 0
```

temp — ssh — 102×25

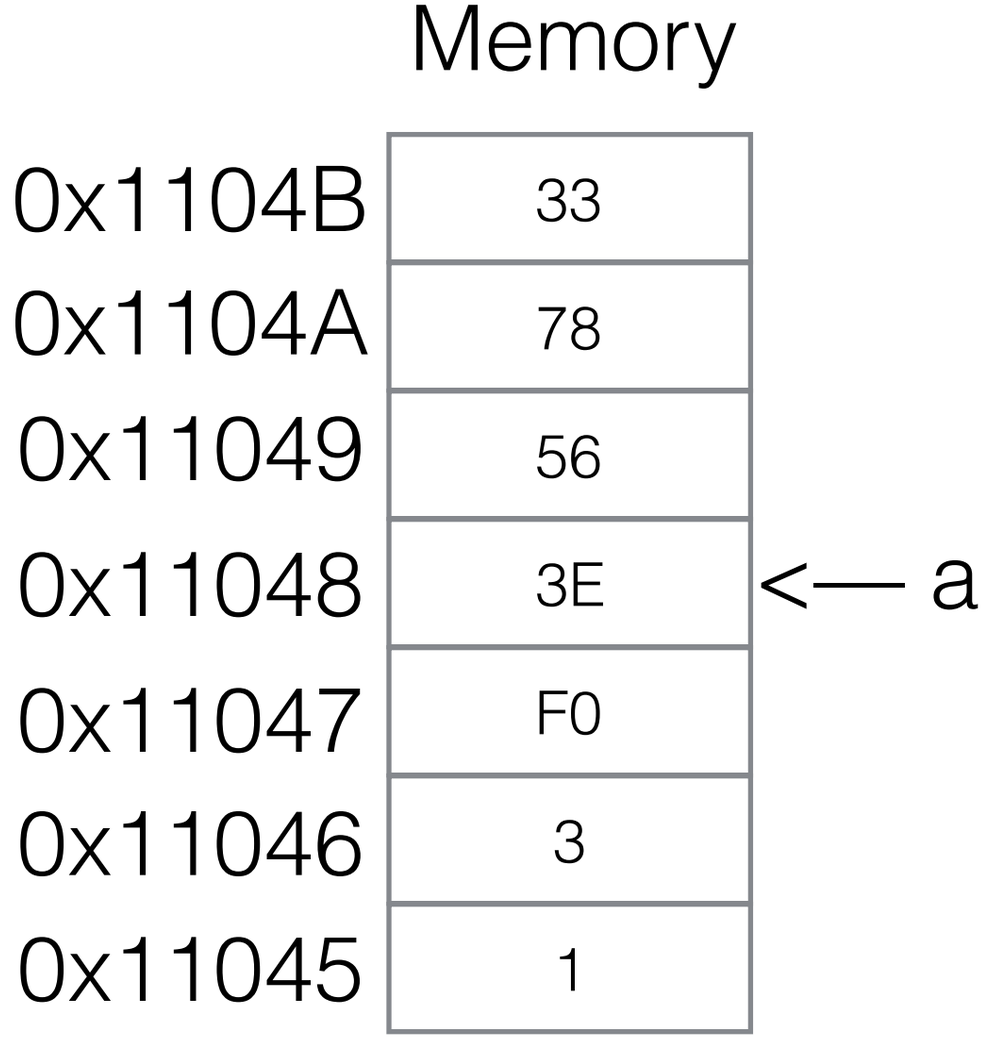
Python ssh +

```
    23
    24                          ;;;  ------------------------------------------------------
110 1b 00 00 00 01 00 00 0 ---
120 80 00 00 00 22 00 00 0      25  ;;;  code area
130 10 00 00 00 00 00 00 0      26  ;;;  ------------------------------------------------------
140 03 00 00 00 a4 90 04 0 ---
150 00 00 00 00 00 00 00 0      27
160 11 00 00 00 03 00 00 0      28                          section .text
170 c0 00 00 00 27 00 00 0      29                          global  _start
180 01 00 00 00 00 00 00 0      30
                                31                      _start:
                                32  00000000 B804000000                 mov     eax, 4          ; print
                                33  00000005 BB01000000                 mov     ebx, 1          ; to stdout
                                34  0000000A B9[00000000]               mov     ecx, message    ; string
                                35  0000000F BA1C000000                 mov     edx, msgLen     ; # of chars
                                36
                                37  00000014 CD80                       int     0x80            ; ask Linux to print
                                38
                                39
                                40
                                41                      ;;;  exit()
                                42
                                43  00000016 B801000000                 mov     eax,1
                                44  0000001B BB00000000                 mov     ebx,0
:
```

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

Direct

mov eax, dword[a]

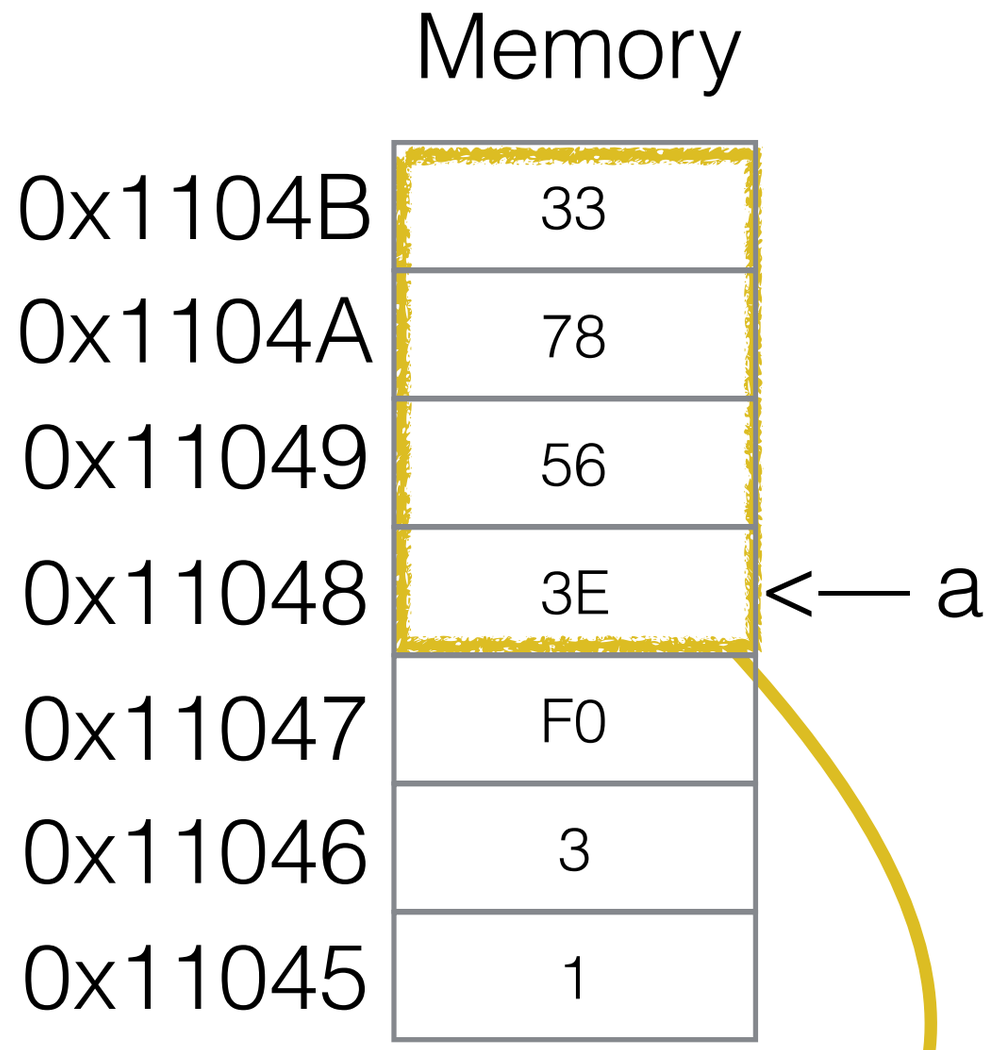


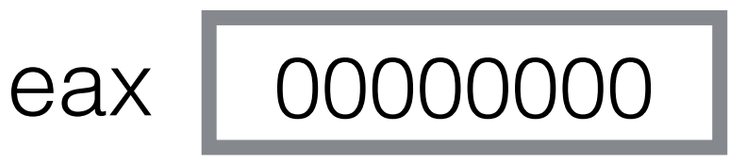
eax 00000000

Before…

Direct

mov eax, dword[a]





Before…

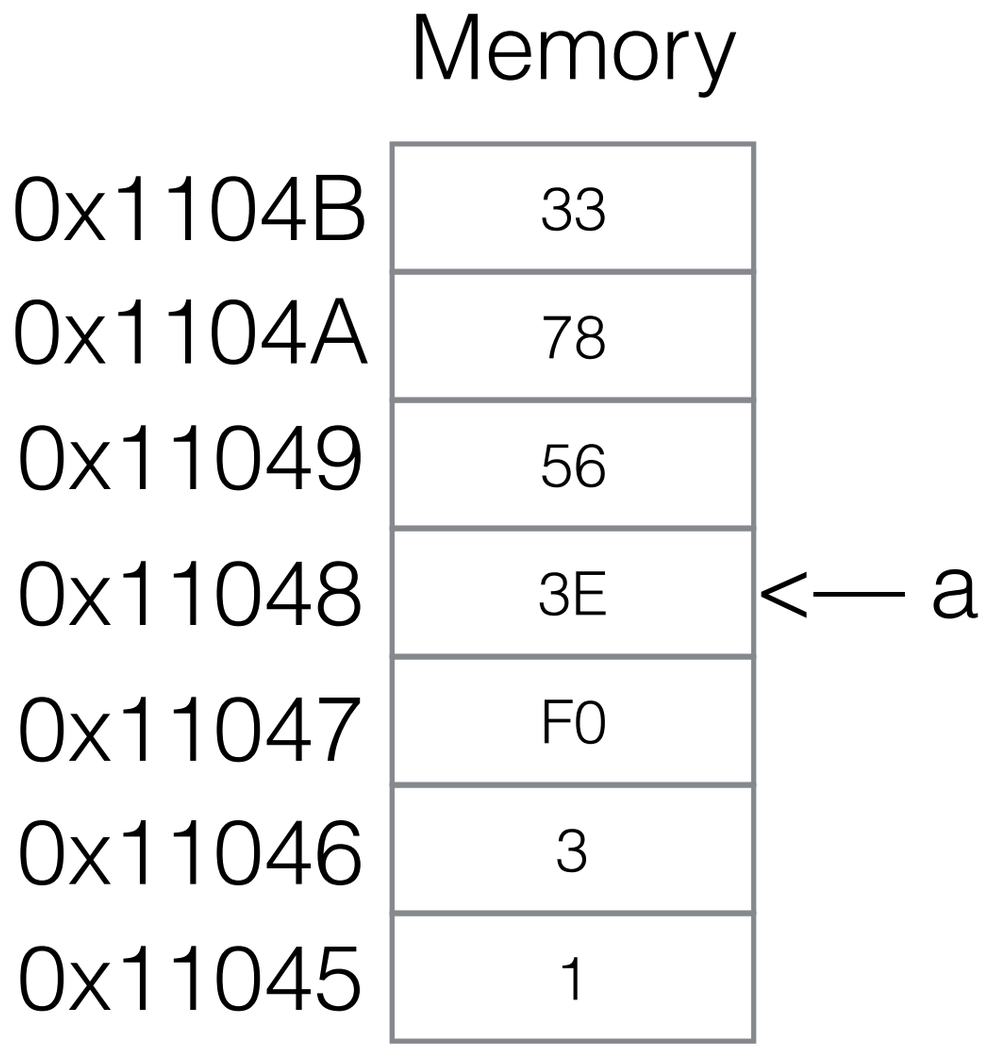
After…

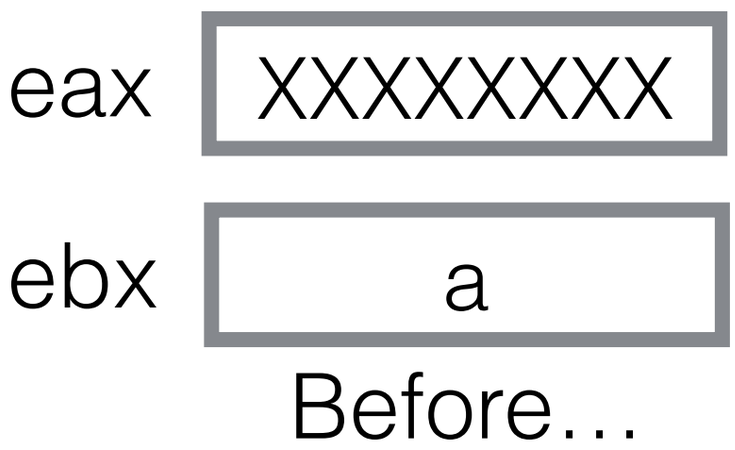
- Immediate
- Direct
- **Indirect**

- Indirect plus Displacement
- Indirect Indexed
- Indirect Indexed plus Displacement

Indirect

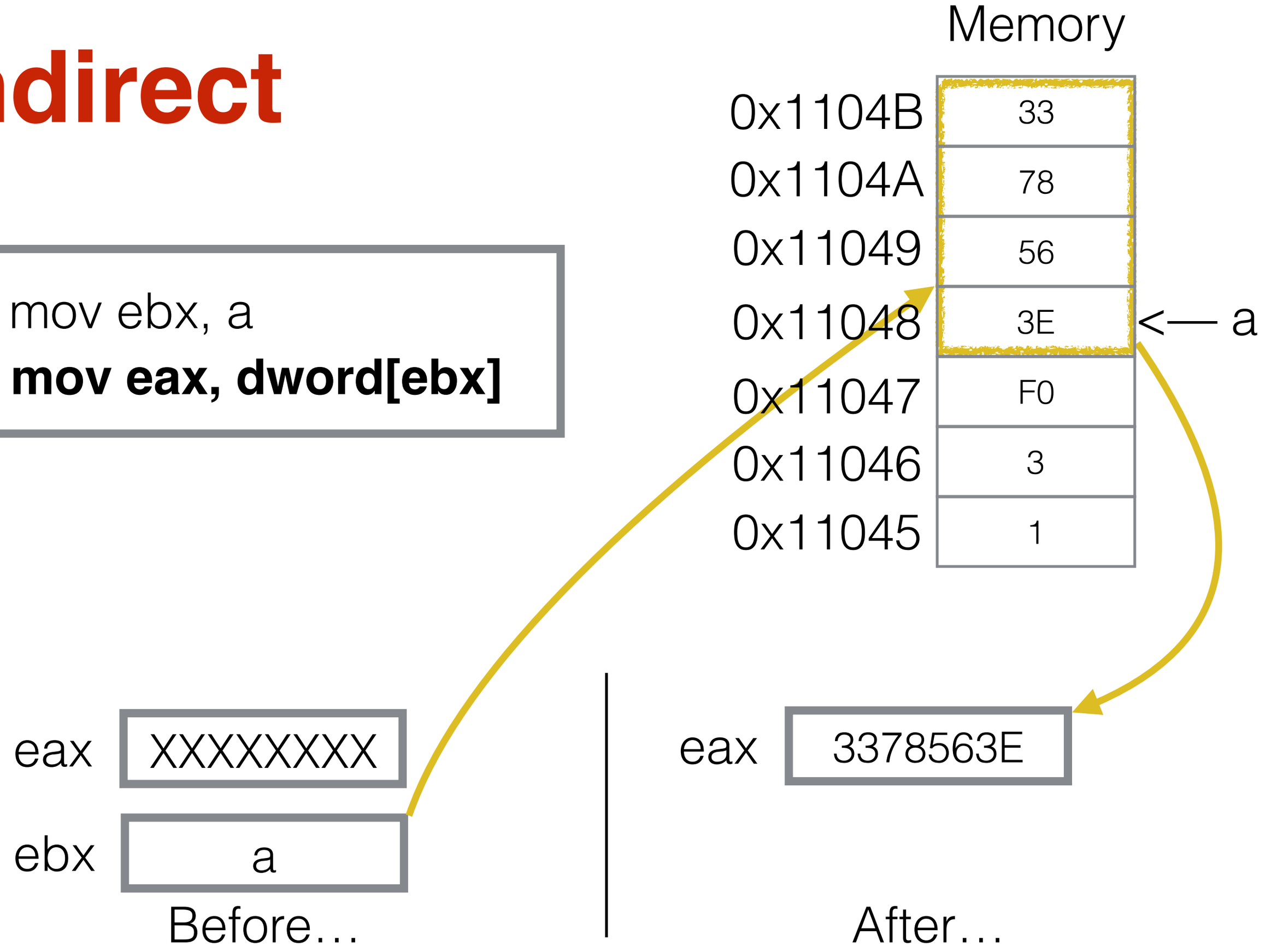
```
mov ebx, a
mov eax, dword[ebx]
```





Indirect

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



- 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

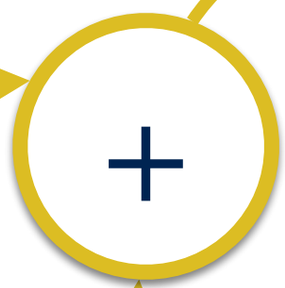
eax	00000000
ebx	a

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

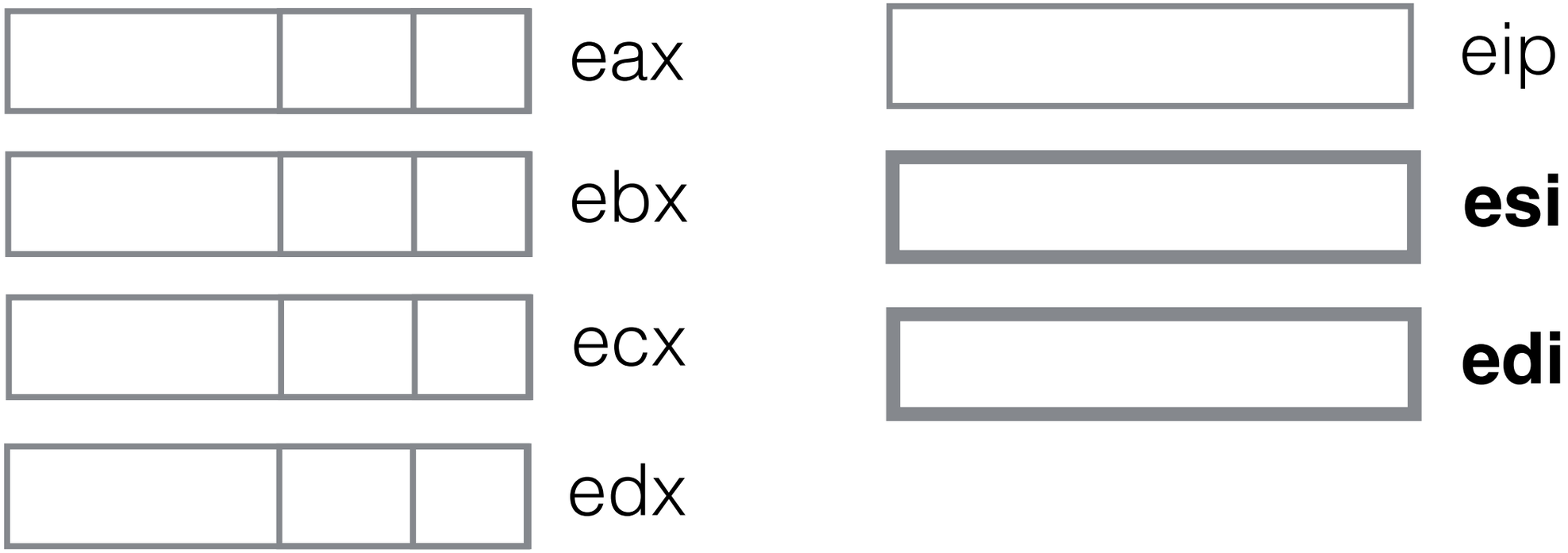
After…

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

2 New Registers!

2 New Registers!

2 New Registers!



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

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

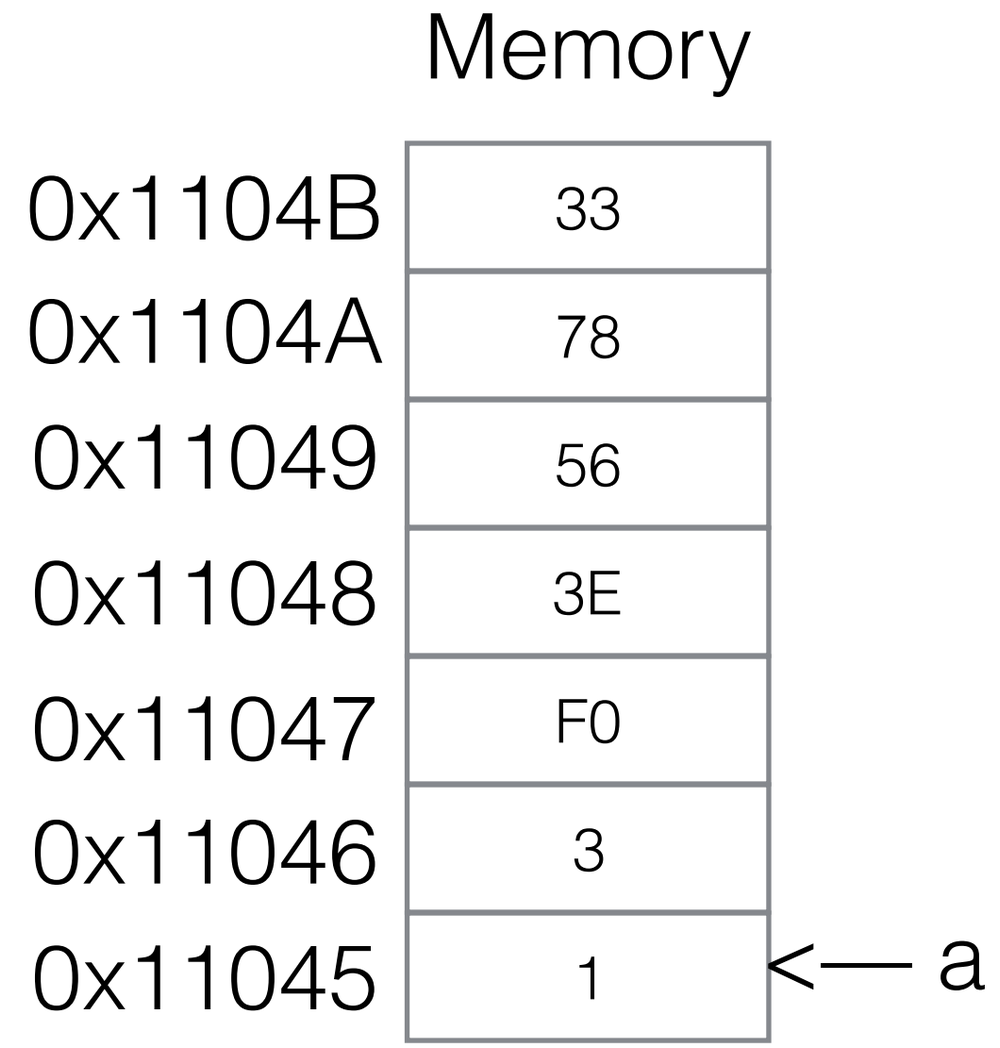
2 New Registers!

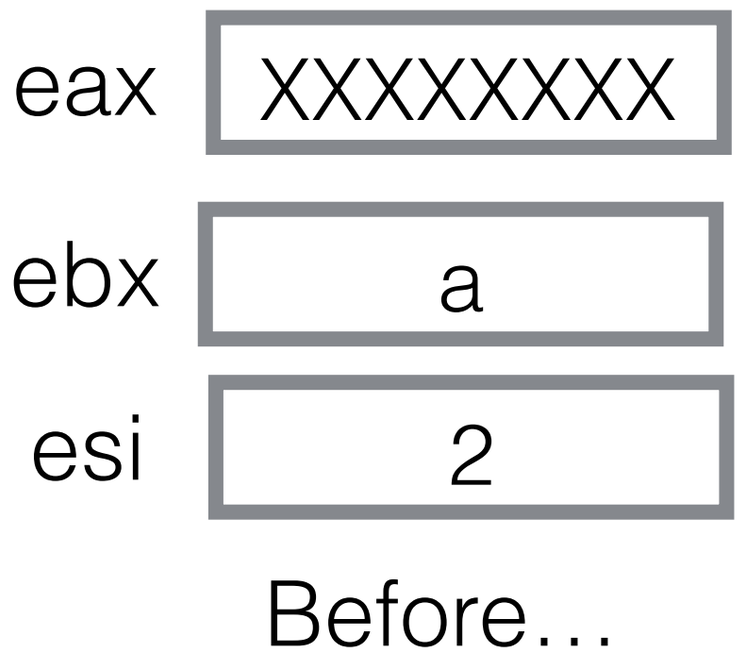
(By the way, we can use esi and edi in indirect mode)

- 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]
```

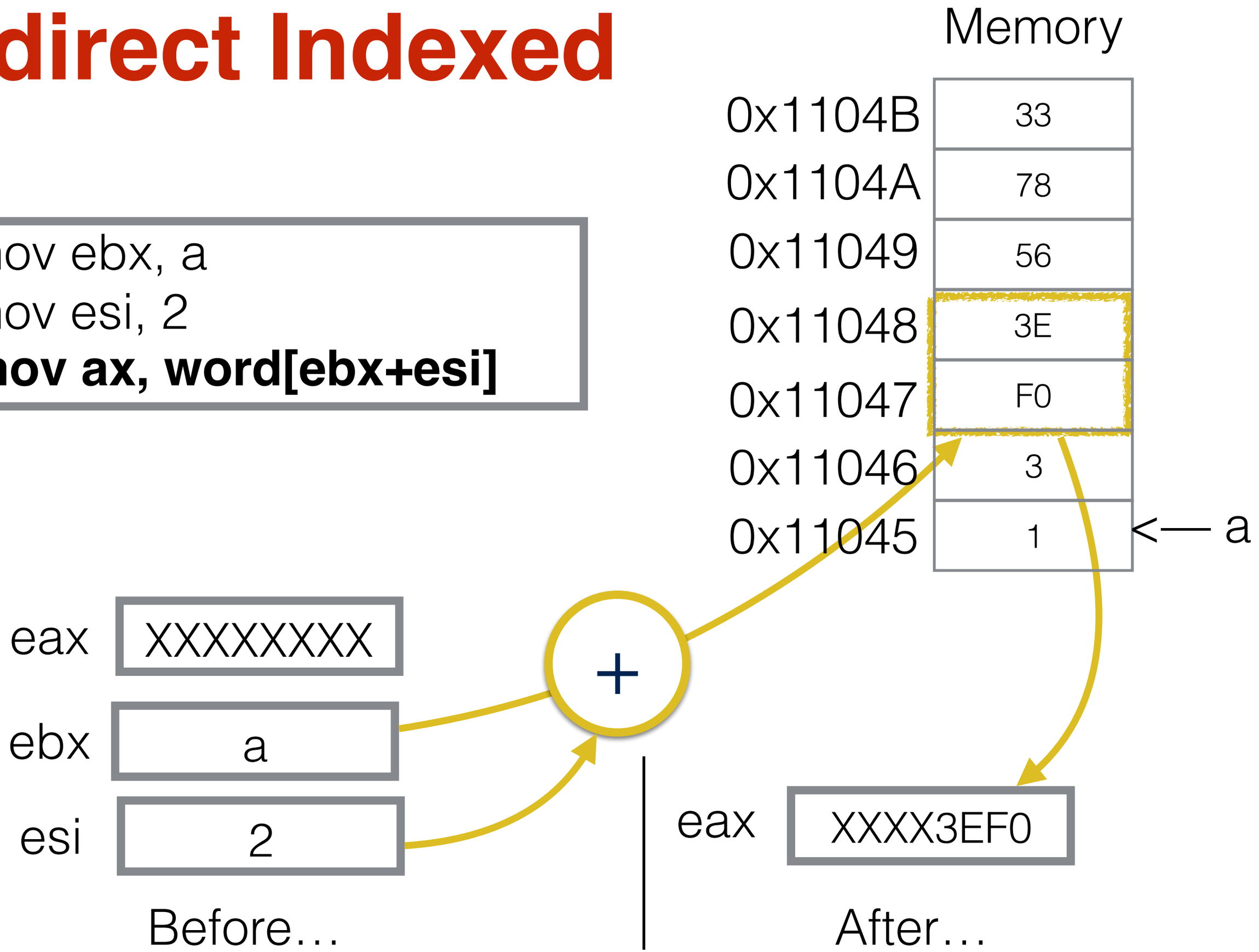




Before…

Indirect Indexed

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



- 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]
```

eax	XXXXXXXX
ebx	a
esi	2

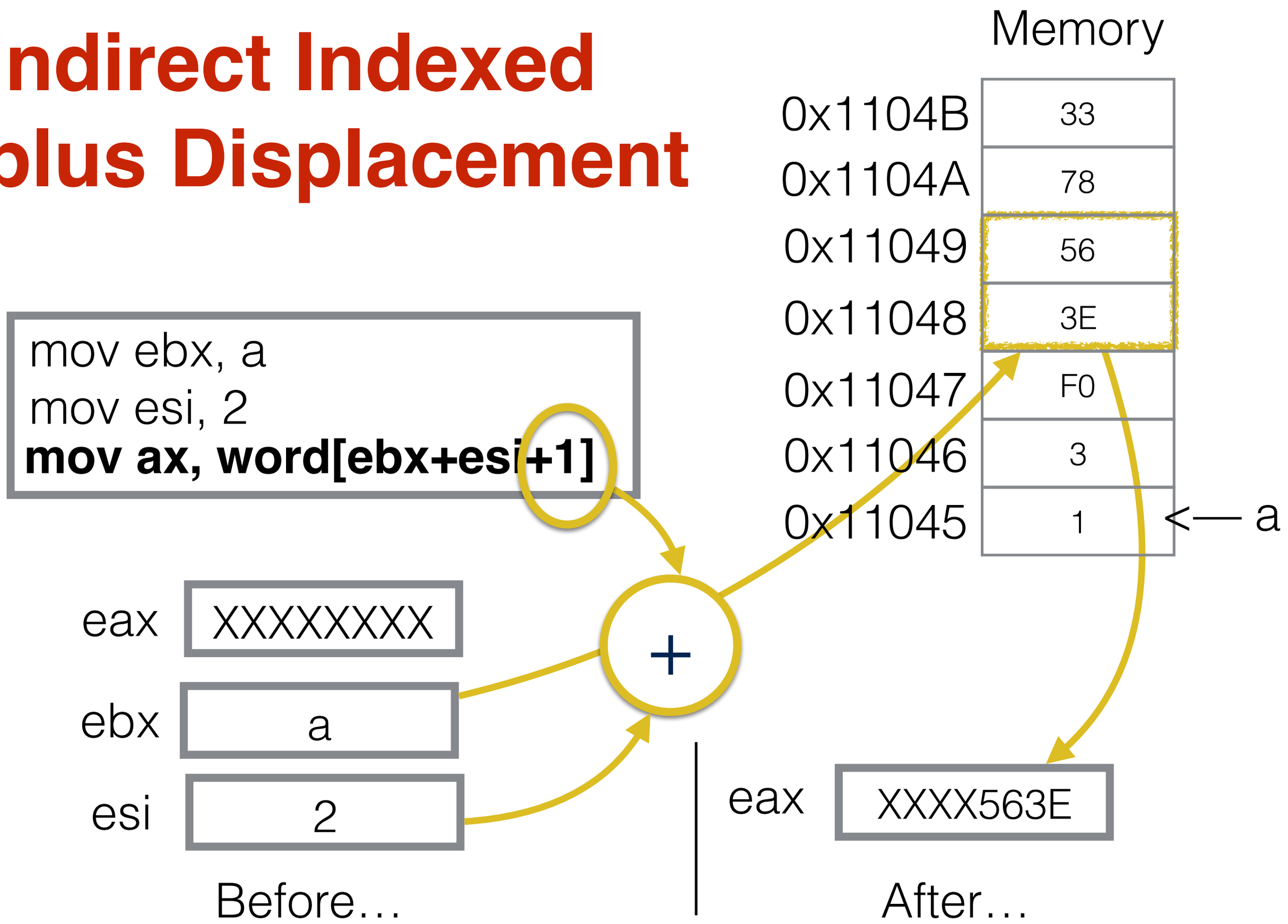
Before…

Memory

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

Indirect Indexed plus Displacement

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



```
;;;  ---------------------------------------------------------
;;;  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
```

Solution 2

```
msg     db      "HELLOTHEREHOWAREYOU"
MSGLEN  equ     $-msg

        mov     ebx, msg        ; ebx points to 1st char of msg
        mov     ecx, MSGLEN     ; # of chars in string
for:    sub     byte[ebx],32    ; lower to upper case, in memory
        inc     ebx             ; ebx points to next char
        loop    for
```

Solution 3

```
Table   db      0,0,0,0,0,0,0,0
N       equ     8

        mov     ebx, Table          ; ebx points to Table[0]
        mov     esi, 0              ; esi is index 0
        mov     al, 1               ; first power of 2
        mov     ecx, N              ; number of items in Table
for:    mov     byte[ebx+esi], al   ; Table[i] <- al
        inc     esi                 ; point to next uncomputed fib
        add     al, al              ; al <- 2*al
        loop    for                 ; go back
```

Solution 4

```
fib     dw      0, 0, 0, 0, 0, 0, 0, 0
        dw      0, 0, 0, 0, 0, 0, 0, 0
NOFIB   equ     16

        mov     word[fib], 1            ; init fib[0]
        mov     word[fib+2], 1          ; init fib[1]

        mov     ebx, fib                ; ebx points to fib
        mov     esi, 2*2                ; esi is index 2
        mov     ecx, NOFIB-2            ; compute 14 fib in loop
        mov     ax, word[ebx+esi-1*2]   ; ax <- fib[0]

for:    mov     ax, word[ebx+esi-1*2]
        add     ax, word[ebx+esi-2*2]   ; ax <- fib[0]+fib[1]
        mov     word[ebx+esi], ax       ; fib[i] <- ax
        add     esi,2                   ; esi now index of next fib
        loop    for
```

Solution 5

```
Powers  dd      0,0,0,0,0,0,0,0,0,0
NOPOW   equ     10

        mov     ebx, Powers     ; ebx points to powers
        mov     eax, 1
        mov     ecx, NOPOW      ; ready to loop 9 times

for:    mov     dword[ebx], eax ; cell of array
        add     ebx, 4          ; point to next empty cell
        add     eax, eax        ; eax <- eax * 2
        loop    for             ; go 10 times
```

Solution 6

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

        mov     ebx, msg1
        mov     esi, 0
        mov     edi, msg2+MSGLEN-1
        mov     ecx, MSGLEN

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