CSC231—Assembly

Week #9 — Fall 2017

Dominique Thiébaut
dthiebaut@smith.edu
Looping Through Arrays
Looping Through Arrays

Loop INSTRUCTION

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)
Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1104C</td>
<td>12</td>
</tr>
<tr>
<td>0x1104B</td>
<td>33</td>
</tr>
<tr>
<td>0x1104A</td>
<td>78</td>
</tr>
<tr>
<td>0x11049</td>
<td>56</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11047</td>
<td>F0</td>
</tr>
<tr>
<td>0x11046</td>
<td>3</td>
</tr>
<tr>
<td>0x11045</td>
<td>1</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Memory</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1104C</td>
<td>12</td>
</tr>
<tr>
<td>0x1104B</td>
<td>33</td>
</tr>
<tr>
<td>0x1104A</td>
<td>78</td>
</tr>
<tr>
<td>0x11049</td>
<td>56</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11047</td>
<td>F0</td>
</tr>
<tr>
<td>0x11046</td>
<td>3</td>
</tr>
<tr>
<td>0x11045</td>
<td>1</td>
</tr>
</tbody>
</table>

**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
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
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
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
Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11045</td>
<td>0x10</td>
</tr>
<tr>
<td>0x11046</td>
<td>0x33</td>
</tr>
<tr>
<td>0x11047</td>
<td>0x56</td>
</tr>
<tr>
<td>0x11048</td>
<td>0x3E</td>
</tr>
<tr>
<td>0x11049</td>
<td>0x78</td>
</tr>
<tr>
<td>0x1104A</td>
<td>0xF0</td>
</tr>
<tr>
<td>0x1104B</td>
<td>0x12</td>
</tr>
</tbody>
</table>

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?

```assembly
section .data

DNA     db      "AGCTANATTTTAGC... ", "GGTC... ", ...
 db      "GCCCTTTTTAAAA"

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    for

Another way to reserve bytes in memory
We stopped here last time…
Friday Lab: Video Introduction
Understanding Indirect Addressing
<table>
<thead>
<tr>
<th>Index</th>
<th>DNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'A'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1101398</td>
<td>'G'</td>
</tr>
<tr>
<td>1101399</td>
<td>'G'</td>
</tr>
<tr>
<td>1101400</td>
<td>'T'</td>
</tr>
<tr>
<td>1101401</td>
<td>'T'</td>
</tr>
<tr>
<td>1101402</td>
<td>'G'</td>
</tr>
<tr>
<td>1101403</td>
<td>'G'</td>
</tr>
<tr>
<td>1101404</td>
<td>'C'</td>
</tr>
<tr>
<td>1101405</td>
<td>'T'</td>
</tr>
<tr>
<td>1101406</td>
<td>'A'</td>
</tr>
</tbody>
</table>

Java Array:
<table>
<thead>
<tr>
<th>Index</th>
<th>DNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101388</td>
<td>'G'</td>
</tr>
<tr>
<td>1101389</td>
<td>'G'</td>
</tr>
<tr>
<td>1101390</td>
<td>'T'</td>
</tr>
<tr>
<td>1101391</td>
<td>'T'</td>
</tr>
<tr>
<td>1101392</td>
<td>'C'</td>
</tr>
<tr>
<td>1101393</td>
<td>'G'</td>
</tr>
<tr>
<td>1101394</td>
<td>'T'</td>
</tr>
<tr>
<td>1101395</td>
<td>'A'</td>
</tr>
<tr>
<td>1101396</td>
<td>'A'</td>
</tr>
</tbody>
</table>

- Store 'G' at Index 1101390.
Store 'G' at Indexes 1101388 to 1101396.
Addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>DNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001396</td>
<td>'A'</td>
</tr>
<tr>
<td>1101395</td>
<td>'A'</td>
</tr>
<tr>
<td>1101394</td>
<td>'T'</td>
</tr>
<tr>
<td>1101393</td>
<td>'G'</td>
</tr>
<tr>
<td>1101392</td>
<td>'C'</td>
</tr>
<tr>
<td>1101391</td>
<td>'T'</td>
</tr>
<tr>
<td>1101390</td>
<td>'T'</td>
</tr>
<tr>
<td>1101389</td>
<td>'G'</td>
</tr>
<tr>
<td>1101388</td>
<td>'G'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Nasm label
Addresses

<table>
<thead>
<tr>
<th>Addresses</th>
<th>DNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001396</td>
<td>'A'</td>
</tr>
<tr>
<td>1101395</td>
<td>'A'</td>
</tr>
<tr>
<td>1101394</td>
<td>'T'</td>
</tr>
<tr>
<td>1101393</td>
<td>'G'</td>
</tr>
<tr>
<td>1101392</td>
<td>'C'</td>
</tr>
<tr>
<td>1101391</td>
<td>'T'</td>
</tr>
<tr>
<td>1101390</td>
<td>'T'</td>
</tr>
<tr>
<td>1101389</td>
<td>'G'</td>
</tr>
<tr>
<td>1101388</td>
<td>'G'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Store 'G' at Address 1101390

Nasm label
### DNA

<table>
<thead>
<tr>
<th>Addresses</th>
<th>DNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001396</td>
<td>'A'</td>
</tr>
<tr>
<td>1101395</td>
<td>'A'</td>
</tr>
<tr>
<td>1101394</td>
<td>'T'</td>
</tr>
<tr>
<td>1101393</td>
<td>'G'</td>
</tr>
<tr>
<td>1101392</td>
<td>'C'</td>
</tr>
<tr>
<td>1101391</td>
<td>'T'</td>
</tr>
<tr>
<td>1101390</td>
<td>'T'</td>
</tr>
<tr>
<td>1101389</td>
<td>'G'</td>
</tr>
<tr>
<td>1101388</td>
<td>'G'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Address Store**

- Store 'G' at Addresses 1101388 to 1101396

**Nasm label**

- D. Thiebaut, Computer Science, Smith College
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  XXXXXXXXX

Before…
Immediate

mov ax, 0x1122

Before…

After…
```python
# code area

section .text

global _start

_start:
  mov eax, 4
  ; print
  mov ebx, 1
  ; to stdout
  mov ecx, message
  ; string
  mov edx, msgLen
  ; # of chars
  int 0x80
  ; ask Linux to print

;;; exit()

  mov eax, 1
  mov ebx, 0
```
• Immediate

• Direct

• Indirect

• Indirect plus Displacement

• Indirect Indexed

• Indirect Indexed plus Displacement
Direct

mov eax, dword[a]

Memory

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1104B</td>
<td>33</td>
</tr>
<tr>
<td>0x1104A</td>
<td>78</td>
</tr>
<tr>
<td>0x11049</td>
<td>56</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11047</td>
<td>F0</td>
</tr>
<tr>
<td>0x11046</td>
<td>3</td>
</tr>
<tr>
<td>0x11045</td>
<td>1</td>
</tr>
</tbody>
</table>

Before...

eax 00000000
Direct

mov eax, dword[a]

Before…

eax 00000000

After…

eax 3378563E

Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1104B</td>
<td>33</td>
</tr>
<tr>
<td>0x1104A</td>
<td>78</td>
</tr>
<tr>
<td>0x11049</td>
<td>56</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11047</td>
<td>F0</td>
</tr>
<tr>
<td>0x11046</td>
<td>3</td>
</tr>
<tr>
<td>0x11045</td>
<td>1</td>
</tr>
</tbody>
</table>
• Immediate

• Direct

• **Indirect**
  
  • Indirect plus Displacement
  
  • Indirect Indexed

  • Indirect Indexed plus Displacement
Indirect

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

Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1104B</td>
<td>33</td>
</tr>
<tr>
<td>0x1104A</td>
<td>78</td>
</tr>
<tr>
<td>0x11049</td>
<td>56</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11047</td>
<td>F0</td>
</tr>
<tr>
<td>0x11046</td>
<td>3</td>
</tr>
<tr>
<td>0x11045</td>
<td>1</td>
</tr>
</tbody>
</table>

Before...

`````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````
Indirect

mov ebx, a

mov eax, dword[ebx]

Before...

eax

 ebx

 a

After...

 eax

 3378563E

Memory
• Immediate

• Direct

• Indirect

• **Indirect plus Displacement**
  • Indirect Indexed
  • Indirect Indexed plus Displacement
**Indirect plus Dispt.**

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

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

Memory

Before...

```assembly
mov eax, dword[ebx+3]
```
**Indirect plus Dispt.**

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

### Before...

- **eax**: 00000000
- **ebx**: a

### After...

- **eax**: 3378563E

---

**Memory**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11045</td>
<td>33</td>
</tr>
<tr>
<td>0x11046</td>
<td>78</td>
</tr>
<tr>
<td>0x11047</td>
<td>56</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11049</td>
<td>F0</td>
</tr>
<tr>
<td>0x1104A</td>
<td>3</td>
</tr>
<tr>
<td>0x1104B</td>
<td>1</td>
</tr>
</tbody>
</table>
- Immediate
- Direct
- Indirect
- Indirect plus Displacement

- **Indirect Indexed**

- Indirect Indexed plus Displacement
2 New Registers!

eax
ebx
ecx
edx
2 New Registers!

eax
ebx
cdx
edx
eip
2 New Registers!

"i" in esi, edi for **index**
"s" for **source**, "d" for **destination**
2 New Registers!

Data Registers:
- eax
- ebx
- ecx
- edx

Index Registers:
- eip
- esi
- edi
• 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...

```
| eax | X|X|X|X|X|X|X|X|X|
| ebx | a |
| esi | 2 |
```

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

\[ a \leftarrow 1 \]
Indirect Indexed

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

Before...

- ebx: a
- esi: 2
- eax: XXXXXXXX

After...

- ebx: a
- esi: 2
- eax: XXXX3EF0

Memory:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11045</td>
<td>33</td>
</tr>
<tr>
<td>0x11046</td>
<td>78</td>
</tr>
<tr>
<td>0x11047</td>
<td>56</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11049</td>
<td>F0</td>
</tr>
<tr>
<td>0x1104A</td>
<td>3</td>
</tr>
<tr>
<td>0x1104B</td>
<td>1</td>
</tr>
</tbody>
</table>
• 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

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1104B</td>
<td>33</td>
</tr>
<tr>
<td>0x1104A</td>
<td>78</td>
</tr>
<tr>
<td>0x11049</td>
<td>56</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11047</td>
<td>F0</td>
</tr>
<tr>
<td>0x11046</td>
<td>3</td>
</tr>
<tr>
<td>0x11045</td>
<td>1</td>
</tr>
</tbody>
</table>

eax

Before…

ebx

esi

2
Indirect Indexed plus Displacement

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

Before...

```
eax
ebx
esi
```

After...

```
eax
```

Memory:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11045</td>
<td>1</td>
</tr>
<tr>
<td>0x11046</td>
<td>3</td>
</tr>
<tr>
<td>0x11047</td>
<td>F0</td>
</tr>
<tr>
<td>0x11048</td>
<td>3E</td>
</tr>
<tr>
<td>0x11049</td>
<td>56</td>
</tr>
<tr>
<td>0x1104A</td>
<td>78</td>
</tr>
<tr>
<td>0x1104B</td>
<td>33</td>
</tr>
</tbody>
</table>
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.

```assembly
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      "HELLO THERE HOW ARE YOU"
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:    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
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 5
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