CSC231 - Assembly

Week #8
Fall 2019

Dominique Thiébaut
dthiebaut@smith.edu
Printing ints

- Two options:
  - print as unsigned
  - print as 2's complement signed
section .data
alpha dd 0x0000000F ; just 15
beta dd 0x7FFFFFFF ; the largest number with a leading 0
gamma dd 0x80000000 ; the first "logical" number with a leading bit equal to 1
delta dd 0xFFFFFFFF ; the largest binary pattern with all 1s

section .text
_global _start

_start:
    mov eax, dword[alpha]
call _printInt
    call _println
call _printDec
    call _println
    call _println
    mov eax, dword[beta]
call _printInt
    call _println
call _printDec
    call _println
    call _println
    mov eax, dword[gamma]
call _printInt
    call _println
call _printDec
    call _println
    call _println
    mov eax, dword[delta]
call _printInt
    call _println
call _printDec
    call _println
    call _println

getcopy printSignedUnsigned.asm
section .data
alpha dd 0x0000000F  ; just 15
beta dd 0x7FFFFFFF  ; the largest number with a leading 0
gamma dd 0x80000000  ; the first "logical" number with a leading bit equal to 1
delta dd 0xFFFFFFFF  ; the largest binary pattern with all 1s

section .text
  global _start

_start:
  mov eax, dword[alpha]
call _printInt
call _println
call _println
  mov eax, dword[beta]
call _printDec
call _println
call _println
call _println
  mov eax, dword[gamma]
call _printInt
call _println
call _println
call _println
  mov eax, dword[delta]
call _printInt
call _println
call _println
call _println

cs231a@marax ~$ nasm -f elf printSignedUnsigned.asm
cs231a@marax ~ $ ld -melf_i386 printSignedUnsigned.o 231Lib.o -o printSignedUnsigned
cs231a@marax ~ $ ./printSignedUnsigned
15
15
2147483647
2147483647
-2147483648
2147483648
-1
4294967295
section .data
alpha dd 0x0000000F ; just 15
beta dd 0x7FFFFFFF ; the largest number with a leading 0
gamma dd 0x80000000 ; the first "logical" number with a leading bit equal to 1
delta dd 0xFFFFFFFF ; the largest binary pattern with all 1s

section .text
global _start
_start:
    mov eax, dword[alpha]
call __printInt
call __println
call __printDec
call __println
    mov eax, dword[beta]
call __printInt
call __println
call __printDec
call __println
    mov eax, dword[gamma]
call __printInt
call __println
call __printDec
call __println
    mov eax, dword[delta]
call __printInt
call __println
call __printDec
call __println

_cs231a@marax ~$ nasm -f elf printSignedUnsigned.asm
_cs231a@marax ~ $ ld -m elf_i386 printSignedUnsigned.o 231Lib.o -o printSignedUnsigned
_cs231a@marax ~ $ ./printSignedUnsigned
15
2147483647
2147483647
-2147483648
2147483648
-1
4294967295
Do you get this?

https://xkcd.com/
Outline

• Looping & Arrays
• Fibonaccis without Arrays
• Looping through Arrays: Indirect Addressing
• Addressing Modes
• The IF Statement: Game of Life
Example 2
Fibonaccis

http://i.dailymail.co.uk/i/pix/2016/07/17/13/1A32203E000005DC-3694326-image-m-17_1468760305397.jpg
Example 2
Fibonacci numbers

1, 1, 2, 3, 5, 8, 13, 21, 34...
; print the first 10 Fibonacci terms

_start:
    mov    eax, 1       ; fibn
    mov    ebx, 1       ; fibn-1
    call   _printDec   ; (print as unsigned)
    call   _println

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

for:     mov    edx, ebx     ; edx becomes fibn-2
         mov    ebx, eax     ; ebx becomes fibn-1
         add    eax, edx     ; eax becomes fibn
         call   _printDec   ; (print as unsigned)
         call   _println
         loop   for

getcopy fib.asm
Exercise

What if we start a loop with 0 in ecx?

mov ecx, 0
mov eax, 1
for:
call _printDec
call _println
inc eax
loop for

How many lines are printed by this program?
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
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
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>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>10</td>
</tr>
</tbody>
</table>

### Section .data

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

### Section .text

```assembly
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>0</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 'z'</td>
</tr>
<tr>
<td>0x1104B</td>
<td>33</td>
</tr>
<tr>
<td>0x1104C</td>
<td>12</td>
</tr>
</tbody>
</table>

### Section `.data`

- `A`:
  ```
  A       db   1,3,0xF0,0x3E,0x56
  ```
- `B`:
  ```
  B       db   0x78,0x33,0x12
  ```

### Section `.text`

```plaintext
_start:
  mov    al, 'z'
  mov    ebx, A
  mov    byte[ebx], 0
  mov    ebx, B
  mov    byte[ebx], al
```
Example 2: Filling an Array with 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
Exercise

Problem #1:
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.
| Char | Dec | Oct | Hex | | Char | Dec | Oct | Hex | | Char | Dec | Oct | Hex |
|------|-----|-----|-----| | | | | | | | | | | |
| (sp) | 32  | 0040 | 0x20 | | @ | 64 | 0100 | 0x40 | | ` | 96 | 0140 | 0x60 |
| !    | 33  | 0041 | 0x21 | | A | 65 | 0101 | 0x41 | | a | 97 | 0141 | 0x61 |
| "    | 34  | 0042 | 0x22 | | B | 66 | 0102 | 0x42 | | b | 98 | 0142 | 0x62 |
| #    | 35  | 0043 | 0x23 | | C | 67 | 0103 | 0x43 | | c | 99 | 0143 | 0x63 |
| $    | 36  | 0044 | 0x24 | | D | 68 | 0104 | 0x44 | | d | 100 | 0144 | 0x64 |
| %    | 37  | 0045 | 0x25 | | E | 69 | 0105 | 0x45 | | e | 101 | 0145 | 0x65 |
| &    | 38  | 0046 | 0x26 | | F | 70 | 0106 | 0x46 | | f | 102 | 0146 | 0x66 |
| '    | 39  | 0047 | 0x27 | | G | 71 | 0107 | 0x47 | | g | 103 | 0147 | 0x67 |
| (    | 40  | 0050 | 0x28 | | H | 72 | 0110 | 0x48 | | h | 104 | 0150 | 0x68 |
| )    | 41  | 0051 | 0x29 | | I | 73 | 0111 | 0x49 | | i | 105 | 0151 | 0x69 |
| *    | 42  | 0052 | 0x2a | | J | 74 | 0112 | 0x4a | | j | 106 | 0152 | 0x6a |
| +    | 43  | 0053 | 0x2b | | K | 75 | 0113 | 0x4b | | k | 107 | 0153 | 0x6b |
| ,    | 44  | 0054 | 0x2c | | L | 76 | 0114 | 0x4c | | l | 108 | 0154 | 0x6c |
| -    | 45  | 0055 | 0x2d | | M | 77 | 0115 | 0x4d | | m | 109 | 0155 | 0x6d |
| ,    | 46  | 0056 | 0x2e | | N | 78 | 0116 | 0x4e | | n | 110 | 0156 | 0x6e |
| /    | 47  | 0057 | 0x2f | | O | 79 | 0117 | 0x4f | | o | 111 | 0157 | 0x6f |
| 0    | 48  | 0060 | 0x30 | | P | 80 | 0120 | 0x50 | | p | 112 | 0160 | 0x70 |
| 1    | 49  | 0061 | 0x31 | | Q | 81 | 0121 | 0x51 | | q | 113 | 0161 | 0x71 |
| 2    | 50  | 0062 | 0x32 | | R | 82 | 0122 | 0x52 | | r | 114 | 0162 | 0x72 |
| 3    | 51  | 0063 | 0x33 | | S | 83 | 0123 | 0x53 | | s | 115 | 0163 | 0x73 |
| 4    | 52  | 0064 | 0x34 | | T | 84 | 0124 | 0x54 | | t | 116 | 0164 | 0x74 |
| 5    | 53  | 0065 | 0x35 | | U | 85 | 0125 | 0x55 | | u | 117 | 0165 | 0x75 |
| 6    | 54  | 0066 | 0x36 | | V | 86 | 0126 | 0x56 | | v | 118 | 0166 | 0x76 |
| 7    | 55  | 0067 | 0x37 | | W | 87 | 0127 | 0x57 | | w | 119 | 0167 | 0x77 |
| 8    | 56  | 0070 | 0x38 | | X | 88 | 0130 | 0x58 | | x | 120 | 0170 | 0x78 |
| 9    | 57  | 0071 | 0x39 | | Y | 89 | 0131 | 0x59 | | y | 121 | 0171 | 0x79 |
| ;    | 58  | 0072 | 0x3a | | Z | 90 | 0132 | 0x5a | | z | 122 | 0172 | 0x7a |
| <    | 59  | 0073 | 0x3b | | | 91 | 0133 | 0x5b | | { | 123 | 0173 | 0x7b |
| =    | 60  | 0074 | 0x3c | | \ | 92 | 0134 | 0x5c | | | 124 | 0174 | 0x7c |
| >    | 61  | 0075 | 0x3d | | | 93 | 0135 | 0x5d | | | 125 | 0175 | 0x7d |
| ?    | 62  | 0076 | 0x3e | | ^ | 94 | 0136 | 0x5e | | | 126 | 0176 | 0x7e |
|      | 63  | 0077 | 0x3f | | _ | 95 | 0137 | 0x5f | | | | |

D. Thiebaut, Computer Science, Smith College
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 "GCCCTTTTTAAAA"
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 XXXXXXXXX

Before…
Immediate

mov ax, 0x1122

Before…

After…

eax XXXXXXXX

eax XXXX1122
Immediate

\[ a \text{ db } "Hello" \]

\[ \text{...} \]

\[ \text{mov eax, a} \]

\[ \text{eax } X \]

Before…

\[ \text{eax } 0x12345678 \]

After…
• Immediate

• Direct
  • Indirect
  • Indirect plus Displacement
  • Indirect Indexed
  • Indirect Indexed plus Displacement
Direct

mov eax, dword[a]

eax 00000000

Before…

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>
Direct

```assembly
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

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

Before...

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

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

Before…

Memory

<table>
<thead>
<tr>
<th>Address</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>

After…

```assembly
eax
```

```assembly
3378563E
```
• Immediate

• Direct

• Indirect

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

Before…

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

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 00000000
ebx a

← a
Indirect plus Dispt.

Before...

mov ebx, a

mov eax, dword[ebx+3]

After...

eax 3378563E

ebx a

eax 00000000

Memory

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

<-- a

Indirect plus Dispt.
• Immediate

• Direct

• Indirect

• Indirect plus Displacement

• **Indirect Indexed**

• Indirect Indexed plus Displacement
2 New Registers!

<table>
<thead>
<tr>
<th>eax</th>
<th>ebx</th>
<th>ecx</th>
<th>edx</th>
</tr>
</thead>
</table>

D. Thiebaut, Computer Science, Smith College
2 New Registers!

eax  eip
ebx
ecx
edx
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]

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...

eax  XYYYYYYYYY
ebx  a
esi  2
Indirect Indexed

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

Before…

eax
ebx
esi

2
a

XXX

After…

eax

XXX3EF0

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

<— a
• 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]

Before...

eax

ebx

esi

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>

← a
Indirect Indexed plus Displacement

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

Before…

<table>
<thead>
<tr>
<th>eax</th>
<th>XXXXXXXXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebx</td>
<td>a</td>
</tr>
<tr>
<td>esi</td>
<td>2</td>
</tr>
</tbody>
</table>

After…

<table>
<thead>
<tr>
<th>eax</th>
<th>XXX563E</th>
</tr>
</thead>
</table>

Memory

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

← a
We Stopped Here Last Time
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.
The IF-Statement
2 Videos to Watch at a Later Time…

https://www.youtube.com/watch?v=FdMzngWchDk

https://www.youtube.com/watch?v=k2lZ1qsx4CM
https://www.youtube.com/watch?v=CgOcEZinQ2I
A 1-D Version
A 1-D Version
A 1-D Version

```
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
```
Rules of Life

**Rule 1:** 0 neighbors
Rules of Life

**Rule 1**: 0 neighbors

Underpopulation
Rules of Life

Rule 2: 1 neighbor
Rules of Life

Rule 2: 1 neighbor

Right Environment
Rules of Life

Rule 3: 2 neighbors
Rule 3: 2 neighbors
Problem of the Day(s): Implement 1D Game of Life in Assembly!
How to Approach This?
#Step 1: Write Algorithm in an More Comfortable Language...
# gameOfLife.py
# D. Thiebaut
# 1-Dimensional Game of Life

def life( dish, N ):
    newGen = ""
    for i in range( 0, N ):
        neighbors = 0
        if i>0 and dish[i-1]!="": neighbors += 1
        if i < N-1 and dish[i+1]!="": neighbors += 1
        if neighbors == 1:
            newGen += "#"
        else:
            newGen += " "
    return newGen

def main():
    N = 40
    dish = (N//2-10)*"#" + 10*" #" + (N//2-10)*" "
    dish = dish[0:N]

    # print first generation
    print( dish )

    # repeat, for some generations
    for generation in range( 20 ):
        newGen = life( dish, N )
        print( newGen )
        dish = newGen

main()
We stopped here last time…
I wrote the start of the game of life 1D and seg-faulted on the first try…
Heloise figured out that I forgotten to save eax and ebx…