Introduction aux Systèmes d'Exploitation

# Unit 7: Loops, Conditions, and Addressing Modes

François Taïani

# Outline

- Loops
- If statements
- Addressing Modes
- Bonus

# Typical Construct : Loop

- Reminder : `cmp`, `jle`, `mov`, `dec`
- How would you implement the following in x86 asm?
  - ➔ (using the `rbx` register for i)

```
int i;

for(i=15; i>0;i--) {
  // Do something
  printf("%i\n",i);
} // EndFor
```

# Typical Construct : Loop

```
  mov       rbx, 0xf            ; start value: i = 15
begin:
  cmp       rbx,0               ; if i <= 0 then go to end
  jle       end





  dec       rbx
  jmp       begin
end:
  ...
```

# Typical Construct : Loop

```
SECTION      .data
message:     db        " ", 10      ; note the newline at the end
msgLen:      equ $-message

SECTION     .text
  GLOBAL     _start
_start:
  mov        rbx, 0xf               ; start value: i = 15
begin:
  cmp        rbx,0                  ; if i <= 0 then go to end
  jle        end
  mov        rcx,rbx                ; we use rcx to compute the ascii code
  add        rcx,'0'                ; of the character we want to print
  mov        [message],cl           ; and move this character into the message string
  mov        rax, 1                 ; system call for write
  mov        rdi, 1                 ; file handle 1 is stdout
  mov        rsi, message           ; address of string to output
  mov        rdx, msgLen            ; number of bytes
  syscall                           ; invoke operating system to do the write
  dec        rbx                    ; no need to save/restore rbx: done in sys. call.
  jmp        begin
end:
  ...
```

# Making sense of the output

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 0000 0000 NUL | 01 0000 0001 SOH | 02 0000 0010 STX | 03 0000 0011 ETX | 04 0000 0100 EOT | 05 0000 0101 ENQ | 06 0000 0110 ACK | 07 0000 0111 BEL | 08 0000 1000 BS | 09 0000 1001 HT | 10 0000 1010 LF | 11 0000 1011 VT | 12 0000 1100 FF | 13 0000 1101 CR | 14 0000 1110 SO | 15 0000 1111 SI | 8 |
| 1 | 16 0001 0000 DLE | 17 0001 0001 DC1 | 18 0001 0010 DC2 | 19 0001 0011 DC3 | 20 0001 0100 DC4 | 21 0001 0101 NAK | 22 0001 0110 SYN | 23 0001 0111 ETB | 24 0001 1000 CAN | 25 0001 1001 EM | 26 0001 1010 SUB | 27 0001 1011 ESC | 28 0001 1100 FS | 29 0001 1101 GS | 30 0001 1110 RS | 31 0001 1111 US | 9 |
| 2 | 32 0010 0000 SP | 33 0010 0001 ! | 34 0010 0010 " | 35 0010 0011 # | 36 0010 0100 $ | 37 0010 0101 % | 38 0010 0110 & | 39 0010 0111 ' | 40 0010 1000 ( | 41 0010 1001 ) | 42 0010 1010 * | 43 0010 1011 + | 44 0010 1100 , | 45 0010 1101 - | 46 0010 1110 . | 47 0010 1111 / | A |
| 3 | 48 0011 0000 0 | 49 0011 0001 1 | 50 0011 0010 2 | 51 0011 0011 3 | 52 0011 0100 4 | 53 0011 0101 5 | 54 0011 0110 6 | 55 0011 0111 7 | 56 0011 1000 8 | 57 0011 1001 9 | 58 0011 1010 : | 59 0011 1011 ; | 60 0011 1100 < | 61 0011 1101 = | 62 0011 1110 > | 63 0011 1111 ? | B |
| 4 | 64 0100 0000 @ | 65 0100 0001 A | 66 0100 0010 B | 67 0100 0011 C | 68 0100 0100 D | 69 0100 0101 E | 70 0100 0110 F | 71 0100 0111 G | 72 0100 1000 H | 73 0100 1001 I | 74 0100 1010 J | 75 0100 1011 K | 76 0100 1100 L | 77 0100 1101 M | 78 0100 1110 N | 79 0100 1111 O | C |
| 5 | 80 0101 0000 P | 81 0101 0001 Q | 82 0101 0010 R | 83 0101 0011 S | 84 0101 0100 T | 85 0101 0101 U | 86 0101 0110 V | 87 0101 0111 W | 88 0101 1000 X | 89 0101 1001 Y | 90 0101 1010 Z | 91 0101 1011 [ | 92 0101 1100 \ | 93 0101 1101 ] | 94 0101 1110 ^ | 95 0101 1111 _ | D |
| 6 | 96 0110 0000 ` | 97 0110 0001 a | 98 0110 0010 b | 99 0110 0011 c | 100 0110 0100 d | 101 0110 0101 e | 102 0110 0110 f | 103 0110 0111 g | 104 0110 1000 h | 105 0110 1001 i | 106 0110 1010 j | 107 0110 1011 k | 108 0110 1100 l | 109 0110 1101 m | 110 0110 1110 n | 111 0110 1111 o | E |
| 7 | 112 0111 0000 p | 113 0111 0001 q | 114 0111 0010 r | 115 0111 0011 s | 116 0111 0100 t | 117 0111 0101 u | 118 0111 0110 v | 119 0111 0111 w | 120 0111 1000 x | 121 0111 1001 y | 122 0111 1010 z | 123 0111 1011 { | 124 0111 1100 \| | 125 0111 1101 } | 126 0111 1110 ~ | 127 0111 1111 DEL | F |

http://www.ecowin.org/aulas/resources/tables/asciitable.jpg

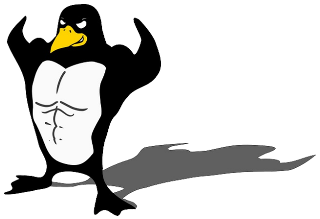
# Optimization (Special Case)

```
  mov       rbx, 0xf              ; start value: 15
begin:






  dec       rbx
  jnz       begin                 ; if rbx!=0 then loop back to begin
end:
  ...
```

# Optimization (Special Case)

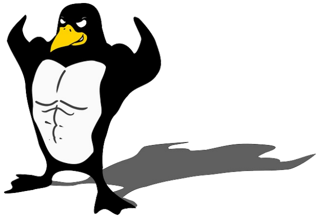
```
SECTION     .data
message:    db        " ", 10      ; note the newline at the end
msgLen:     equ $-message

SECTION    .text
  GLOBAL    _start
_start:
  mov       rbx, 0xf             ; start value: 15
begin:
  mov       rcx,rbx              ; we use rcx to compute the ascii code
  add       rcx,'0'              ; of the character we want to print
  mov       [message],cl         ; and move this character into the message string

  mov       rax, 1               ; system call for write
  mov       rdi, 1               ; file handle 1 is stdout
  mov       rsi, message         ; address of string to output
  mov       rdx, msgLen          ; number of bytes
  syscall                        ; invoke operating system to do the write
  dec       rbx
  jnz       begin                ; if rbx!=0 then loop back to begin
end:
  ...
```

# Can you spot the difference?



```
SECTION     .data
message:    db        " ", 10      ; note the newline at the end
msgLen:     equ $-message

SECTION    .text
  GLOBAL    _start
_start:
  mov       rcx, 0xf              ; start value: 15
begin:
  mov       rbx,rcx               ; we use rcx to compute the ascii code
  add       rbx,'0'               ; of the character we want to print
  mov       [message],bl          ; and move this character into the message string
  push      rcx                   ; (!) saving rcx, as clobered by 'syscall'
  mov       rax, 1                ; system call for write
  mov       rdi, 1                ; file handle 1 is stdout
  mov       rsi, message          ; address of string to output
  mov       rdx, msgLen           ; number of bytes
  syscall                         ; invoke operating system to do the write
  pop       rcx                   ; (!) restoring rcx
  loop      begin                 ; equivalent to 'dec rcx' + 'jnz begin'
end:
  ...
```

# Can you spot the difference?



```
  mov       rcx, 0xf            ; start value: 15
begin:



  loop      begin               ; equivalent to 'dec rcx' + 'jnz begin'
end:
  ...
```

# Typical Construct: If

- How would you implement the following in x86?

```
int x = 5 ;
int y = 10;

if (x < y) {
  printf("X smaller than Y");
} else {
  printf("X greater or equal to Y");
}
```

➔ Tip : define X et Y as memory labels in the .data segments

```
SECTION      .data
x:           dw  5
y:           dw 10
```

# Typical Construct: If

```
  mov       ax, [x]              ; need to use ax as manipulating words
  cmp       ax, [y]              ; (16-bit values)
  jge       sinon
  ...                            ; do something if x<y

  jmp       continue
sinon:
  ...                            ; something else if x≥y

continue:
  ...                            ; leaving if then else block
```

# Typical Construct: If

```
  mov       ax, [x]              ; need to use ax as manipulating words
  cmp       ax, [y]              ; (16-bit values)
  jge       sinon
  mov       rsi,msg1             ; select message 1 to be printed
  mov       rdx,len1
  jmp       continue
sinon:
  mov       rsi,msg2             ; select message 2 to be printed
  mov       rdx,len2
continue:
  mov       rax, 1               ; system call for write
  mov       rdi, 1               ; file handle 1 is stdout
  syscall                        ; invoke operating system to do the write
end:
  ...
```

# Typical Construct: If

```
SECTION      .data
x:           dw 50
y:           dw 10
msg1:        db 'x smaller than y', 10
len1:        equ $-msg1
msg2:        db 'x greater or equal to y', 10
len2:        equ $-msg2
SECTION     .text
  GLOBAL     _start
_start:
  mov        ax, [x]               ; need to use ax as manipulating words
  cmp        ax, [y]               ; (16-bit values)
  jge        sinon
  mov        rsi,msg1              ; select message 1 to be printed
  mov        rdx,len1
  jmp        continue
sinon:
  mov        rsi,msg2              ; select message 2 to be printed
  mov        rdx,len2
continue:
  mov        rax, 1                ; system call for write
  mov        rdi, 1                ; file handle 1 is stdout
  syscall                          ; invoke operating system to do the write
end:
  ...
```

# Addressing Modes

- Refers to how operands are obtained

- We have seen three so far
  - Immediate (`mov rax,0x8`)
  - Register (`mov rax,rbx`)
  - Direct (`mov rax,[100]`)

- But there are more !

# Addressing Modes (cont.)

- **Not involving memory**
  - **Immediate (or litteral)** : `mov rax,0x41` Reminder
  - **Register** : `mov rax, rbx` Reminder
- **Involving memory**
  - **Direct :** `mov rax,[0x16] ; 0x16 can be a label` Reminder
  - **Indirect:** look up address position in a register
    - `mov rax,[rbx] ; [rbx]->memory starting at rbx`
  - **Indirect with displacement**
    - `mov rax,[rbx+10] ;also rbx[10], 10[rbx]`
  - **General case**
    - `mov rax,[rbx+scale*rsi+10] ;rbx[10+rsi*scale]`

scale∈{1,2,4,8}

Constant can be a label

# Addressing Modes (cont.)

- If `message` and `len` contains the following

```
message:     db 'HELLOWORLD',10
len:         equ $-message
```

- How would you downcase the whole string?
  - ➔ Tip: use a loop on `rsi`, use indexed addressing

# Addressing Modes (cont.)

```
  mov      rsi,9
loop:
  add      BYTE [message+rsi],'a'-'A'
  dec      rsi
  jge      loop
```

- How would you do if the string contains a space?
  - ➔ `message:    db 'HELLO WORLD',10`

# Addressing Modes (cont.)

```
  mov    rsi,9
loop:
  cmp    BYTE [message+rsi],' '
  je     cont
  add    BYTE [message+rsi],'a'-'A'
cont:
  dec    rsi
  jge    loop
```

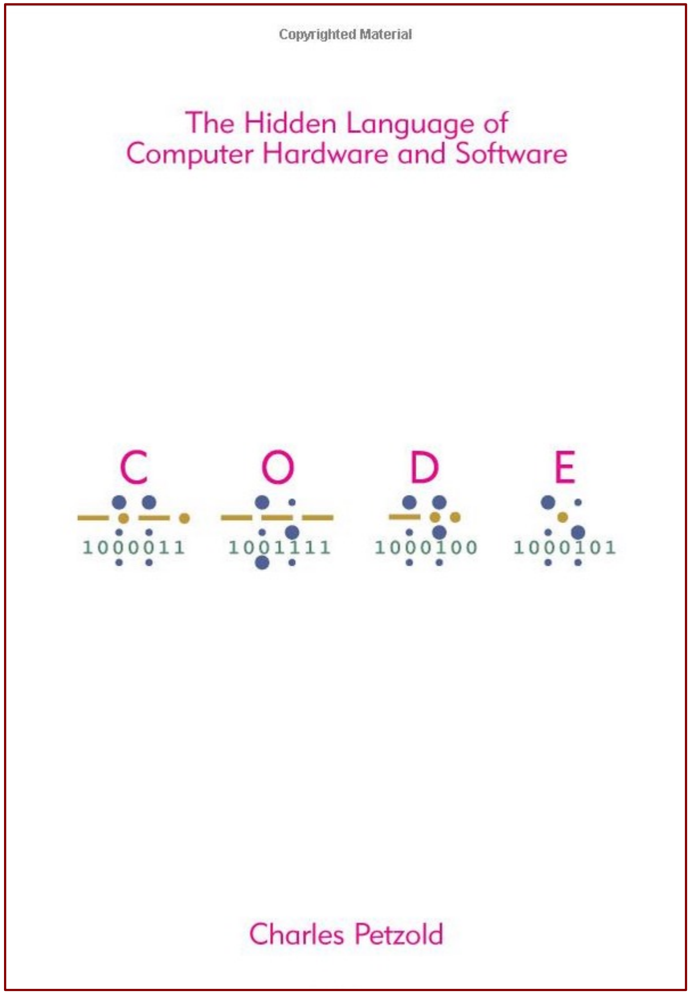
- Homework: strings containing
  - ➔ any non-letters, and lowercase letters
  - ➔ E.g. `message:    db 'HELLO World !!??',10`

# Summary

At the end of this session you should be able to

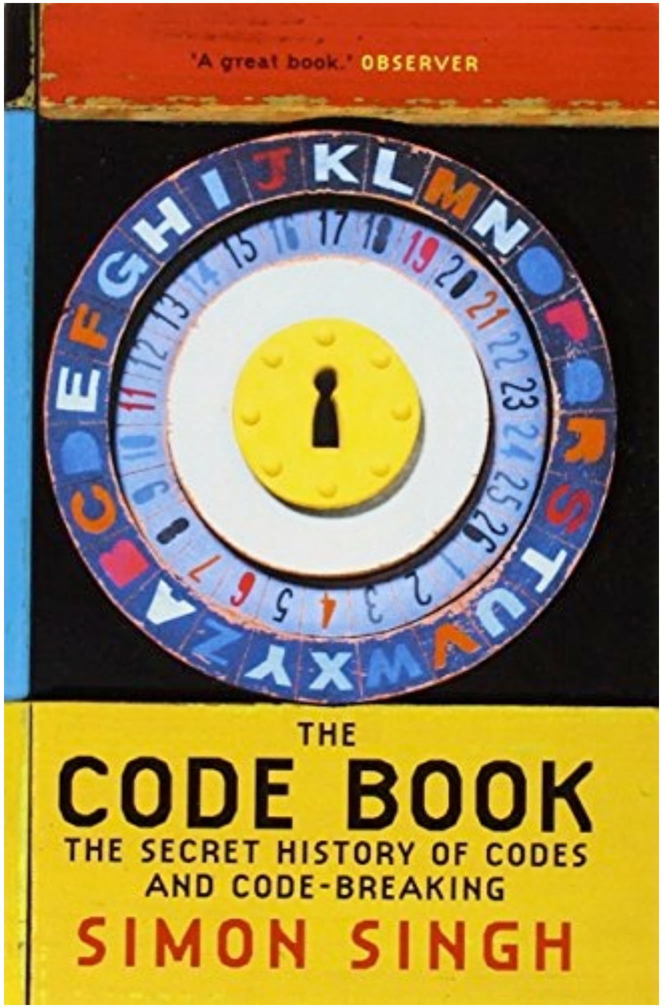
- be able to translate in assembly simple control flow constructs (loops, if-then-else statements)
- understand the main addressing modes of the x86
- draft simple programs in assembly code

# Further Reading




- http://www.charlespetzold.com/code/

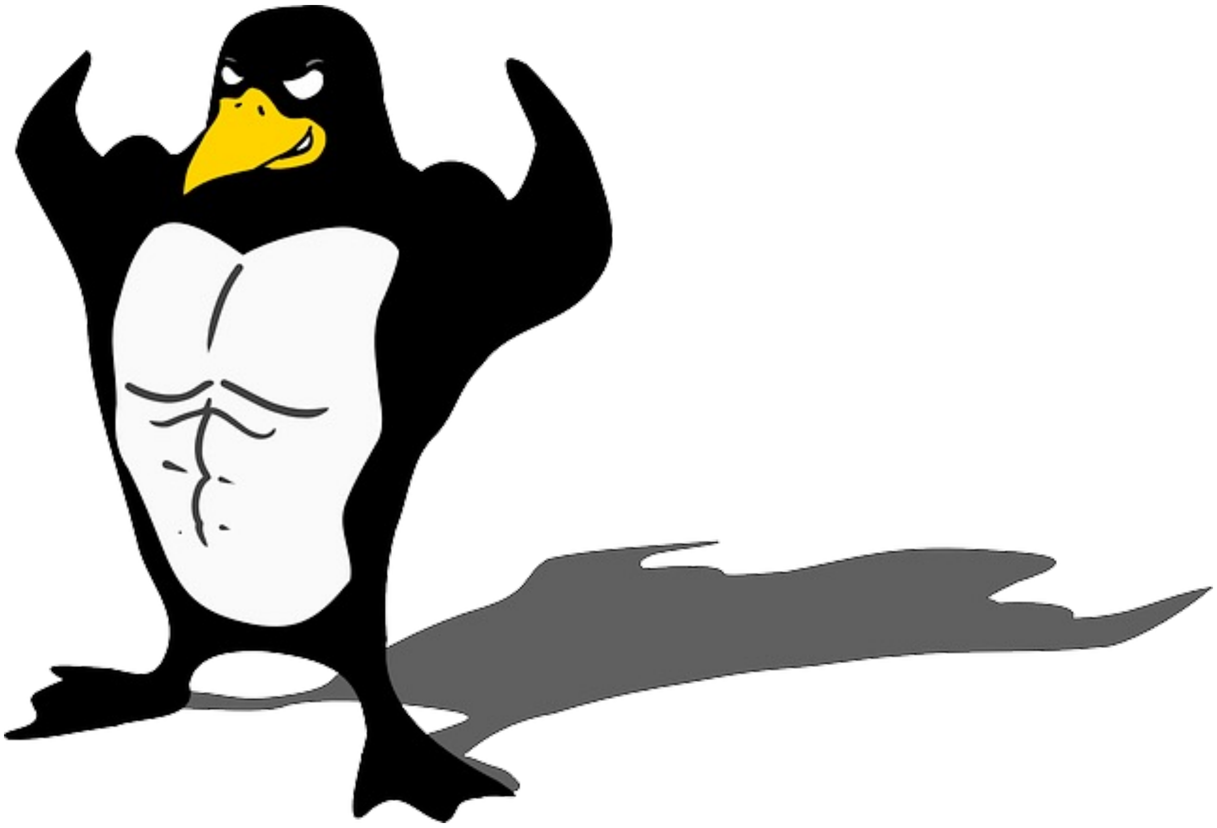
# Even Further Reading




- http://simonsingh.net/books/the-code-book/

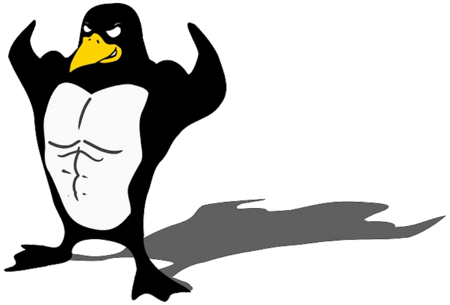
# Bonus Material

- Not exam material



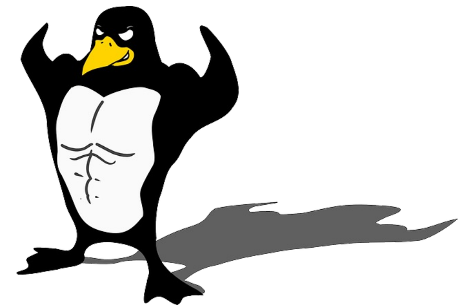
# Operands of different sizes

- ~~`mov` rax, BYTE [message]~~
  - ➔ Does not work: “error: mismatch in operand sizes”
  - ➔ Reason: does not know how to set extra bits of rax



- Two extra `mov` instructions
  - ➔ `movzx` Sets extra bits to zero
  - ➔ `movsx` Sets extra bits to bit of sign (for signed values)

# hello_world.asm on DOS

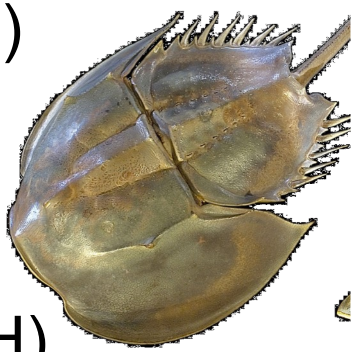


```
.MODEL SMALL          ; memory model
.STACK                ; stack segment
.DATA                 ; data segment
        MESSAGE DB 'HELLO WORLD$'
.CODE
start:
        MOV AX, @DATA                ; initialising DS
        MOV DS, AX                   ; (16bit only)

        MOV DX, offset MESSAGE       ; DX points to string MESSAGE
        MOV AH, 09H                  ; 09H = DOS print routine
        INT 21H                      ; calling DOS print routine

        MOV AH, 4CH                  ; 4CH = DOS exit routine
        INT 21H                      ; calling DOS exit routine
END start
```

```
C:\> tasm.exe hello_world.asm
C:\> tlink.exe hello_world.obj
C:\> hello_world.exe
HELLO WORLD
C:\>
```

# Instruction INT 21h

- Interruptions
  - ➔ Causes code to jump to location stored in interrupt table
  - ➔ Triggered by HW, proc (/zero), or special op (INT)
  - ➔ Proc state saved on INT, restored on return (IRET)
- In 8086 / 8186 : INT 21h = "paleo-software"
  - ➔ Prog run in "real mode" = direct access to CPU
  - ➔ INT 21h : access to **DOS routines** (routine # in AH)



- ≥ 8286 : privilege levels
  - ➔ Normal prog = run in **protected mode** (= user mode)
  - ➔ INT: intercepted by OS → can be used for syscalls
  - ➔ On x86-64 : SYSENTER & SYSCALL (faster)

https://fr.wikipedia.org/wiki/Limulidae