CSC231 - Assembly

Week #3

Dominique Thiébaut
dthiebaut@smith.edu
Questions

line1 db "ABCDEFG",10
line2 db "HIJKLMN",10
line3 db "OPQRSTU",10
line4 db "VWXYZ__",10
msgLen equ $-line1

1. What is the difference between this above code versus just 1 variable and db everything? Is there a difference at all?

2. is msgLen always a necessary variable as a kind of “where you are”? like a pointer? do i always need to consider it when writing new program?

3. what is the meaning of equ and $-msg?

4. every single time you change something in the .asm, must you nasm and ld? if i don't do that, then what happens?

5. 10 which stands for next line is only 1 character in ecx, why?
Nerd Alert!
Quick Introduction To Hexadecimal

• A system for representing numbers
• Base 16
• 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
• **Binary** $\longleftrightarrow$ **Hexadecimal**: natural conversion
<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>
Python is a Good Tool!
Hexadecimal, Listings, & Opcodes
emacs hello.asm

nasm -f elf hello.asm -l hello.lst

(minus ell)
Hello           db      "Hello there!", 10, 10
HelloLen        equ     $-Hello

section .text
global  _start

_start:

;;; print message
    mov     eax, 4          ; write
    mov     ebx, 1          ; stdout
    mov     ecx, Hello      ; address of message to print
    mov     edx, HelloLen   ; # of chars to print
    int     0x80

;;; exit
    mov     ebx, 0
    mov     eax, 1
    int     0x80
section .data
Hello db "Hello there!", 10, 10
HelloLen equ $-Hello

section .text
global _start
_start:
;;; print message
mov eax, 4 ; write
mov ebx, 1 ; stdout
mov ecx, Hello ;
mov edx, HelloLen ;
int 0x80

;;; exit
mov ebx, 0
mov eax, 1
int 0x80
section .data
Hello db "Hello there!", 10, 10
HelloLen equ $-Hello
section .text
global _start
_start:

;;; print message
mov eax, 4 ; write
mov ebx, 1 ; stdout
mov ecx, Hello ;
mov edx, HelloLen ;
int 0x80

;;; exit
mov ebx, 0
mov eax, 1
int 0x80
We stopped here last time...
Two Tips for Assembly Language Programmers

- Seeing invisible characters
- Segmentation faults
Seeing Invisible Characters

```asm
section .data
message db "Hello there!", 10
fib db 1,1,2,3,5,8,13,21

section .text
global _start

_start:
mov eax, 4
mov ebx, 1
mov ecx, message
mov edx, 15 ; this is wrong. It should
            ; be 13, but I purposely made
            ; a mistake

int 0x80

-UU-:***-F1 printWeird.asm 57% L36 (Assembler) ---------------
```

D. Thiebaut, Computer Science, Smith College
Seeing Invisible Characters

```bash
$ nasm -f elf printWeird.asm
$ ld -melf_i386 printWeird.o -o printWeird
$ ./printWeird
Hello there!
```

D. Thiebaut, Computer Science, Smith College
Seeing Invisible Characters

```
cs231a@marax:~/handout$ nasm -f elf printWeird.asm
/cs231a@marax:~/handout$ ld -melf_i386 printWeird.o -o printWeird
/cs231a@marax:~/handout$ ./printWeird
Hello there!
```

```
cs231a@marax:~/handout$
/cs231a@marax:~/handout$ ./printWeird | cat -v
Hello there!
```

```
^A^Acs231a@marax:~/handout$
```
Hello there!

Hello there!

^A^Ac
Two Tips for Assembly Language Programmers

• Seeing invisible characters

• Segmentation faults
```assembly
section .data
message db "Hello there!", 10

;;; code area

section .text
global _start

_start:
    mov    eax, 4
    mov    ebx, 1
    mov    ecx, message
    mov    edx, 1300 ; this is wrong. It should be 13, but I purposely made a mistake

    int    0x80

;;; exit() -F1 segFault.asm 53% L33 (Assembler) -----------------------------------
```
D. Thiebaut, Computer Science, Smith College

```
$ nasm -f elf segFault.asm
$ ld -melf_i386 segFault.o -o segFault
$ ./segFault
Hello there!
```

```
$ segFault.asm:1:1: severe error: ‘segFault’ undeclared (first use in this function)
    .text
$ segFault.asm:2:1: error: ‘asmmessage’ undeclared (first use in this function)
```

```
$ D. cs231a@marax:~$ handout$
```
Hexdump
Visualizing Bytes in RAM (Almost!)

emacs hello.asm

`nasm -f elf hello.asm`  
`ld -melf_i386 hello. -o hello`  
`.\hello`

`nasm -f elf hello.asm`  
`ld -melf_i386 hello. -o hello`  
`hexdump -v -C hello`
Hexdump

```
$ hexdump -v -C hello
00000000 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 |.ELF........|
00000010 02 00 03 00 01 00 00 00 00 00 00 00 34 00 20 0 |........4..|
00000020 b0 01 00 00 00 00 00 00 00 34 00 20 00 02 00 |0 4 2 8 0 0|
00000030 06 00 03 00 01 00 00 00 00 00 00 80 04 08 0 |0 0 0 0 0 0|
00000040 00 80 04 08 a2 00 00 00 a2 00 00 00 05 00 0 |0 0 0 0 5 0|
00000050 00 10 00 00 01 00 00 00 00 00 00 00 00 04 0 |0 0 0 0 0 0|
00000060 a4 90 04 08 0e 00 00 00 00 00 00 00 00 00 0 |9 0 0 0 0 0|
00000070 00 10 00 00 01 00 00 00 00 00 00 00 00 00 0 |0 0 0 0 0 0|
00000080 b8 04 00 00 00 00 bb 01 00 00 00 b9 a4 90 04 |8 0 4 a 0 0|
00000890 0e 00 00 00 cd 80 bb 00 00 00 00 b8 01 00 0 |0 0 0 0 0 0|
000009a0 cd 80 00 00 48 65 6c 6c 6f 20 74 68 65 72 |Hello there!|
00000ab0 0a 0a 00 00 00 00 00 00 00 00 00 00 00 00 0 |
00000c0 00 00 00 00 00 00 00 00 00 80 00 04 08 00 0 |
00000d0 03 00 01 00 00 00 00 00 00 00 00 00 00 00 0 |
00000e0 03 00 02 00 01 00 00 00 00 00 00 00 00 00 0 |
00000f0 04 00 ff 0b 00 00 00 00 00 00 00 00 00 00 0 |
0000100 00 00 02 00 11 00 00 00 00 00 00 00 00 00 0 |
0000110 00 00 ff 1f 00 00 00 00 00 00 00 00 00 00 0 |
0000120 10 00 01 00 1a 00 00 00 00 00 00 00 00 00 0 |
0000130 10 00 02 00 26 00 00 00 00 00 00 00 00 00 0 |
0000140 10 00 02 00 2d 00 00 00 00 00 00 00 00 00 0 |
0000150 10 00 02 00 00 68 65 6c 6f 00 00 00 00 00 4 |.hello.asm.H|
0000160 06 6c 6c 6f 00 48 65 6c 6f 00 00 00 00 00 5 |ello.HelloLen,|
0000170 0c 62 73 73 5f 73 74 61 72 74 00 5f 65 64 6 |bss_start.edata|
0000180 61 00 00 00 00 00 00 00 00 00 00 00 00 00 0 |.end..symtab.|
0000190 73 74 72 74 61 62 00 2e 73 68 73 74 72 61 6 |strtab..shstrtab|
00001a0 00 2e 74 65 64 61 74 61 00 00 00 00 00 00 0 |.text..data...|
```

D. Thiebaut, Computer Science, Smith College
section .data
Hello db "Hello there!", 10, 10
HelloLen equ $-Hello

section .text
global _start

_start:
;;; print message
mov eax, 4 ; write
mov ebx, 1 ; stdout
mov ecx, Hello ;
mov edx, HelloLen ;
int 0x80

;;; exit
mov ebx, 0
mov eax, 1
int 0x80
Our Goal for This Week

```c
int x, y, sum;

x = 3;
y = 5;
sum = x + y;
```
• **mov** & **add** instructions

• **Registers**

• Memory storage **directives**
You already know some of this material...

- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
The *mov* instruction

`mov dest, source`
The *mov* instruction

\[ \text{mov} \text{ dest, source} \]
The **`mov`** instruction

\[ \text{mov dest, source} \]
Pentium Registers

eax
ebx
ecx
edx
Operands

• `mov` reg, reg
• `mov` reg, mem
• `mov` mem, reg
• `mov` reg, imm
• `mov` mem, imm
section .data
    a      dd      1234

section .text
    mov     eax, 34
    mov     ebx, 12345
    mov     edx, eax
    mov     ecx, ebx

    eax
    ebx
    ecx
    edx
section .data
a      dd      1234

section .text
mov     eax, dword[a]
mov     ebx, eax
mov     eax, 1234
mov     dword[a], eax

eax
ebx
cdx
dx
section .data
hello  db  "Hi!"
helloL equ $-hello
a dd  1234

section .text
mov    eax, 123456789
mov    dword[a], eax
mov    dword[a], 0
mov    ecx, dword[a]
The add instruction

add dest, source
The add instruction

\[
dest = dest + source
\]

\[
\text{add } dest, source
\]
section .data
a dd 1234

section .text
mov eax, 3
mov ebx, 5
add eax, ebx
section .data
a dd 1234

section .text
mov eax, dword[a]
add eax, 1
mov dword[a], eax
section .data
a    dd    1234

section .text
add    dword[a], 1

eax
ebx
ecx
edx
Exercise

```c
int x, y, sum;
x = 3;
y = 5;
sum = x + y;
```

Translate this into Assembly
We stopped here last time...