CSC231 - Assembly

Week #3

Dominique Thiébaut
dthiebaut@smith.edu
mov
add

memory

registers

hexdump
listings
number systems
Let's Review Last Week's Material...
section .data
Hello    db "Hello there!", 10, 10
HelloLen equ $-Hello

section .text
_global  _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
Hexdump
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
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>40</td>
<td>100</td>
<td>#64;</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>101</td>
<td>#65;</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>102</td>
<td>#66;</td>
<td>B</td>
</tr>
<tr>
<td>67</td>
<td>43</td>
<td>103</td>
<td>#67;</td>
<td>C</td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>104</td>
<td>#68;</td>
<td>D</td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td>105</td>
<td>#69;</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>46</td>
<td>106</td>
<td>#70;</td>
<td>F</td>
</tr>
<tr>
<td>71</td>
<td>47</td>
<td>107</td>
<td>#71;</td>
<td>G</td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>110</td>
<td>#72;</td>
<td>H</td>
</tr>
<tr>
<td>73</td>
<td>49</td>
<td>111</td>
<td>#73;</td>
<td>I</td>
</tr>
</tbody>
</table>

RAM

'H'
Processor-RAM Connection

Pentium

RAM
Our Goal for This Week

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

• Mov & add instructions
• Registers
• Memory storage options
The mov instruction

mov dest, source
The mov instruction

```
section .data
Hello db "Hello there!", 10, 10
HelloLen equ $-Hello

section .text
global _start
_start:

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

;;; exit
mov ebx, 0
mov eax, 1
int 0x80
```

mov dest, source
Operands

- `mov` reg, reg
- `mov` reg, mem
- `mov` mem, reg
- `mov` reg, imm
- `mov` mem, imm
Pentium Registers

eax
ebx
ecx
edx
section .data
a     dd    1234

section .text
mov   eax, 34
mov   ebx, 12345
mov   edx, eax
mov   ecx, ebx
mov   edx, eax
mov   ebx, 12345
mov   ecx, eax
mov   edx, ebx
section .data
    a   dd    1234

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

eax  
ebx  
ecx  
edx  

We stopped here last time…
• bit
• nybble
• byte
• double-word
• Hexadecimal:

0000 0
0001 1
0010 2
0011 3
0100 4
0101 5
0110 6
0111 7
1000 8
1001 9
1010 A
1011 B
1100 C
1101 D
1110 E
1111 F
<table>
<thead>
<tr>
<th></th>
<th>2&lt;sup&gt;0&lt;/sup&gt;</th>
<th>2&lt;sup&gt;1&lt;/sup&gt;</th>
<th>2&lt;sup&gt;2&lt;/sup&gt;</th>
<th>2&lt;sup&gt;3&lt;/sup&gt;</th>
<th>2&lt;sup&gt;4&lt;/sup&gt;</th>
<th>2&lt;sup&gt;5&lt;/sup&gt;</th>
<th>2&lt;sup&gt;6&lt;/sup&gt;</th>
<th>2&lt;sup&gt;7&lt;/sup&gt;</th>
<th>2&lt;sup&gt;8&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000 0000</td>
<td>0000 0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2&lt;sup&gt;8&lt;/sup&gt; = 256</td>
</tr>
<tr>
<td>1</td>
<td>1111 1110</td>
<td>1111 1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ 2^{32} = 4,294,967,296 \]
Exercise

section .data
hello  msg     "Hi!"
helloL equ     $-hello
a      dd      1234

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

 eax
 ebx
 ecx
 edx
The add instruction

add  dest, source
section .data
    a      dd      1234

section .text
    mov     eax, 3
    mov     ebx, 5
    add     eax, ebx

Exercise

eax
ebx
ecx
edx
section .data
    a    dd    1234

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

Exercise

eax
ebx
ecx
edx
section .data
da dd 1234

section .text
add dword[a], 1
mov eax, dword[a]
int x = 3;
int y = 5;
int sum;
sum = x + y;

Translate this code into its assembly equivalent.
Translate this code into its assembly equivalent:

```c
int x = 3;
int y = 5;
int z = 10;
int sum;
sum = x + y + 2*z;
```
Exercise

Translate this code into its assembly equivalent:

```c
int x = 3;
int y = 5;
int z = 10;
int sum;

sum = x + 2*y - 3*(z-1);
```
Getting a sense of Speed of Execution

LATEST 7TH GEN INTEL CORE i7 PROCESSOR
The new 7th Gen Intel Core i7-7700HQ processor gives the 14-inch Razer Blade
2.8GHz of quad-core processing power and Turbo Boost speeds, which
automatically increases the speed of active cores – up to 3.8GHz. Work, play
and create with ease and enjoy smooth, high definition 4K content like never
before. With the Razer Blade's thin and light design, you'd never guess it holds all
that power. Only with Intel Inside®.

2.8GHz of quad-core processing power and Turbo Boost speeds, which
automatically increases the speed of active cores – up to 3.8GHz. Work, play
Getting a sense of Speed of Execution

2.8 GHz CPU
2.8 billion cycles/second
1 instruction / cycle
cycle = \frac{1}{2.8 \times 10^9} = 0.35 \text{ ns}

10 instructions $\Rightarrow$ 3.5 ns

can compute 280 million similar equations in 1 second

; sum = x + 2\times y - 3\times (z-1)
mov eax, dword[x]
add eax, dword[y]
add eax, dword[y]
mov ebx, dword[z]
sub ebx, 1
mov ecx, ebx
add ebx, ebx
add ebx, ecx
add eax, ebx
mov dword[sum], eax
How are Integers Stored in Memory?
msg1 db "Hello"
120
11F
11E
11D
11C
11B
11A
119
118
117
116
115
114
113
112
111

\begin{tabular}{ccc}
\textbf{a} & \textbf{dd} & 0x00000003 \\
b & \textbf{dd} & 0x12345678 \\
\end{tabular}
Have we seen this before?
Endianness

- **Little-Endian** Processors
  - Intel Pentium
  - Apple's original MOS 6502
  - Zilog Z80

- **Big-Endian** Processors
  - Motorola 68000
  - Atmel AVR32 (Arduino)
Current generation ARM processors (from ARM6 onwards) have the option of operating in either little-endian or big-endian mode. These terms refer to the way in which multi-byte quantities, such as 32-bit words, are stored in a byte-addressed memory.

http://netwinder.osuosl.org/pub/netwinder/docs/arm/Apps04vC.html
Implications

• Serialization of data

http://ubjson.org/#endian
Pentium Data Registers