Let's Review Last Week's Material...
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      ; address of message to print
        mov     edx, HelloLen   ; # of chars to print
        int     0x80

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

Listing

231b@aurora ~/handout $ nasm -f elf hello.asm -l hello.lst
231b@aurora ~/handout $
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

```
231b@aurora ~/handout $ hexdump -v -C hello
00000000 7f 45 4c 46 01 01 00 00 00 00 00 00 00 00 .ELF         |
00000010 02 00 03 00 01 00 00 00 80 80 04 08 34 00 00 00 ................4.
00000020 dc 00 00 00 00 00 00 00 00 34 00 20 00 02 00 28 00 ................4.
00000030 06 00 03 00 01 00 00 00 00 00 00 00 80 04 08 00 hello                  
00000040 00 80 04 08 a2 00 00 00 a2 00 00 00 05 00 00 00 ......................
00000050 00 10 00 00 01 00 00 00 a4 00 00 00 a4 90 04 08 ......................
00000060 a4 90 04 08 0c 00 00 00 0c 00 00 00 06 00 00 00 ......................
00000070 00 10 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000080 b8 04 00 00 00 bb 01 00 00 00 b9 a4 90 04 08 ba ......................
00000090 0c 00 00 00 cd 80 bb 00 00 00 00 00 b8 01 00 00 00 ......................
000000a0 cd 80 00 00 48 65 6c 6c 6f 77 00 00 00 00 00 00 00 00 00 ...Hello there!
000000b0 0a 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
000000c0 61 62 00 00 73 79 6d 74 74 00 00 00 00 00 00 00 00 00 00 00 ......................
000000d0 78 74 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
000000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000110 80 80 04 08 80 00 00 00 22 00 00 00 00 00 00 00 00 00 00 00 ......................
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000130 01 00 00 00 00 00 00 00 a4 90 04 08 a4 00 00 00 ......................
00000140 0e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000160 00 00 00 00 00 00 b2 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000180 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000190 b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
000001a0 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
000001b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
000001c0 39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
```

D. Thiebaut, Computer Science, Smith College
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
```
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

Why Hexadecimal?
<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>Ø</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>

The diagram illustrates the memory mapping of the characters 'H' in RAM.
Processor-RAM Connection

Pentium

RAM
int x, y, sum;
x = 3;
y = 5;
sum = x + y;

Our Goal for
This Week
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, 1 ; stdout
mov   ecx, Hello ;
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
ccx
edx
section .data
    a    dd  1234

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

    eax
    ebx
    ecx
    edx
```assembly
section .data
a      dd      1234

section .text
mov     eax, dword[a]
mov     ebx, eax
mov     eax, 1234
mov     dword[a], eax
```
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
$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

$2^7 = 128$

$2^8 = 256$

...
\[ 2^{32} = 4,294,967,296 \]
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]

Exercise

eax
ebx
cecx
dedx
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
cx
edx
section .data
da      dd      1234

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

Exercise

eax
ebx
ecx
edx
section .data
    a dd 1234

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

Translate this code into its assembly equivalent

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

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 = 1/(2.8E+9) = 0.35 ns

10 instructions ==> 3.5 ns

can compute 280 million similar equations in 1 second

; sum = x + 2*y + 3*(z-1)
mov eax, dword[x]
add eax, dword[y]
add eax, dword[y]
mov ebx, dword[z]
add ebx, -1
mov ecx, ebx
add ebx, ebx
add ebx, ecx
add eax, ebx
mov dword[sum], eax
We stopped here last time…