How are Integers Stored in Memory?
<table>
<thead>
<tr>
<th>msg1</th>
<th>db</th>
<th>&quot;Hello&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Have we seen this before?

```
231b@aurora ~/handout $ 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 80 80 04 08 34 00 00 00 |
00000020 dc 00 00 00 00 00 00 00 34 00 20 00 02 00 28 00 |
00000030 06 00 03 00 01 00 00 00 00 00 00 00 80 04 08 |
00000040 00 80 04 08 a2 00 00 00 00 00 00 00 00 00 00 00 |
00000050 00 10 00 00 01 00 00 00 00 00 00 00 a4 00 00 00 |
00000060 a4 90 04 08 0e 00 00 00 0e 00 00 00 00 00 00 00 |
00000070 00 10 00 00 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 0e 00 00 00 cd 80 bb 00 00 00 00 b8 01 00 00 00 |
000000a0 cd 80 00 00 48 65 6c 6c 6f 20 74 68 65 6c 65 |
000000b0 0a 0a 00 2e 73 79 74 69 66 72 0e 73 79 74 72 74 |
000000c0 61 62 00 2e 73 79 74 73 74 72 74 61 62 00 2e 74 65 |
000000d0 78 74 00 2e 64 61 74 69 66 72 0e 73 79 74 72 74 |
000000e0 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 |
00000100 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 |
00000120 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 |
00000130 01 00 00 00 00 03 00 00 00 00 00 00 00 00 00 00 |
00000140 0e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00000150 00 00 00 00 11 00 00 00 03 00 00 00 00 00 00 00 |
00000160 00 00 00 00 00 00 00 00 b2 00 00 00 00 00 00 00 |
00000170 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 |
00000190 b0 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 |
000001a0 10 00 00 00 00 09 00 00 03 00 00 00 00 00 00 00 |
000001b0 00 00 00 00 00 00 7c 02 00 00 39 00 00 00 00 00 |
...|...9...|
```
**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
Pentium Registers

eax
ebx
cex
edx
Pentium Registers

eax  ax
ebx  bx
ecx  cx
edx  dx
## Pentium Registers

<table>
<thead>
<tr>
<th>eax</th>
<th>ah</th>
<th>al</th>
<th>ax</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebx</td>
<td>bh</td>
<td>bl</td>
<td>bx</td>
</tr>
<tr>
<td>ecx</td>
<td>ch</td>
<td>cl</td>
<td>cx</td>
</tr>
<tr>
<td>edx</td>
<td>dh</td>
<td>dl</td>
<td>dx</td>
</tr>
</tbody>
</table>
Think of **ah** and **al** as boxes inside a bigger one called **ax**, and **ax** as half of a bigger box still, called **eax**.
Declaring Variables
• **db**: define byte storage
• **dw**: define word storage
• **dd**: define double-word storage
Examples: `db`

<table>
<thead>
<tr>
<th>msg</th>
<th>db</th>
<th>&quot;Hello&quot;, 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>db</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>db</td>
<td>'H' ; also 72 or 0x48</td>
</tr>
<tr>
<td>c</td>
<td>db</td>
<td>255</td>
</tr>
<tr>
<td>d</td>
<td>db</td>
<td>0x80</td>
</tr>
</tbody>
</table>
### Examples: dw

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>dw</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>dw</td>
<td>1</td>
</tr>
</tbody>
</table>
z  | dw | 255 |
t  | dw | 0x1234 |
<table>
<thead>
<tr>
<th></th>
<th>dd</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>dd</td>
<td>0</td>
</tr>
<tr>
<td>beta</td>
<td>dd</td>
<td>255</td>
</tr>
<tr>
<td>gamma</td>
<td>dd</td>
<td>$0x12345678$</td>
</tr>
</tbody>
</table>
We stopped here last time...
Announcement

Summary of important concepts just seen

- Numbers
- Endianness
- Op Codes
- Machine Language
- Hexadecimal
- Executable Files
Return to the mov instruction

mov dest, source
section .data
lf       db      10
ch       db      0
a        dw      0x1234
b        dw      0
x        dd      0
y        dd      0x12345678

section .text
; put lf in al
section .data
lf       db      10
ch       db      0
a        dw      0x1234
b        dw      0
x        dd      0
y        dd      0x12345678

section .text
; put al in ch

Test Cases

<table>
<thead>
<tr>
<th>eax</th>
<th>ebx</th>
<th>ecx</th>
<th>edx</th>
</tr>
</thead>
<tbody>
<tr>
<td>ah</td>
<td>bh</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>al</td>
<td>bl</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>ax</td>
<td>bx</td>
<td>cx</td>
<td>dx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ah</td>
<td>bh</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>al</td>
<td>bl</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>ax</td>
<td>bx</td>
<td>cx</td>
<td>dx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ah</td>
<td>bh</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>al</td>
<td>bl</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>ax</td>
<td>bx</td>
<td>cx</td>
<td>dx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ah</td>
<td>bh</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>al</td>
<td>bl</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>ax</td>
<td>bx</td>
<td>cx</td>
<td>dx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ah</td>
<td>bh</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>al</td>
<td>bl</td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ah</td>
<td>bh</td>
</tr>
<tr>
<td>ax</td>
<td>bx</td>
<td>cx</td>
<td>dx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
section .data
lf    db      10
ch    db      0
a     dw      0x1234
b     dw      0
x     dd      0
y     dd      0x12345678

section .text
; put a in bx
; put bx in b
; put bx in ax
; put 0 in cx

Test Cases

<table>
<thead>
<tr>
<th>eax</th>
<th>ah</th>
<th>al</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebx</td>
<td>bh</td>
<td>bl</td>
</tr>
<tr>
<td>ecx</td>
<td>ch</td>
<td>cl</td>
</tr>
<tr>
<td>edx</td>
<td>dh</td>
<td>dl</td>
</tr>
<tr>
<td>ax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
section .data
lf    db      10
ch    db      0
a     dw      0x1234
b     dw      0
x     dd      0
y     dd      0x12345678

section .text
; put x in eax
; put y in ecx
; put ecx in edx
; put ex into y

Test Cases

<table>
<thead>
<tr>
<th>eax</th>
<th>ah</th>
<th>al</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebx</td>
<td>bh</td>
<td>bl</td>
</tr>
<tr>
<td>ecx</td>
<td>ch</td>
<td>cl</td>
</tr>
<tr>
<td>edx</td>
<td>dh</td>
<td>dl</td>
</tr>
<tr>
<td></td>
<td>ax</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dx</td>
<td></td>
</tr>
</tbody>
</table>
section .data
lf db 10
ch db 0
a dw 0x1234
b dw 0
x dd 0
y dd 0x12345678

section .text
; put 0 in ah
; put 3 in cx
; put 5 in edx
; put 0x12345678 into eax
section .data
lf       db      10
ch       db      0
a        dw      0x1234
b        dw      0
x        dd      0
y        dd      0x12345678
section .text
mov     eax, dword[a]
mov     dword[b], eax
section .data
lf    db     10
ch    db     0
a     dw     0x1234
b     dw     0
x     dd     0
y     dd     0x12345678

section .text
mov    ax, word[a]
mov    word[lf], ax

Buggy Program #2
Strongly-typed languages

In computer programming, programming languages are often colloquially classified as strongly typed or weakly typed (loosely typed). These terms do not have a precise definition, but in general, a strongly typed language is more likely to generate an error or refuse to compile if the argument passed to a function does not closely match the expected type. On the other hand, a weakly typed language may produce unpredictable results or may perform implicit type conversion.

https://en.wikipedia.org/wiki/Strong_and_weak_typing
We understand mov!
The add instruction Revisited

add dest, source
The add instruction Revisited

add dest, source

reg8
reg16
reg32
mem8
mem16
mem32
imm8
imm16
imm32
The add instruction Revisited

add dest, source

reg8
reg16
reg32
mem8
mem16
mem32

reg8
reg16
reg32
mem8
mem16
mem32

imm8
imm16
imm32
The add instruction Revisited

Sizes have to match
section .data
lf db 10
ch db 0
a dw 0x1234
b dw 0
x dd 0
y dd 0x12345678

section .text
; add 3 to ch
; add 100 to b
; add -1 to edx
; add x to y
Reminder: Our Goal was...

We translated this into Assembly

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

Replace "int" by "short," and translate the program into assembly code.
We stopped here last time...