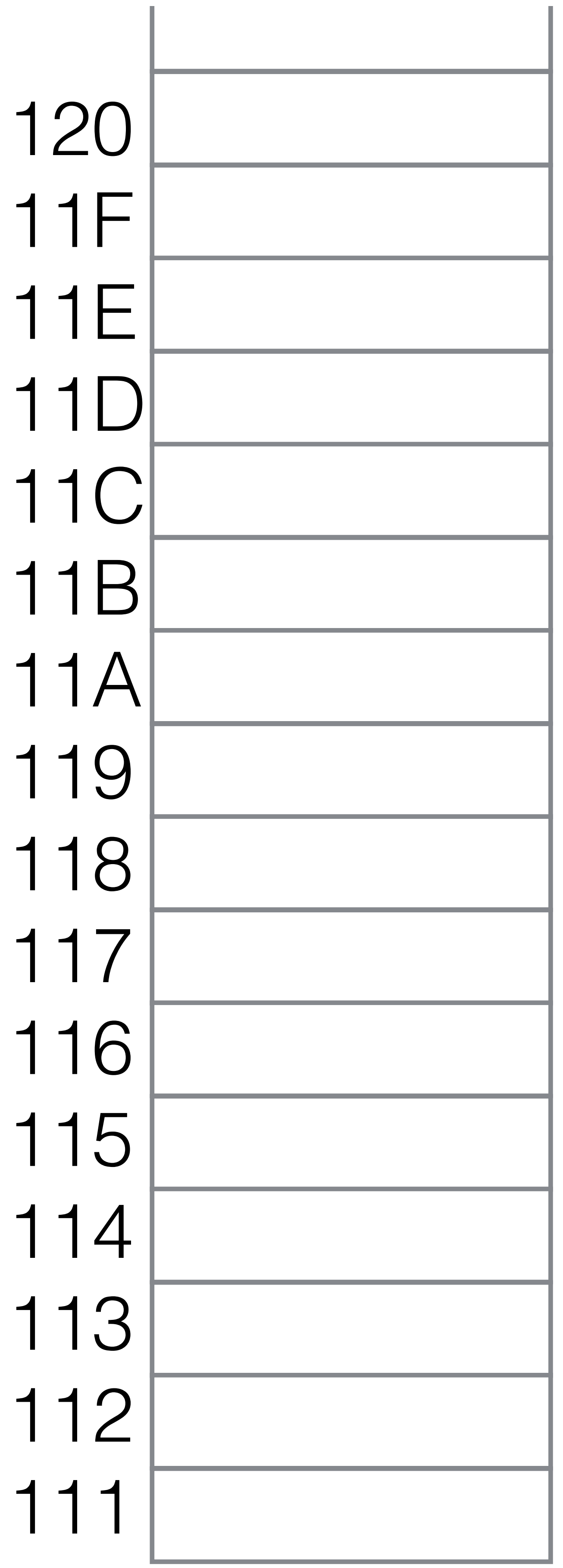
Smith College
Computer Science

CSC231 — Assembly

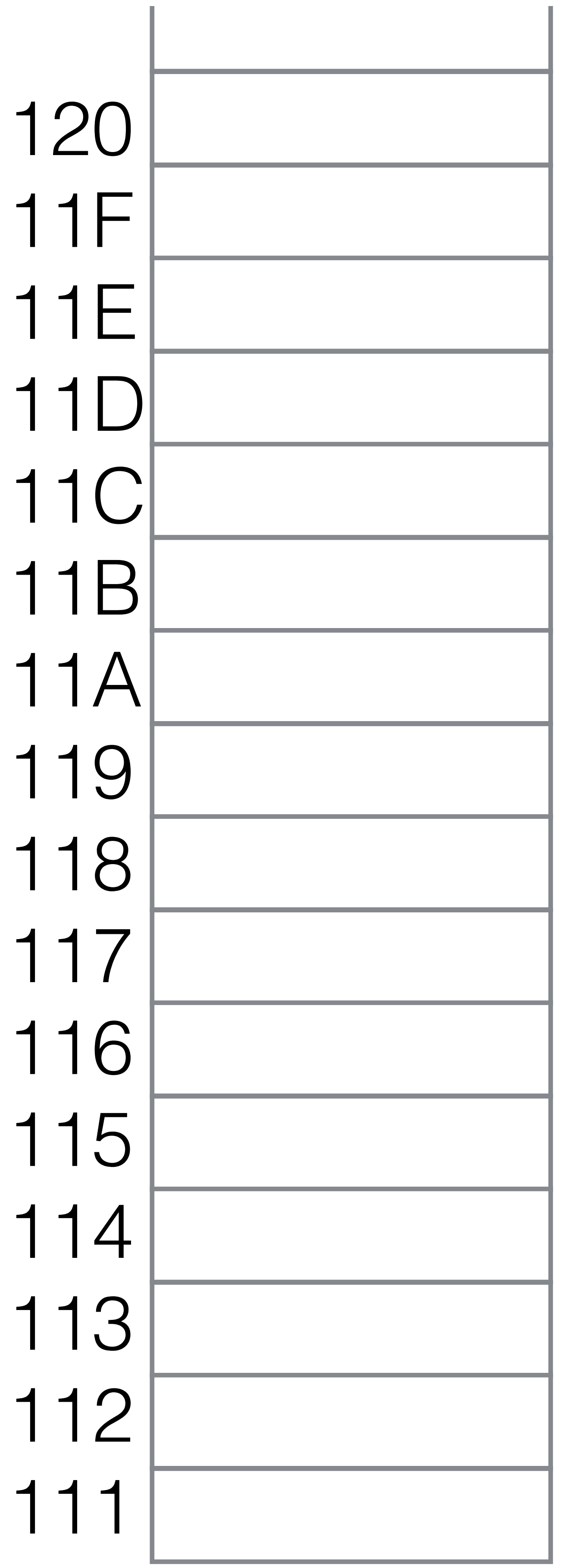
Fall 2017 — Week #4

Dominique Thiébaut
dthiebaut@smith.edu

How are Integers Stored in Memory?

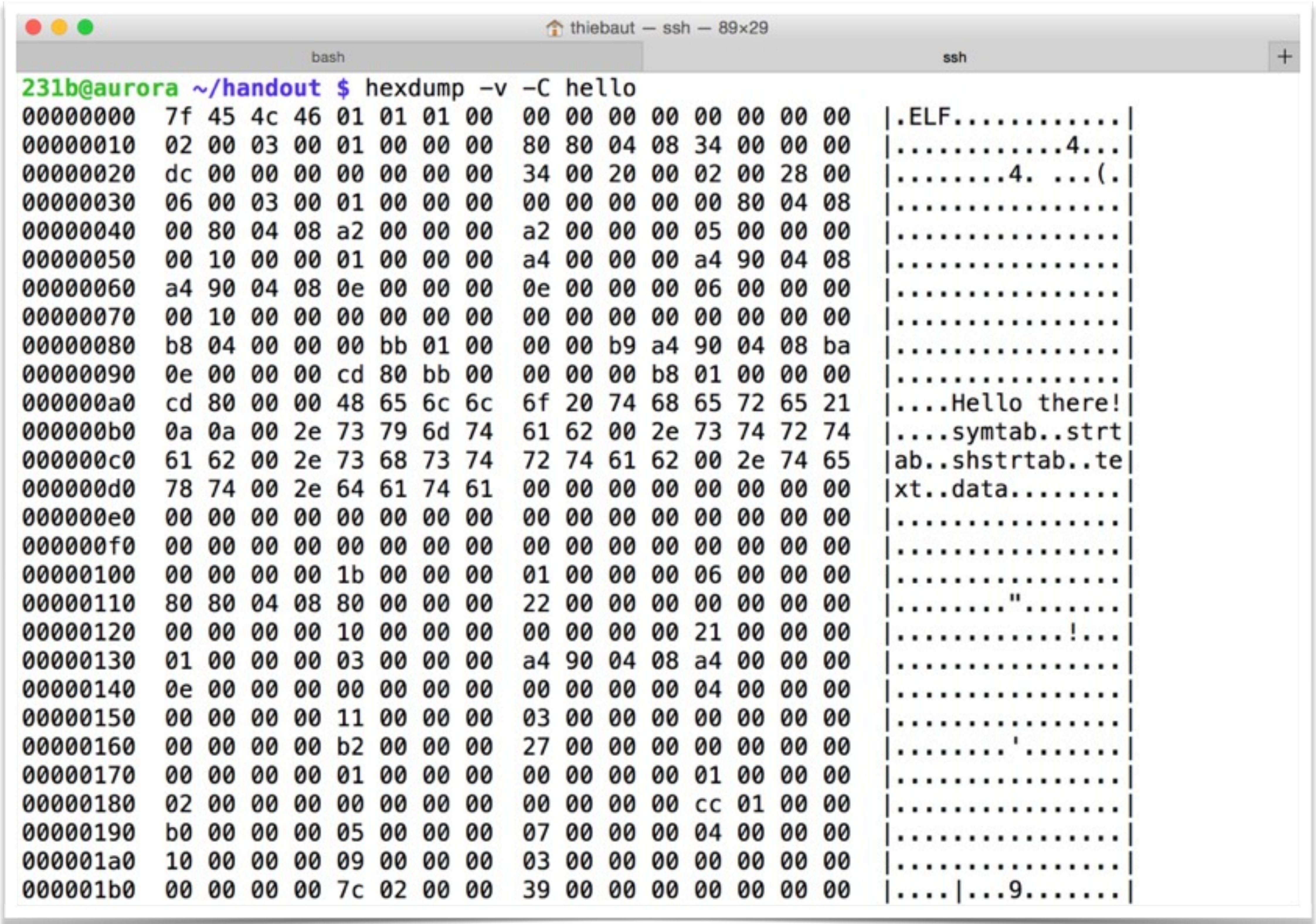


```
msg1    db        "Hello"
```



```
a           dd      0x00000003
b           dd      0x12345678
```

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 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  34 00 20 00 02 00 28 00  |........4. ...(.|
00000030  06 00 03 00 01 00 00 00  00 00 00 00 00 80 04 08  |................|
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 0e 00 00 00  0e 00 00 00 06 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 72 65 21  |....Hello there!|
000000b0  0a 0a 00 2e 73 79 6d 74  61 62 00 2e 73 74 72 74  |....symtab..strt|
000000c0  61 62 00 2e 73 68 73 74  72 74 61 62 00 2e 74 65  |ab..shstrtab..te|
000000d0  78 74 00 2e 64 61 74 61  00 00 00 00 00 00 00 00  |xt..data........|
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 1b 00 00 00  01 00 00 00 06 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 21 00 00 00  |............!...|
00000130  01 00 00 00 03 00 00 00  a4 90 04 08 a4 00 00 00  |................|
00000140  0e 00 00 00 00 00 00 00  00 00 00 00 04 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 b2 00 00 00  27 00 00 00 00 00 00 00  |........'.......|
00000170  00 00 00 00 01 00 00 00  00 00 00 00 01 00 00 00  |................|
00000180  02 00 00 00 00 00 00 00  00 00 00 00 cc 01 00 00  |................|
00000190  b0 00 00 00 05 00 00 00  07 00 00 00 04 00 00 00  |................|
000001a0  10 00 00 00 09 00 00 00  03 00 00 00 00 00 00 00  |................|
000001b0  00 00 00 00 7c 02 00 00  39 00 00 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)

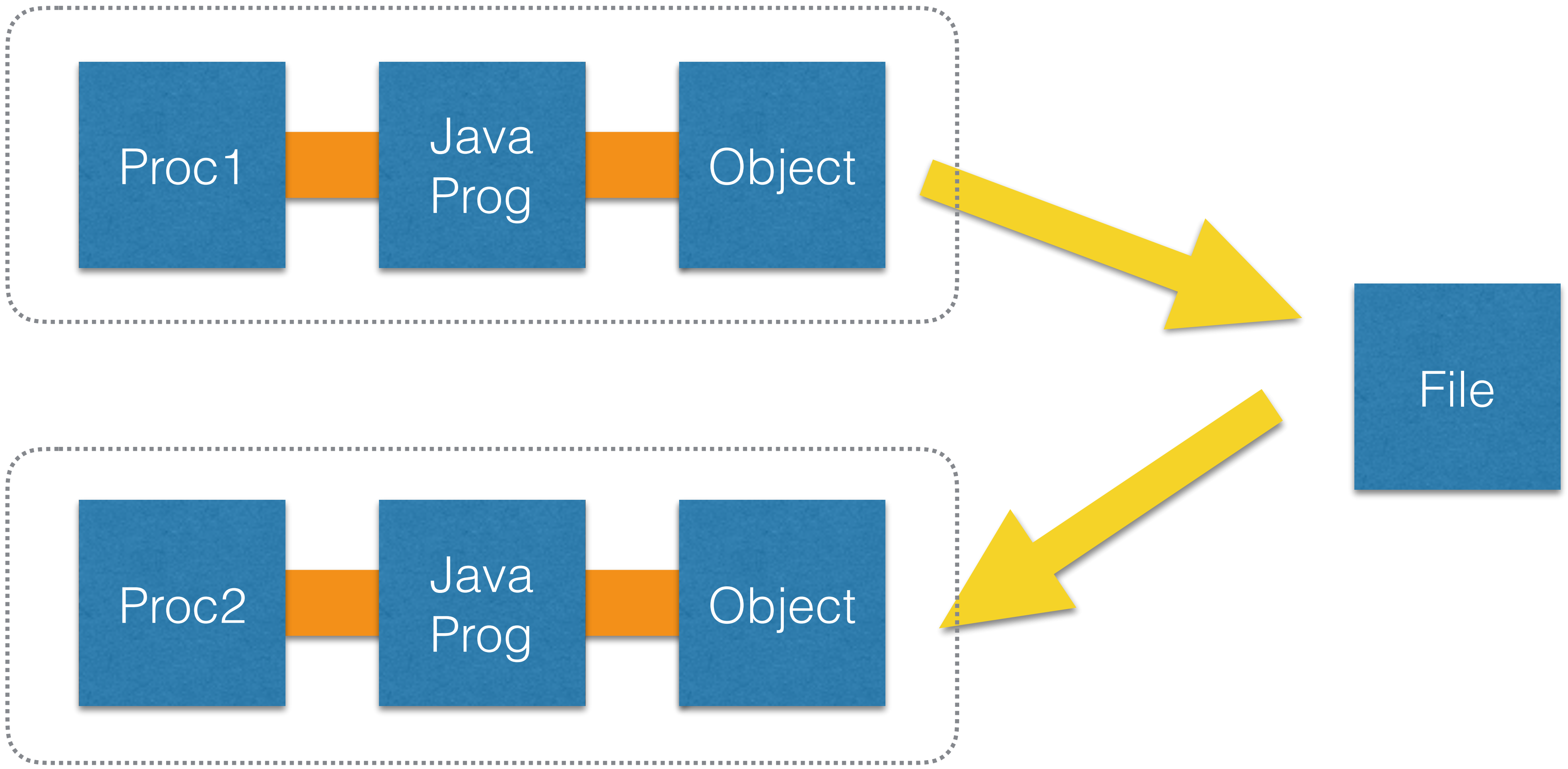
Endianness

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

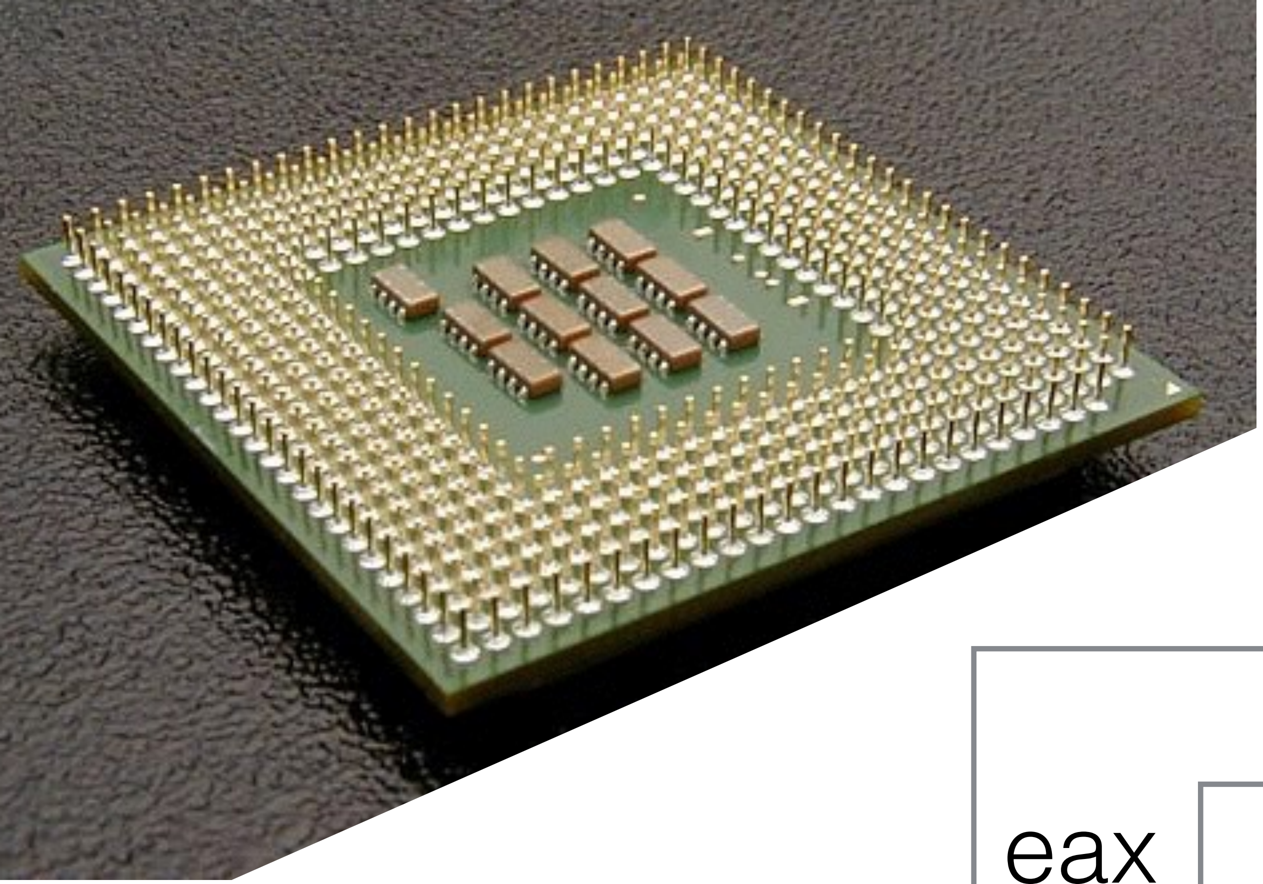
- Serialization of data

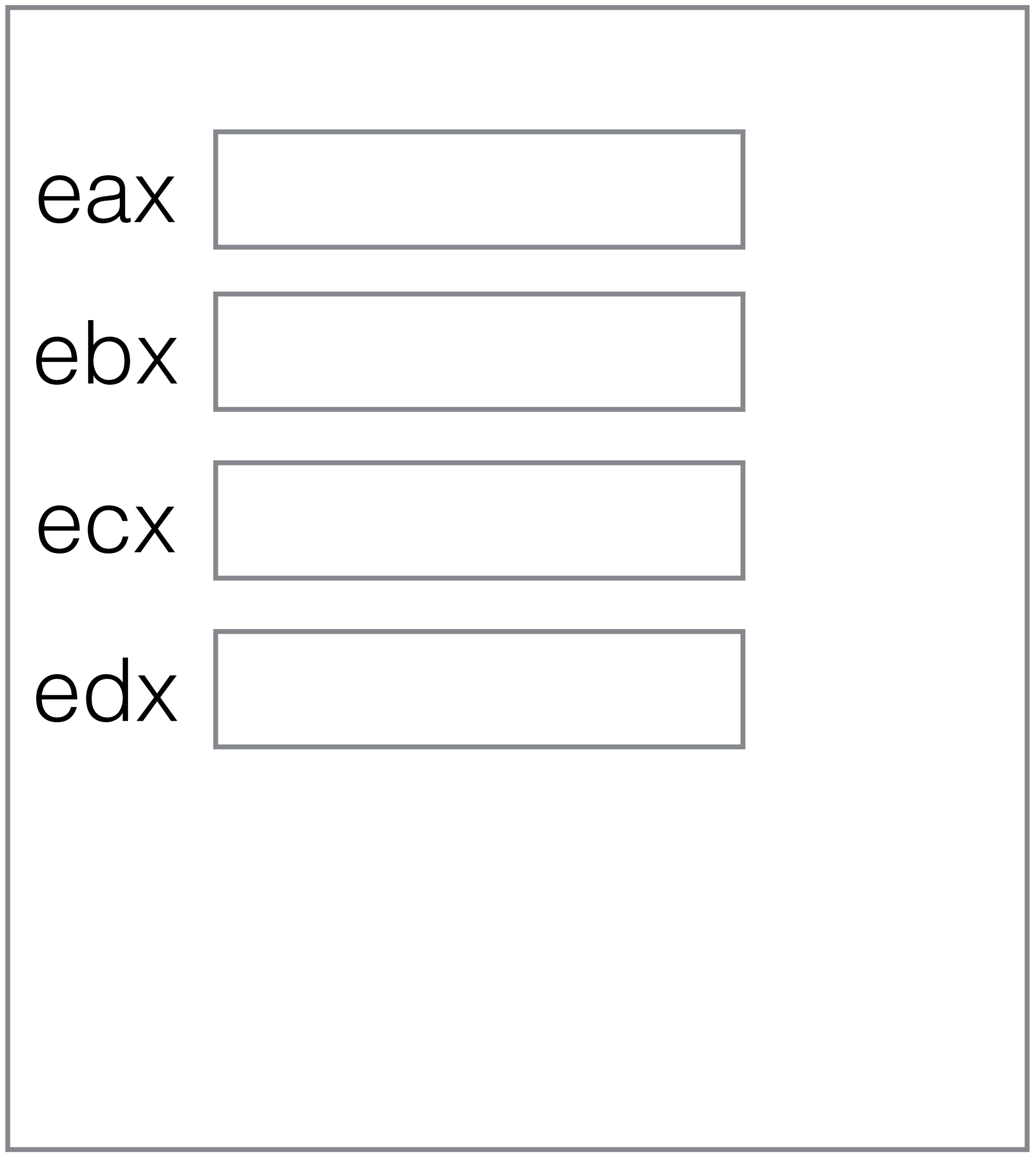


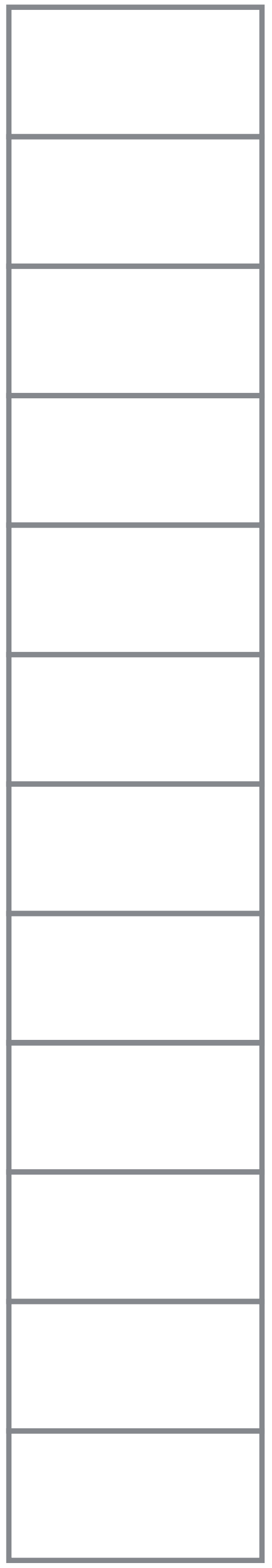
http://ubjson.org/#endian

Pentium Data Registers

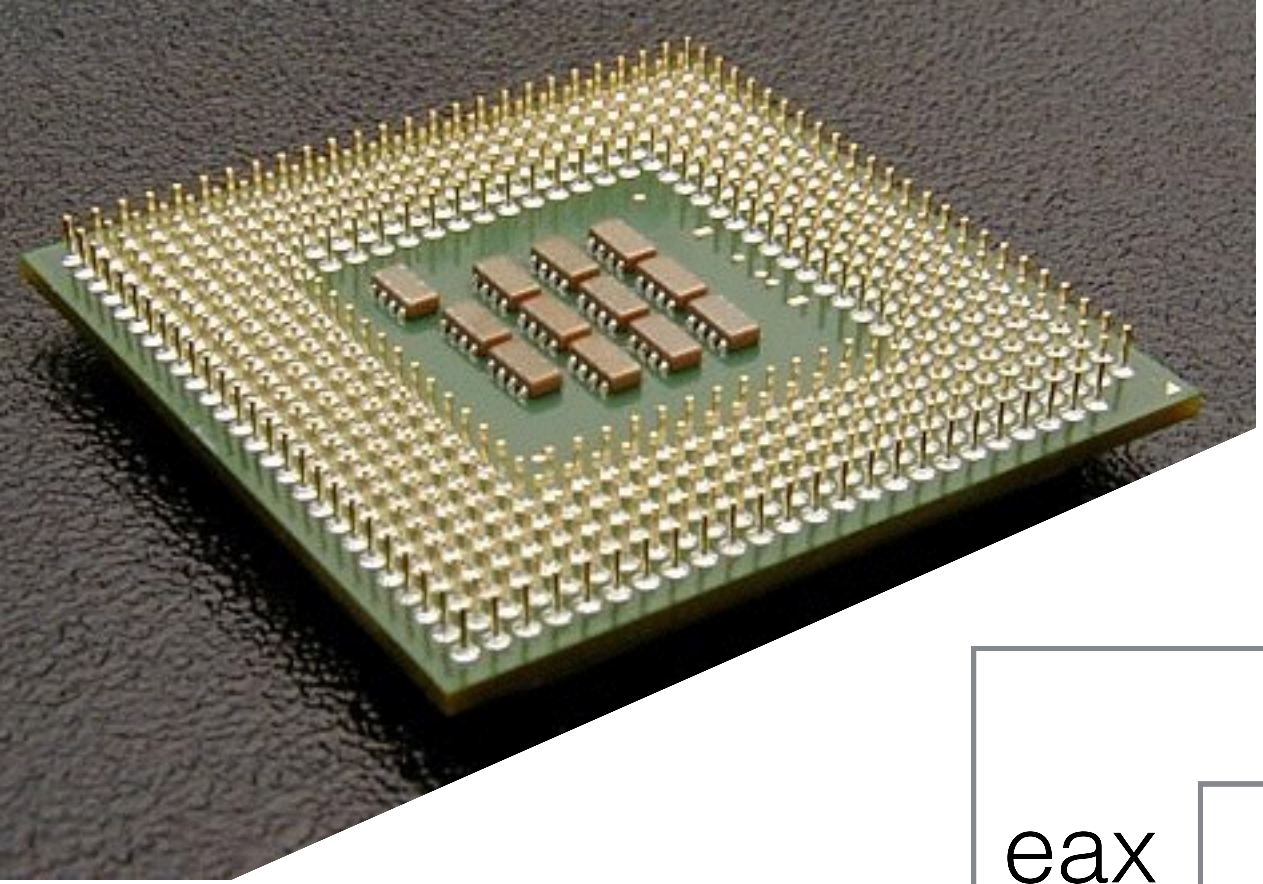
Pentium Registers

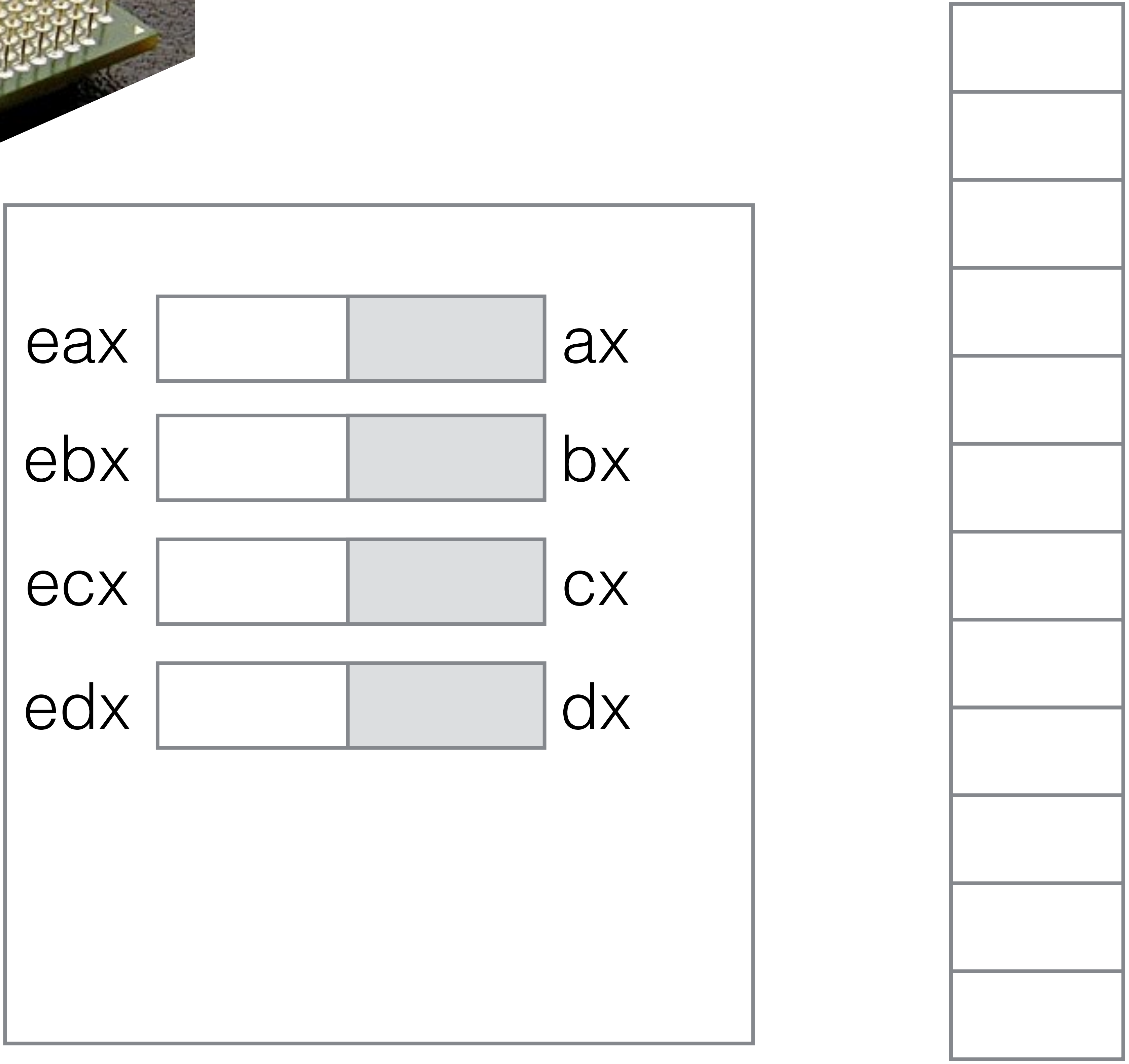




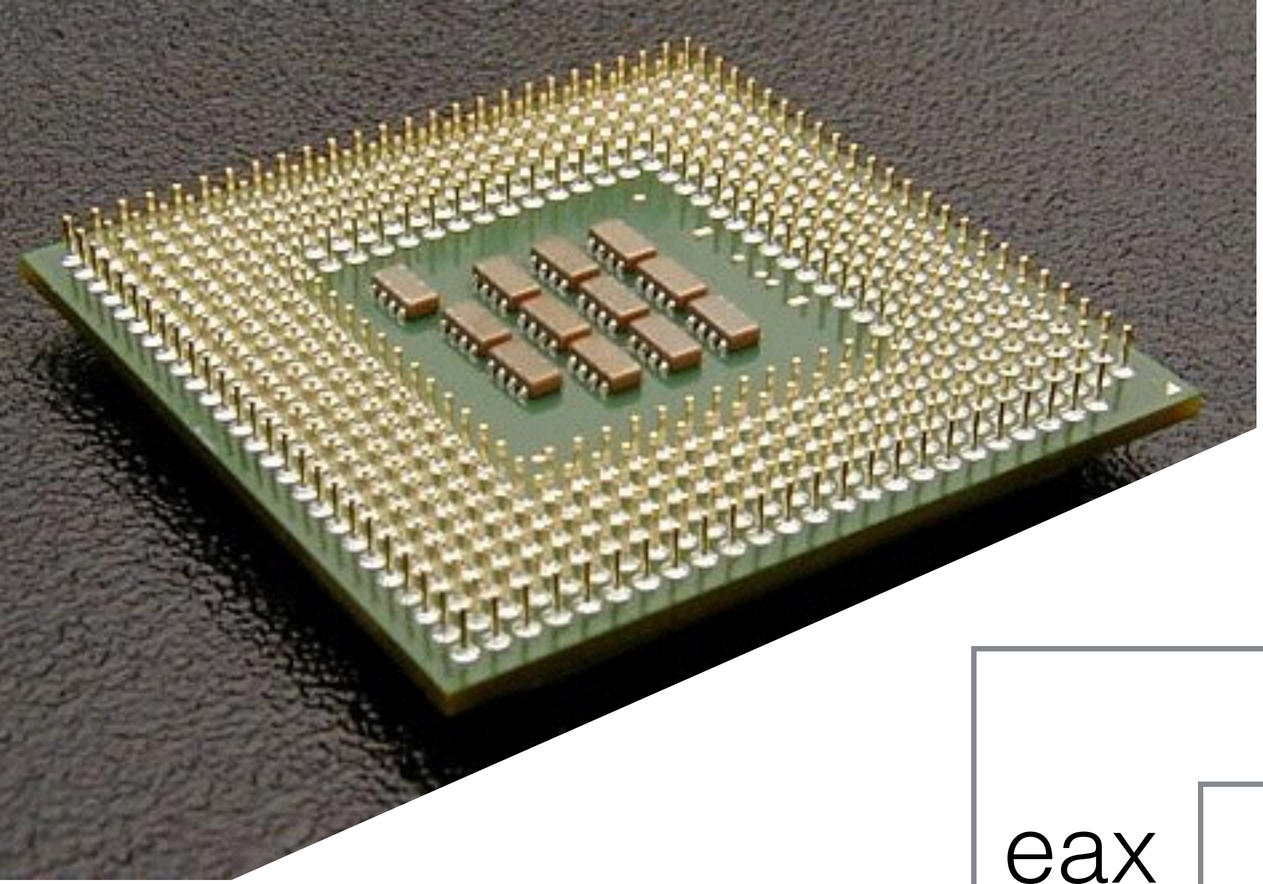


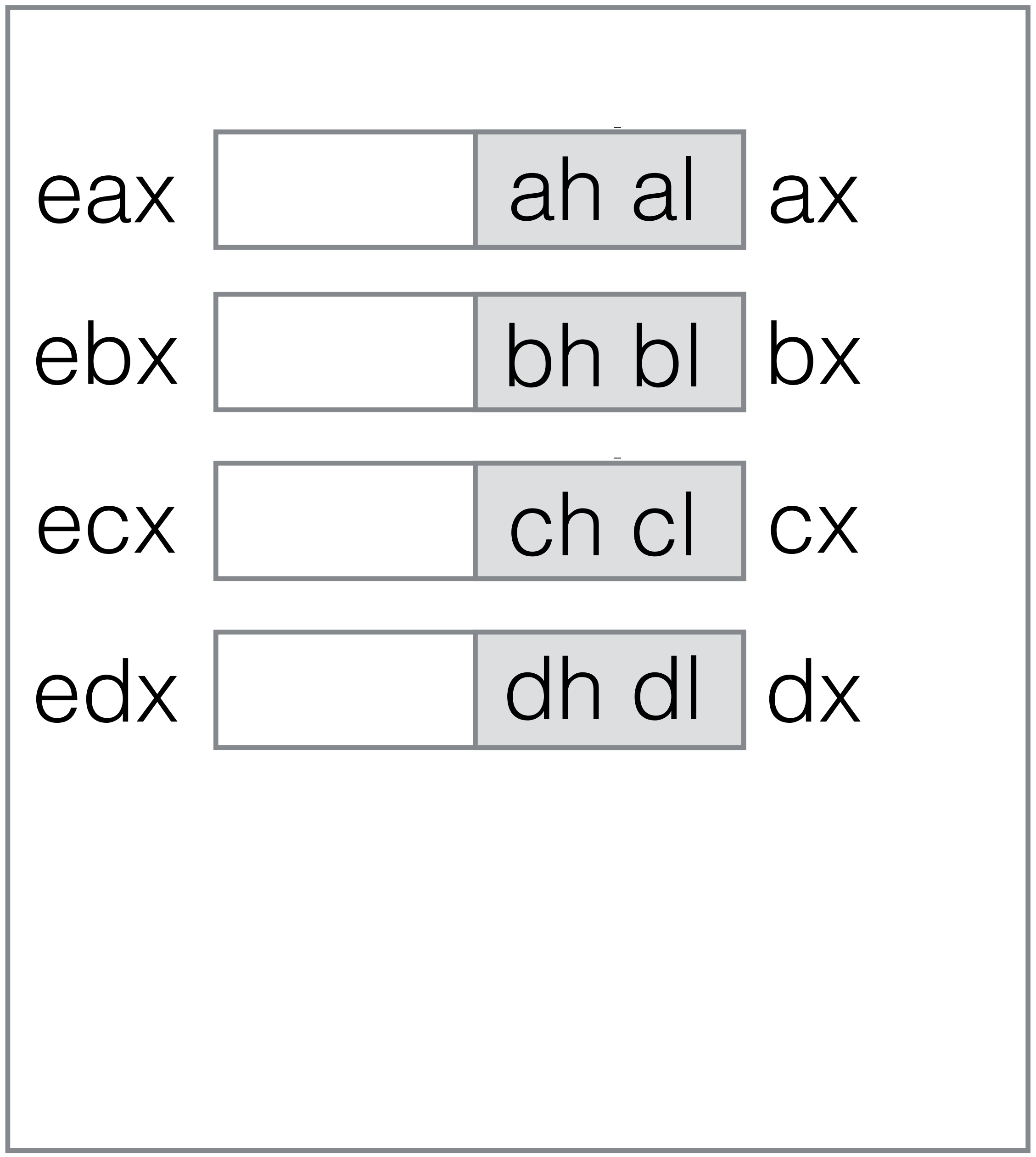
Pentium Registers

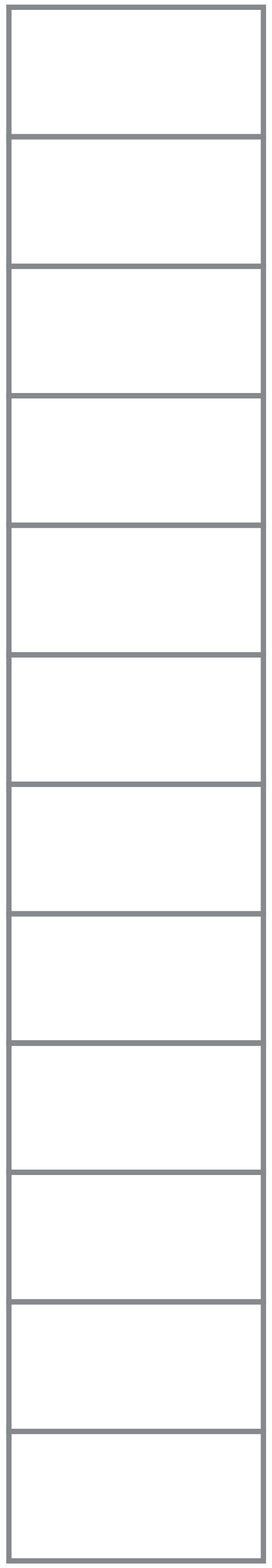




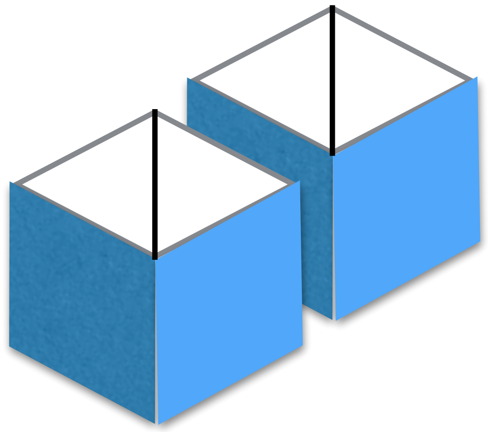
Pentium Registers







*Think of **ah** and **al** as boxes inside a bigger one called **ax**, and **ax** as half of a bigger box still, called **eax**.*



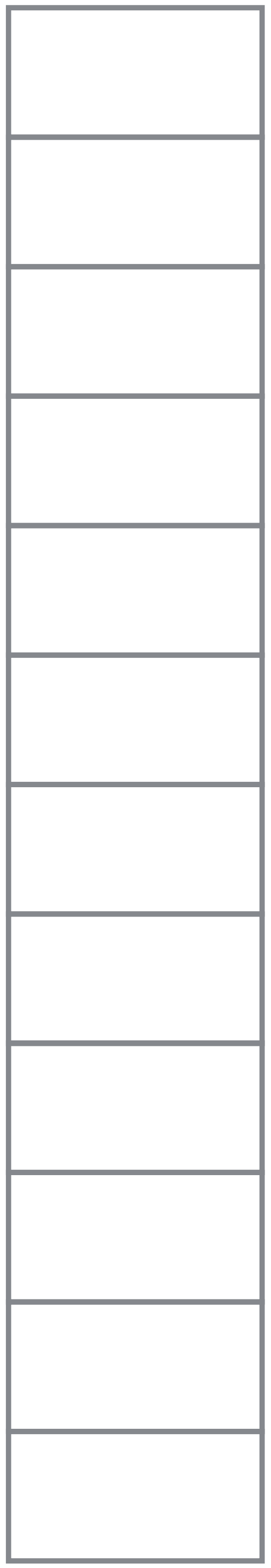
Declaring Variables

db, dw, dd

- **db**: **d**efine **b**yte storage
- **dw**: **d**efine **w**ord storage
- **dd**: **d**efine **d**ouble-word storage

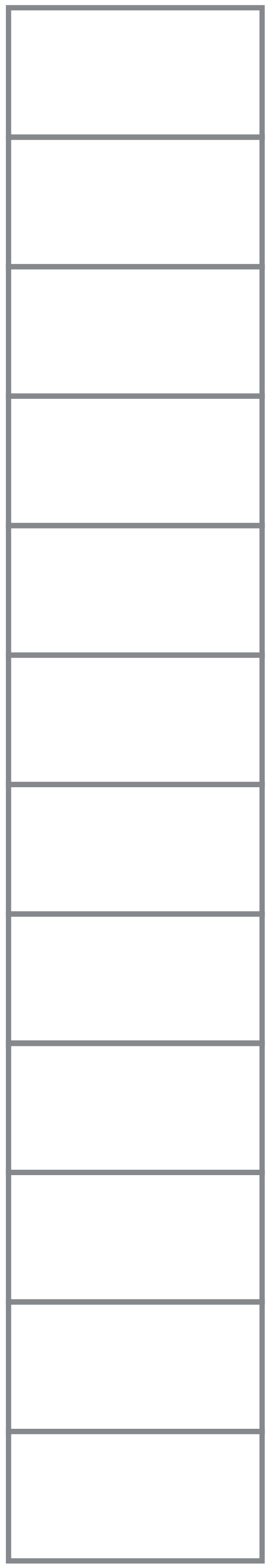
Examples: db

```
msg         db          "Hello", 10

a           db          0
b           db          'H'       ; also 72 or 0x48
c           db          255
d           db          0x80
```



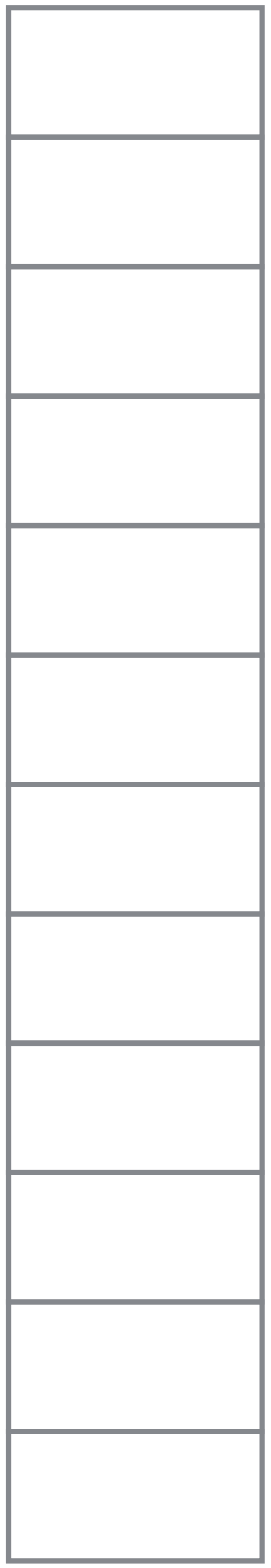
Examples: dw

```
x           dw          0
y           dw          1
z           dw          255
t           dw          0x1234
```



Examples: dd

```
alpha     dd        0
beta      dd        255
gamma     dd        0x12345678
```



We stopped here last time…

Announcement

http://reallylearnportuguese.com/wp-content/uploads/2017/07/announcement.jpg

Summary of important concepts just seen

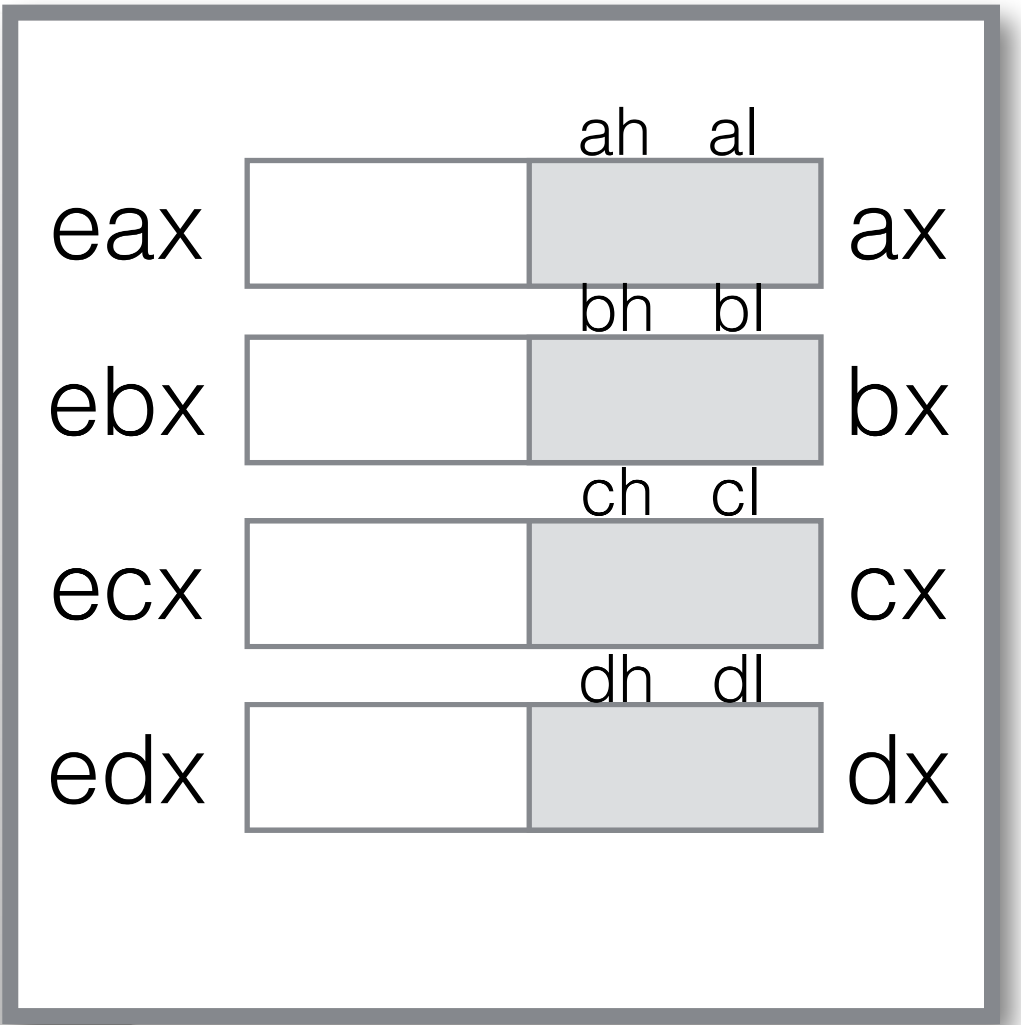
- Numbers
- Endianness
- Op Codes
- Machine Language
- Hexadecimal
- Executable Files

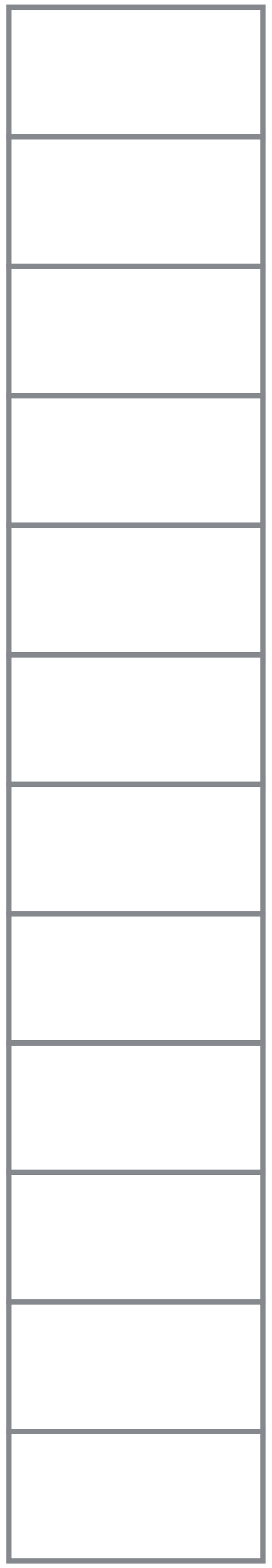
Return to the mov instruction

```
mov  dest, source
```

Test Cases

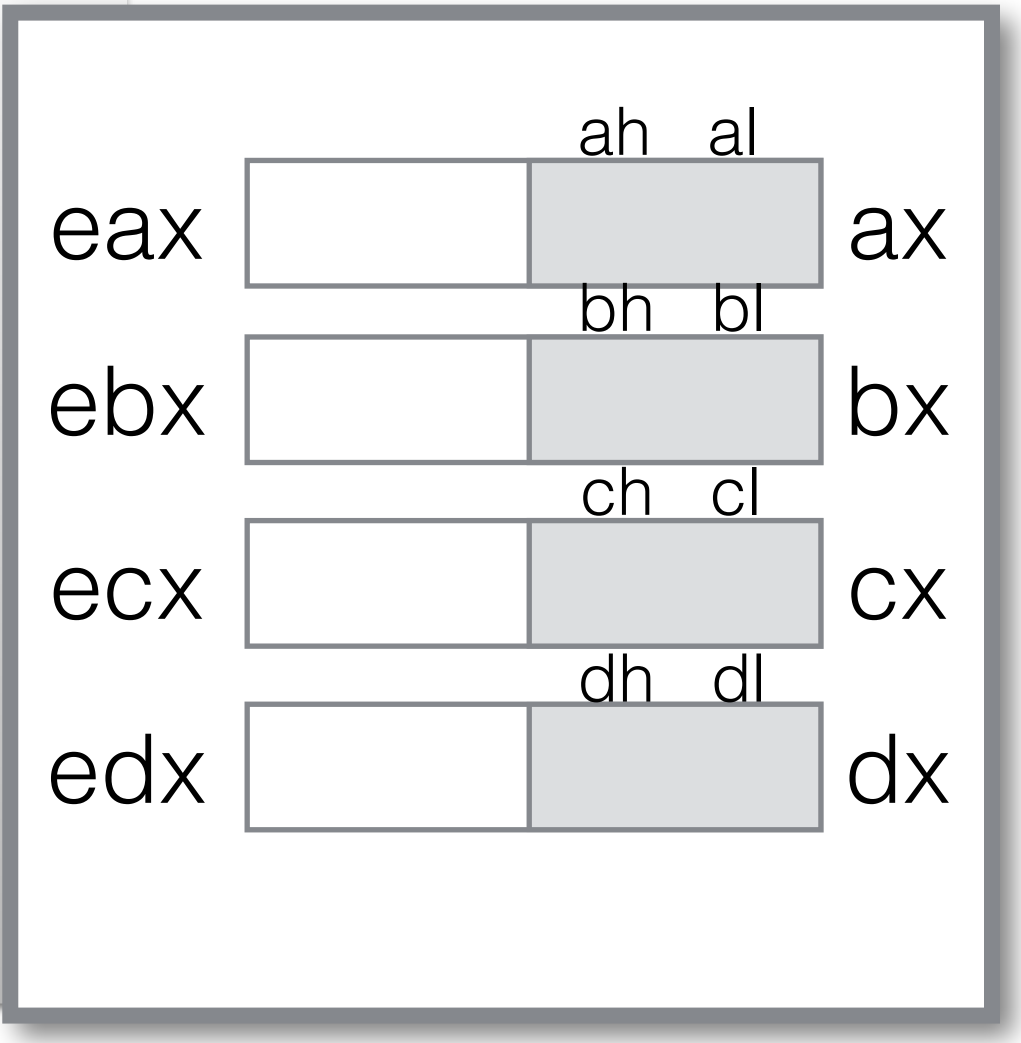
```
            section .data
lf          db          10
ch          db          0
a           dw          0x1234
b           dw          0
x           dd          0
y           dd          0x12345678

            section .text
; put lf in al
```

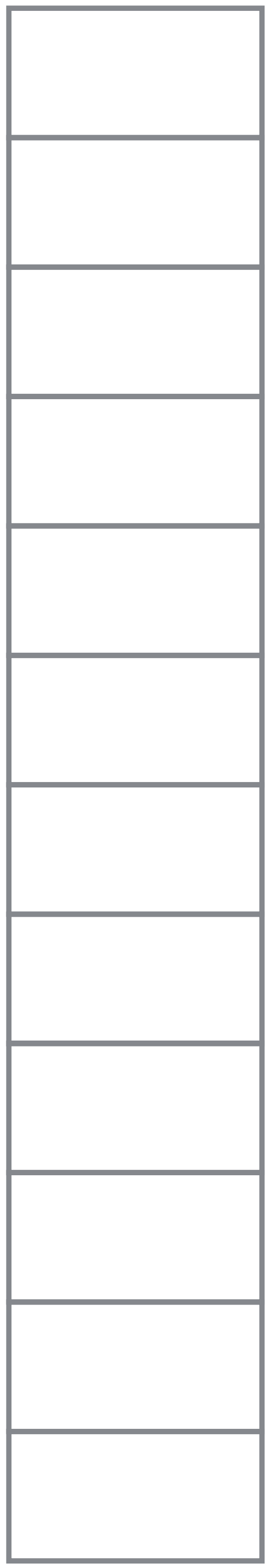




Test Cases

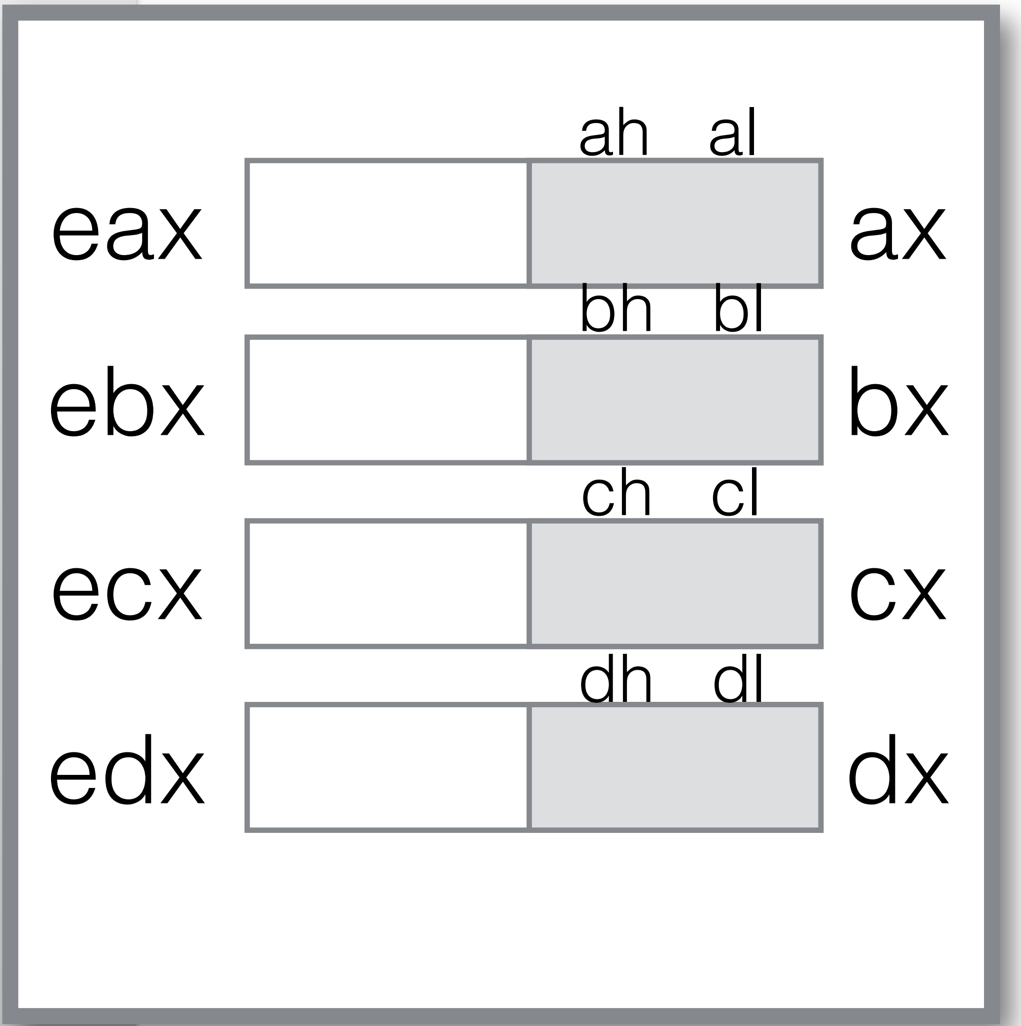
```
          section .data
lf        db      10
ch        db      0
a         dw      0x1234
b         dw      0
x         dd      0
y         dd      0x12345678

          section .text
; put al in ch
```

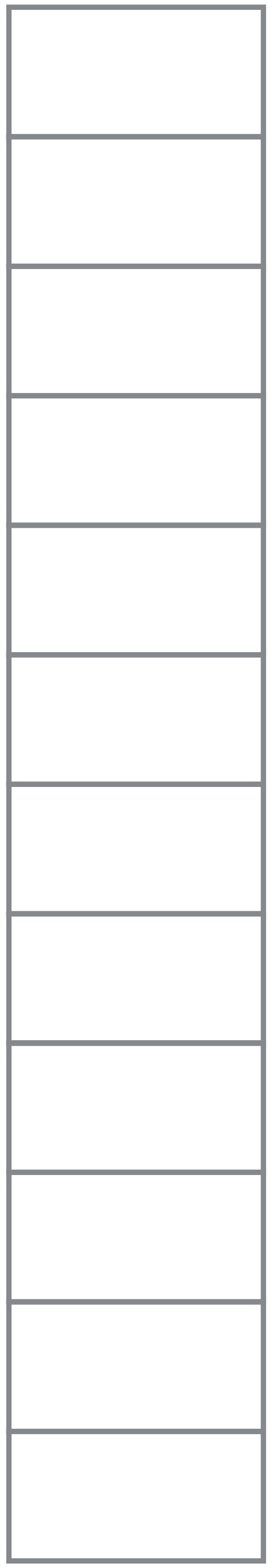




Test Cases

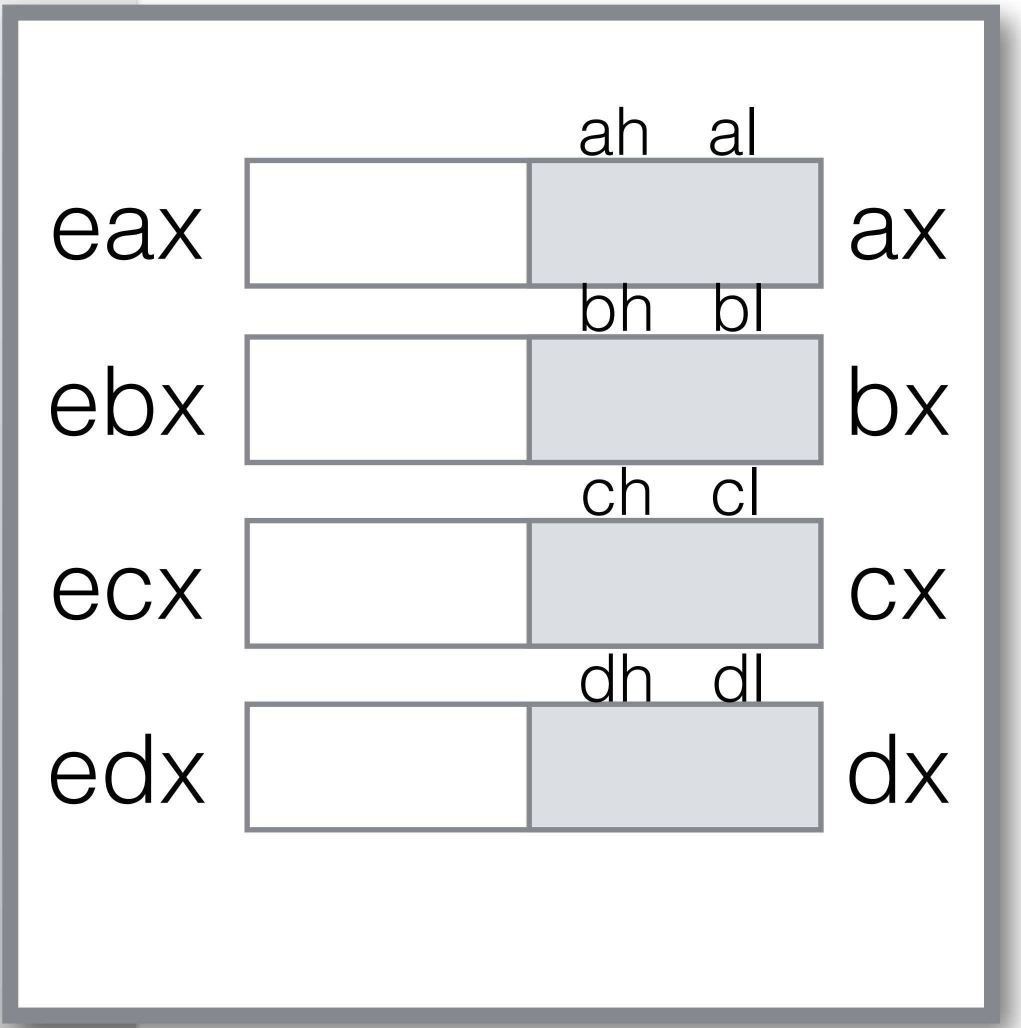
```
            section .data
lf          db        10
ch          db        0
a           dw        0x1234
b           dw        0
x           dd        0
y           dd        0x12345678

            section .text
; put a in bx


; put bx in b


; put bx in ax


; put 0 in cx
```

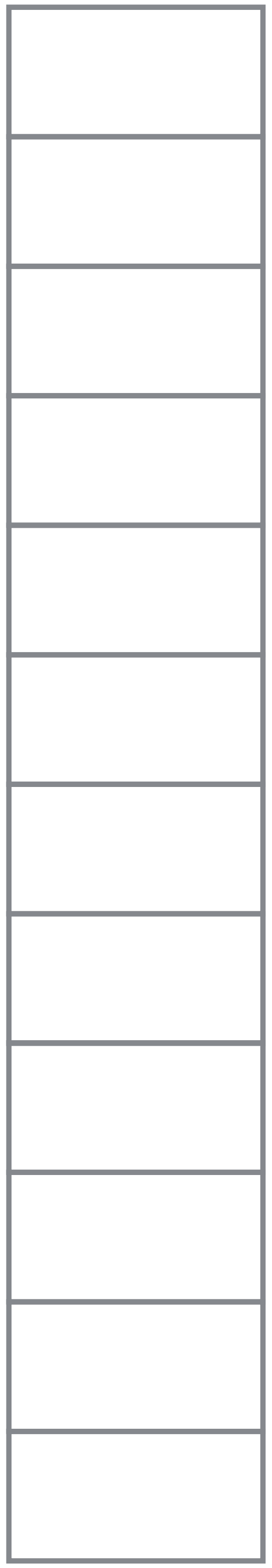




Test Cases

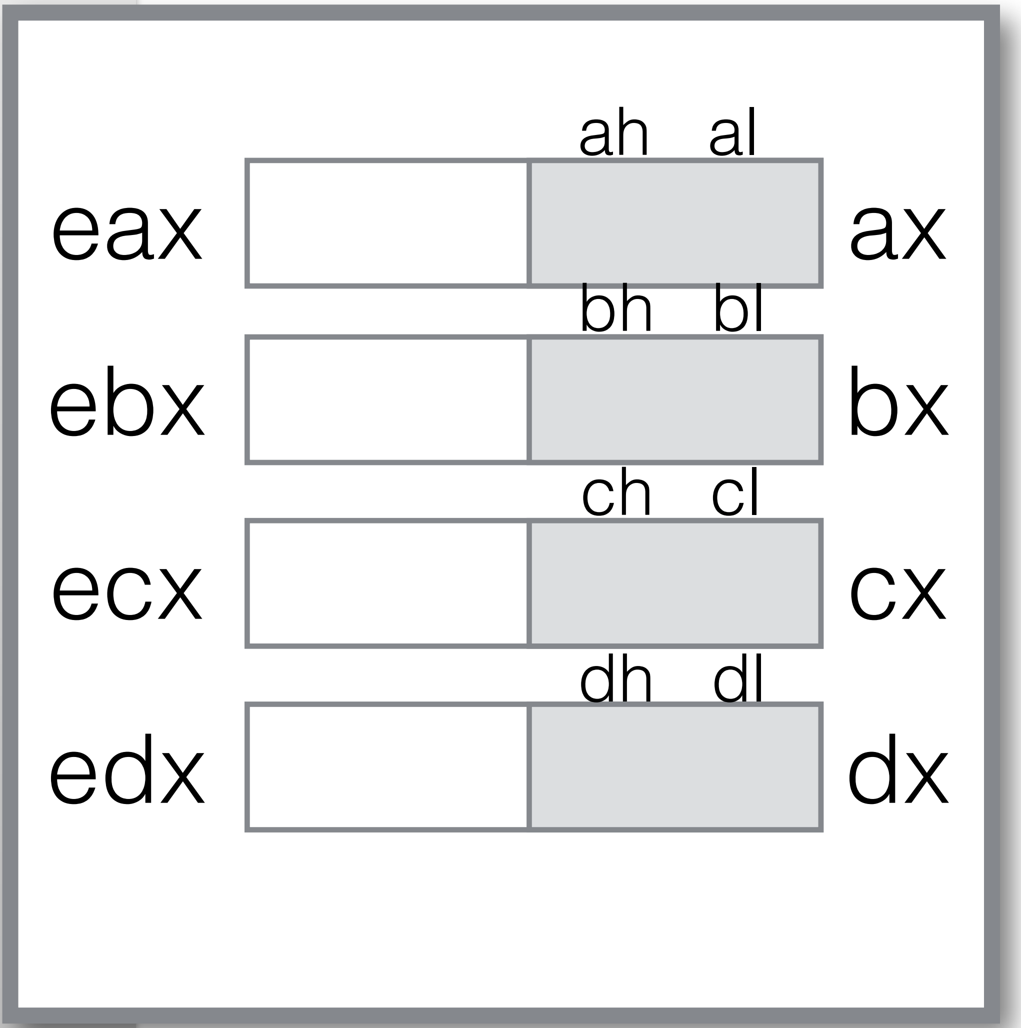
```
            section .data
lf          db      10
ch          db      0
a           dw      0x1234
b           dw      0
x           dd      0
y           dd      0x12345678

            section .text
; put x in eax


; put y in ecx


; put ecx in edx


; put ex into y
```

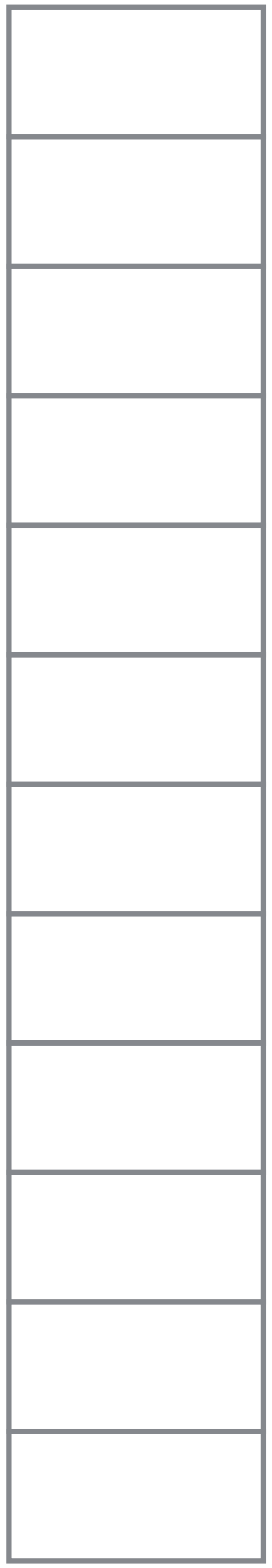




Test Cases

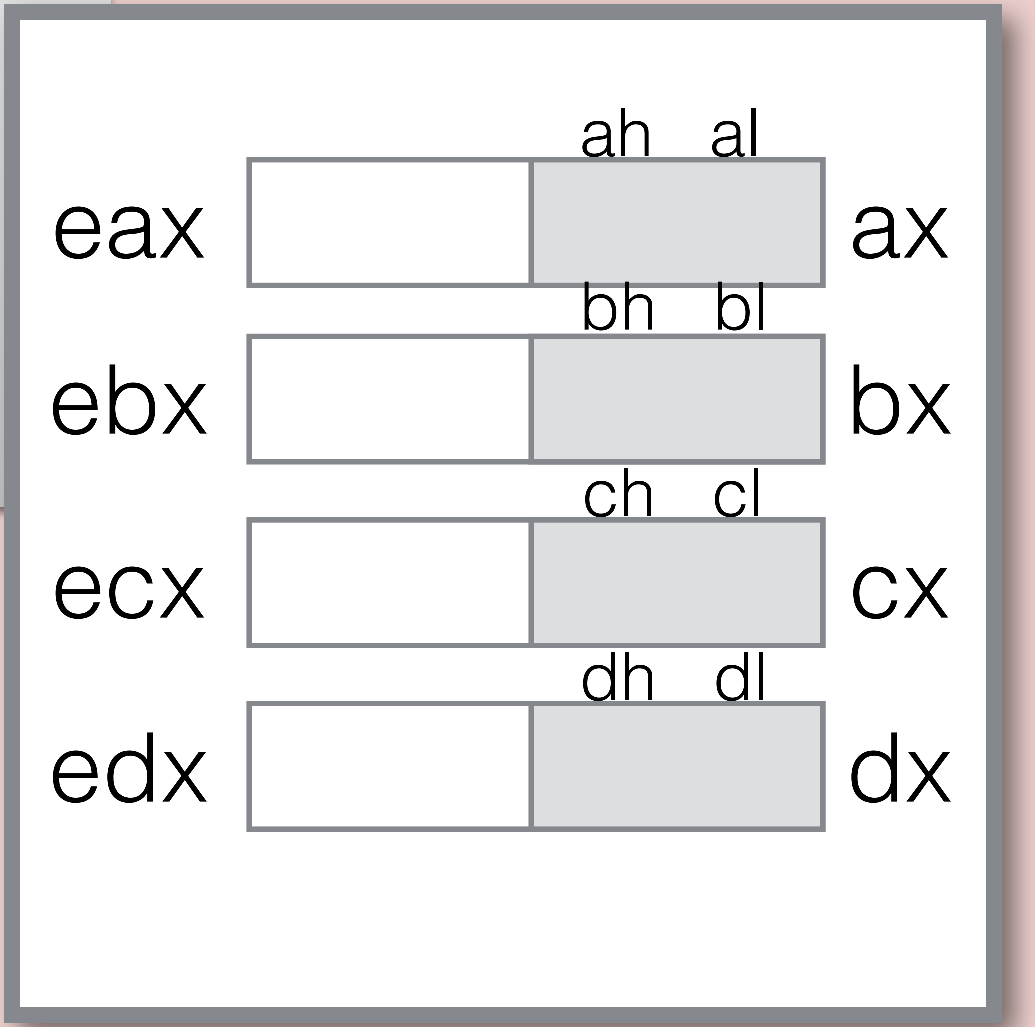
```
            section .data
lf          db        10
ch          db        0
a           dw        0x1234
b           dw        0
x           dd        0
y           dd        0x12345678

            section .text
; put 0 in ah


; put 3 in cx


; put 5 in edx


; put 0x12345678 into eax
```

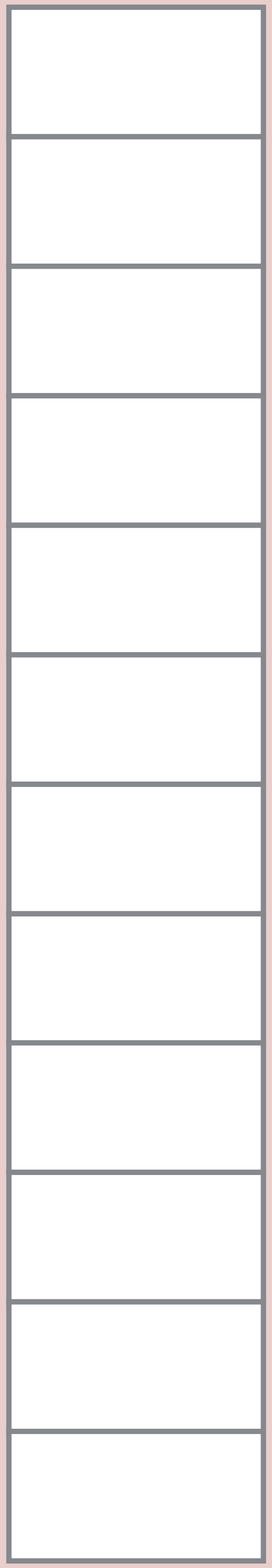




Buggy Program

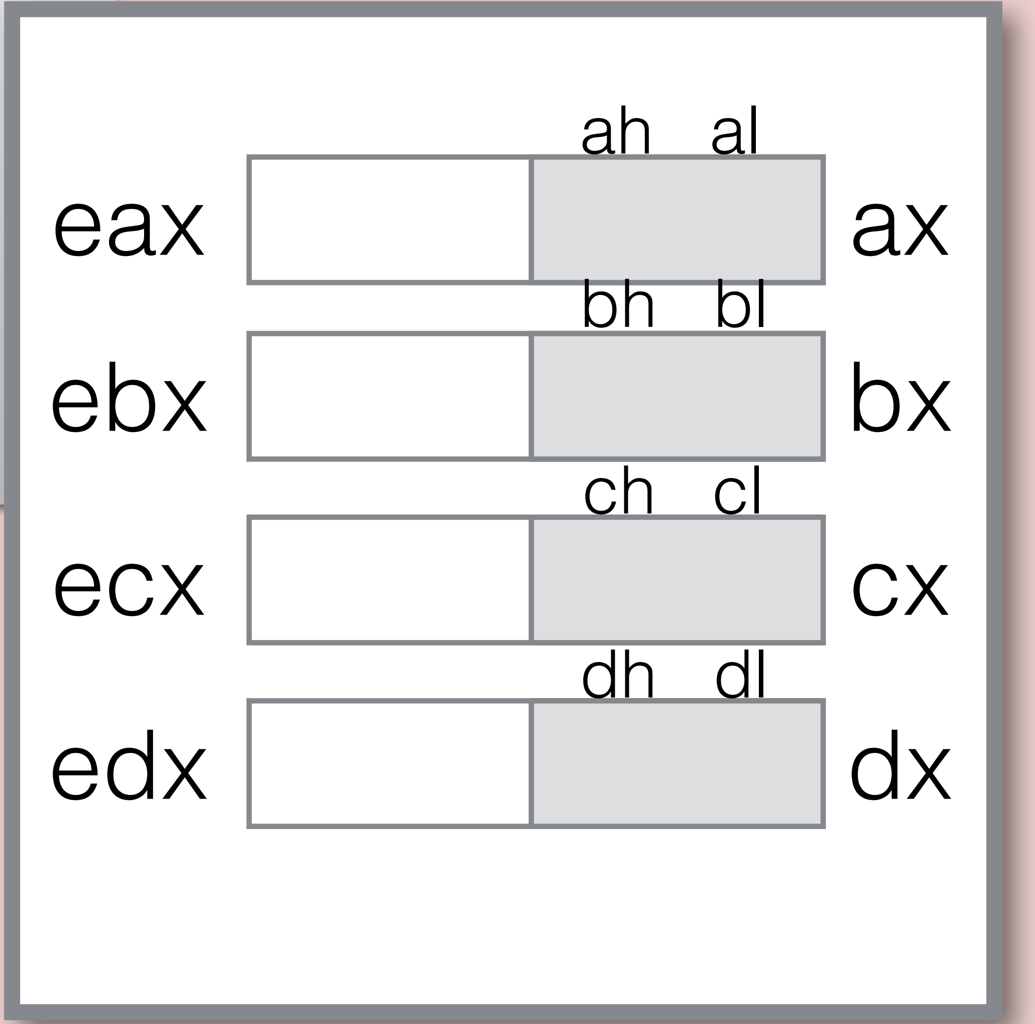
```
            section .data
lf          db          10
ch          db          0
a           dw          0x1234
b           dw          0
x           dd          0
y           dd          0x12345678

            section .text

            mov         eax, dword[a]
            mov         dword[b], eax
```

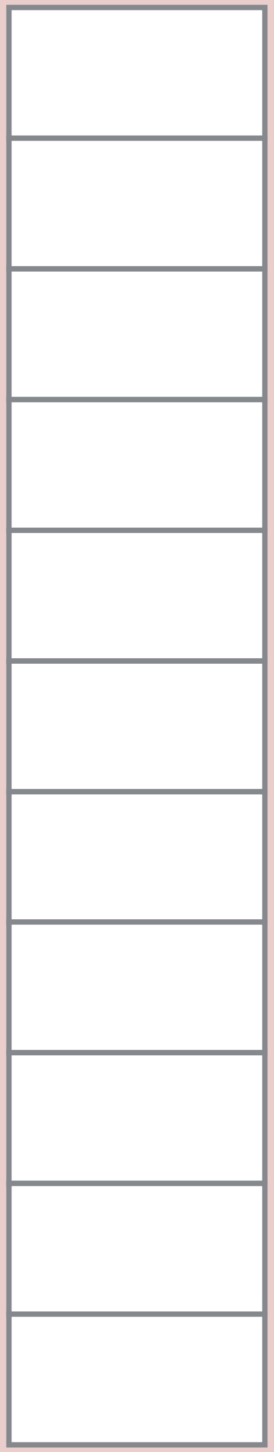




Buggy Program #2

```
          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
```





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

add dest, source

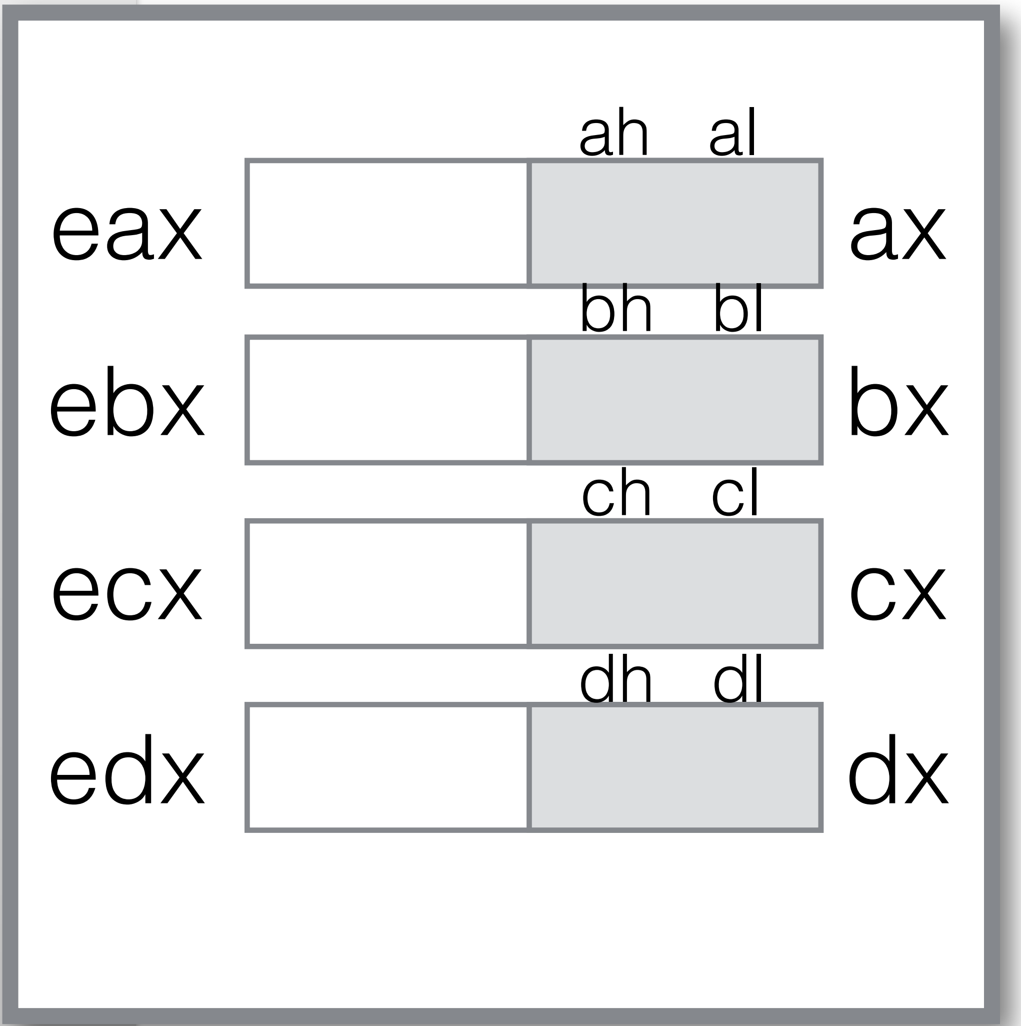
reg8
reg16
reg32
mem8
mem16
mem32

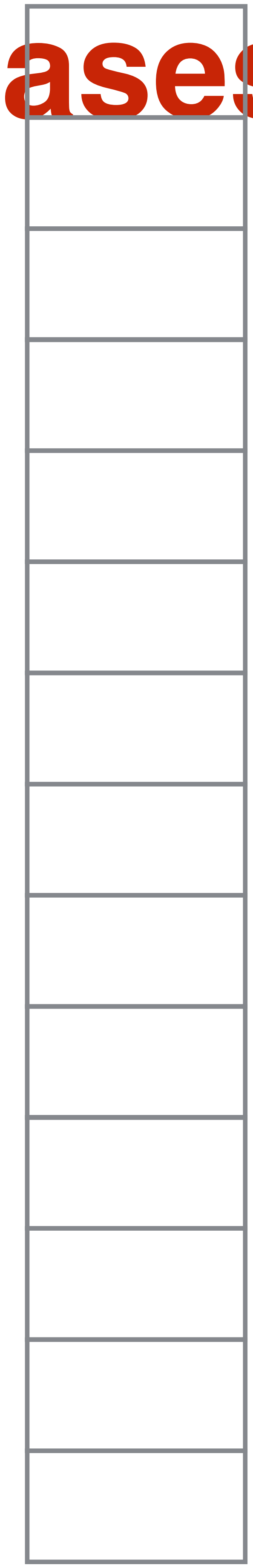
reg8
reg16
reg32
mem8
mem16
mem32
imm8
imm16
imm32

Sizes have to match

Test Cases

```
          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…

```
int x, y, sum;

x = 3;
y = 5;
sum = x + y;
```

We translated this into Assembly

Replace "int" by "short," and translate the program into assembly code.

We stopped here last time…