Data Size and Arithmetic: Examples and Sample Problems

ICS312
Machine-Level and Systems Programming

Henri Casanova (henric@hawaii.edu)
Example

```asm
mov al 0A7h  ; as a programmer, I view this
            ; as a unsigned, 1-byte quantity
            ; (decimal 167)
mov bl 0A7h  ; as a programmer, I view this
            ; as a signed 1-byte quantity (decimal -89)
movzx eax, al;  ; extend to a 4-byte value
            ; (000000A7)
movsx ebx, bl;  ; extend to a 4-byte value
            ; (FFFFFFFA7)
```
Practice

- Consider the following code

  ```
  mov al, 0B2h
  movsx eax, al
  mov bx, eax
  movzx ebx, bx
  ```

- What’s the final value of eax?
- What’s the final value of ebx?
## Practice (Solution)

<table>
<thead>
<tr>
<th>Operation</th>
<th>EAX</th>
<th>EBX</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mov al, 0B2h</code></td>
<td>?? ?? ?? B2</td>
<td>?? ?? ?? ??</td>
</tr>
<tr>
<td><code>movsx eax, al</code></td>
<td>FF FF FF B2</td>
<td>?? ?? ?? ??</td>
</tr>
<tr>
<td><code>mov bx, eax</code></td>
<td>FF FF FF B2</td>
<td>?? ?? FF B2</td>
</tr>
<tr>
<td><code>movzvx ebx, bx</code></td>
<td>FF FF FF B2</td>
<td>00 00 FF B2</td>
</tr>
</tbody>
</table>
Example

```
unsigned short ushort;    // 2-byte quantity
signed char schar;       // 1-byte quantity
int integer;             // 4-byte quantity

schar = 0xAF;
ingeneger = (int) schar;
geneger++;
ushort = integer;

printf("ushort = %d\n",ushort);
```

- What does this code print?
  - Or at least what’s the hex value of the decimal value it prints?
### Example

```c
unsigned short ushort;
signed char schar;
int integer;
schar = 0xAF;
integer = (int) schar;
integer++;  
ushort = integer;
printf("ushort = %d\n", ushort);
```

Because `printf` doesn't specify “h”
ushort is size augmented to 4-bytes
using movzx (because declared as unsigned): 00 00 FF B0
The number is then printed as a signed integer (“%d”): 65456
Carry/Overflow bits

- Which of these operations set the Carry bit to 1? (presumably we care because we think of these as unsigned operations)
  - 0F12 + F212 (2-byte quantities)
  - 00E3 + F74F (2-byte quantities)
  - F1 - FA (1-byte quantities)
  - FB12 - A3AA (2-byte quantities)
  - A314 - B010 (2-byte quantities)

- Which of these operations set the Overflow bit to 1? (presumably we care because we think of these as signed operations)
  - 00E3 + FF4F (2-byte quantities)
  - F1 - 7A (1-byte quantities)
Carry/Overflow bits (Solution)

- Which of these operations set the Carry bit to 1?

  \[
  \begin{align*}
  0F12 & \quad + \quad F212 \\
  = 10124 & \quad \text{Carry bit is set}
  \end{align*}
  \]

  \[
  \begin{align*}
  00E3 & \quad + \quad F74F \\
  = F832 & \quad \text{Carry bit is not set}
  \end{align*}
  \]

- F1 - FA: \quad F1 < FA \quad \text{Carry bit is set}
- FB12 - A3AA: \quad FB12 > A3AA \quad \text{Carry bit is not set}
- A314 - B010: \quad A314 < B010 \quad \text{Carry bit is set}
Carry/Overflow bits (Solution)

- Which of these operations set the Overflow bit to 1?
  - 00E3 + FF4F
    - 00E3 > 0, equal to decimal +227
    - FF4F < 0, 2’s complement = 00B0 + 1 = B1, equal do decimal -177
    - +243 - 177 = 50
    - 2 byte unsigned numbers are in [-32,768, +32,767]
    - Overflow bit is not set
  - F1 - 7A
    - F1 < 0, 2’s complement = 0E + 1 = 0F, equal to decimal -15
    - 7A > 0, equal to 122
    - -15 - 122 = -137
    - 1-byte unsigned numbers are in [-128, +127]
    - Overflow bit is set
Unsigned Overflow

mov al, 0F0h ; al = F0h
mov bl, 0A3h ; bl = A3h
add al, bl ; al = al + bl
movzx eax, al ; increase size for printing
call print_int ; print al as an integer

As a programmer we decided to do some computation with unsigned values.

We put value F0h in al (unsigned F0h is decimal 240)

We put value A3h in bl (unsigned A3h is decimal 163)

We add them together.

The “true” result should be decimal 240+163 = 403, which cannot be encoded on 8 bits (should be < 255)

But the processor just goes ahead: F0 + A3 = 193h, and then drops the leftmost bits to truncate to a 1-byte value to get 93h!

To call print_int, we need the integer in eax, so we movzx al into eax

print_int print the decimal value corresponding to 00000093h, that is: 147!

This is obviously wrong, and we can tell (or will be able to shortly) because the carry bit is in fact set to 1

Note that this is all correct if we assume signed values and replace movzx by movsx, but then our initial interpretation of the two values is different.
Signed Overflow

As a programmer we decided to do some computation with signed values.

We put value 9Ah in al (signed 9Ah is decimal -102).

We put value 73h in bl (signed 73h is decimal +115).

We subtract bl from al.

The “true” result should be decimal -102 - 115 = -217, which cannot be encoded on 8 bits (should be >= -128).

But the processor just goes ahead: 9A - 73 = 27h.

To call print_int, we need the integer in eax, so we movsx al into eax.

print_int prints the decimal value corresponding to 00000027h, that is: 39!

This is obviously wrong, and we can tell (or will be able to shortly) because the overflow bit is in fact set to 1.

Note that this is all correct if we assume unsigned values and replace movsx by movzx, but then our initial interpretation of the two values is different.