Converting between integer representations of different sizes

- **Integer types of different sizes**
  - **Recall** that **programming languages** provide `integer` types of ***different* size**:

```
byte    (1 byte integer (whole number) representation for values between -127..128)
short   (2 byte integer (whole number) representation for values between -32767..32768)
int     (4 byte integer (whole number) representation)
```

- **Representing the *same* value with *different* sizes**
  - **Fact:**
    - We can **represent** the ***same* value** using **integer representation** of ***different* sizes**

  - **Example 1:** (**positive values**)

```
The value 3 (three dots) is represented as follows:

  Using 1 byte  (8 bits):                              00000011
  Using 2 bytes (16 bits):                     0000000000000011
  Using 4 bytes (32 bits):   00000000000000000000000000000011
```

  - **Example 2:** (**negative values**)

```
The value -3 is represented as follows:

  Using 1 byte  (8 bits):                              11111101
  Using 2 bytes (16 bits):                     1111111111111101
  Using 4 bytes (32 bits):   11111111111111111111111111111101
```

- **Conversion**
  - **Conversion:**
    - **Conversion** = a **procedure** to **transform** one **kind** of ***representation*** into ***another* kind** of ***representation*** such that:
      - The ***value*** of that is **represented** by ***both* representation** are the ***same***

Example:

```
The representation of the value 3 in 8 bits is:

       00000011

When we convert this 8 bit representaion into a 16 bit representation,
the result is:

       0000000000000011

Because:

       The binary number represents the (same) value 3 !!!
```

- **General procedure to convert a *smaller* representation into a *larger* representation**
  - **Conversion procedure**:
    - If the **value** represented by the ***smaller* representation** is **positive**:
      - **Add** a bunch of **leading 0 digits** to the ***smaller* representation**

        (The number of **leading 0 digits** is equal to the **difference** in size between the **larger** and the **smaller** representation)

    - If the **value** represented by the ***smaller* representation** is **negative**:
      - **Add** a bunch of **leading 1 digits** to the ***smaller* representation**

        (The number of **leading 1 digits** is equal to the **difference** in size between the **larger** and the **smaller** representation)

  - **Note:**
    - ***Coincidentally***, the ***leading* binary digit** in a **2's complement representation** is:
      - **0** when the **value** represented is **positive**
      - **1** when the **value** represented is **negative**

  - We can **replace** the **above 2 rules** (one for **positive values** and one for **negative values**) by one ***single* rule**:
    - Let x be the **leading digit** in the **binary (2's complement)** representation
    - To **convert** a ***smaller*** representation into a ***larger*** representation:

- **Add** a bunch of **leading** x bits to the ***smaller* representation**

  (The number of **leading 1 digits** is equal to the **difference** in size between the **larger** and the **smaller** representation)

- **"Sign extention"**
  - The **operation**:

    - **Add** a bunch of **leading** x bits to a ***smaller* representation**

    is **called**:

    - ***Sign-bit* extension** or ***sign*-extension** for short....

- The M68000 ext instruction:
  - The ext instruction in **M68000** is used to:

    - **Convert** a ***smaller* 2's complement representation** into a ***larger* 2's complement representation**

  - **Syntax:**

```
EXT.W  Dn      Converts an 8-bits 2's complement representation in reg Dn
               to a 16 bits 2's complement representation (in reg Dn)

EXT.L  Dn      Converts a 16-bits 2's complement representation in reg Dn
               to a 32 bits 2's complement representation (in reg Dn)
```

    Note:

    - There is ***no* instruction** to **convert** a **byte (8 bits)** representation to a **long (32 bits)** representation.
    - **However**, we **can** accomplish the **conversion** from a **byte (8 bits)** representation to a **long (32 bits)** representation in **2 steps**:

```
* We start with an 8 bit representation for some value in register Dn

EXT.W   Dn      * Now we have a 16 bit representation in Dn for the same value
EXT.L   Dn      * Now we have a 32 bit representation in Dn for the same value
```

Examples:

```
(1) Starting with:

          +----------+----------+----------+----------+
     D0 = | 10101010 | 01010101 | 10101010 | 11111110 |
          +----------+----------+----------+----------+
                                              ^^^^^^^^
                                        This byte represents
                                        the value "-2"

  After "EXT.W D0", we will have:

          +----------+----------+----------+----------+
     D0 = | 10101010 | 01010101 | 11111111 | 11111110 |
          +----------+----------+----------+----------+
                                  ^^^^^^^^^^^^^^^^^^^
                                  This WORD represents
                                  the (same) value "-2" !

(2) Starting with:

          +----------+----------+----------+----------+
     D0 = | 10101010 | 01010101 | 11111111 | 11111100 |
          +----------+----------+----------+----------+
                                  ^^^^^^^^^^^^^^^^^^^
                                  This WORD represents
                                  the value "-2"

  After "EXT.L D0", we will have:

          +----------+----------+----------+----------+
     D0 = | 11111111 | 11111111 | 11111111 | 11111110 |
          +----------+----------+----------+----------+
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
            This LONG WORD represents the (same) value "-2" !
```

- **Example Program:** (Demo above code) *Example*
  - Prog file: click here (in /home/cs255000/demo/asm/ext1.s)

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `as255 ext1`
- To run: `m68000`

- **"Converting" a longer representation into a shorter representation**
  - **Fact:**
    - A ***shorter* representation** for the ***same* value** can be **obtained** by:
      - **Truncating** the **upper bits** of the ***longer* representation**

- **Example 1:** (**positive values**)

```
The value 3 (three dots) is represented as follows:

  Using 4 bytes (32 bits):   00000000000000000000000000000011

  Using 2 bytes (16 bits):                   0000000000000011 (= truncate upper 16 bits)
  Using 1 byte  (8 bits):                           00000011 (= truncate upper 24 bits)
```

- **Example 2:** (**negative values**)

```
The value -3 is represented as follows:

  Using 4 bytes (32 bits):   11111111111111111111111111111101

  Using 2 bytes (16 bits):                   1111111111111101
  Using 1 byte  (8 bits):                           11111101
```

- **In other words:**

  - If you need to **"convert"**:

```
int   --->  short
int   --->  byte
short --->  byte
```

    all you need to do is:

    - **Copy** the ***lower*** **part** of the **longer representation** to the **destination variable** of the ***shorter*** **type**

- **Example:** in next webpage !!!
