Assignment statement using different integer sizes

- **Automatic conversion in Java (and other high level programming languages)**
  - **Fact:**

    - ***Most* programming languages** provide an ***automatic* conversion feature** from a ***smaller* 2's complement representation** to a ***larger* 2's complement representation**

  - **Java** has **automatic conversion** for:

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

  - This is **what happens** in a **Java program**:

```
int i;
short s;
byte b;


The Java COMPILER translating the following assignment statements
in a high level language will **automatically** use
"EXT" assembler instructions:

-------------------------------------------------------------------------

s = b;      --->   MOVE.B  b, D0    * Get 8 bits from b into D0
                   EXT.W   D0       * Convert 8 bits to 16 bits representation
                   MOVE.W  D0, s    * Stores 16 bits into short var s
-------------------------------------------------------------------------

i = s;      --->   MOVE.W  s, D0    * Get 16 bits from s
                   EXT.L   D0       * Convert 16 bits to 32 bits representation
                   MOVE.L  D0, i    * Stores 32 bits into int var i
-------------------------------------------------------------------------

i = b;      --->   MOVE.B  b, D0    * Get 8 bits from b into D0
                   EXT.W   D0       * Convert 8 bits to 16 bits representation
                   EXT.L   D0       * Convert 16 bits to 32 bits representation
                   MOVE.L  D0, i    * Stores 32 bits into int var i
```

  - **Example Program:** (Demo above code)

    Example

    - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `as255 ext2`
- To run: `m680000`

- **"Automatic" conversion feature in compilers**
  - **Comment:**
    - The **automatic conversion** feature are ***only*** **available** in **compilers**:
      - The **compiler** can **detect** the ***size*** **difference** and ***insert*** the **conversion** instructions (such as `ext.w` and `ext.l`)

    - When ***you*** write **assembler code**:
      - ***You*** must ***insert*** the **conversion** instruction **when** it is ***necessary*** !!!

- **Converting a *larger* representation to a *smaller* representation**
  - **Recall: Important fact (from cs170):**
    - It is **not *safe*** convert from a ***larger*** **data type** to a ***smaller*** **type**

  - **What happens** when you convert from a ***larger*** **data type (representation)** to a ***smaller*** **type (representation)**:
    - **Prime-directive:**
      - The ***value*** that is **represented** must be ***equal*** **after** the **conversion**

    - **Recall:**

```
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

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
```

- **Procedure** to **convert** from a ***larger* data type (representation)** to a ***smaller* type (representation)**:

  - **Truncate** some of the **leading number** of **digit**

    I.e.: we **keep** the ***trailling* binary digits**

  The **number** of **binary digits** to **truncate** is **equal** to the **difference** in **size** between the ***larger* data type (representation)** and the ***smaller* type (representation)**

- **Example:**

```
short s;
byte b;
```

```
(1) s = 1;          (Stored as: 0000000000000001 --- represents value 1)

    b = (byte) s;

    Jave will assign the lower 8 bits of s to b:

        b will contain the binary representation 00000001

    which correctly represents the value 1
```

```
(2) s = -2;        (Stored as: 1111111111111110 --- represents value -1)

    b = (byte) s;

    Jave will assign the lower 8 bits of s to b:

        b will contain the binary representation 11111110

    which correctly represents the value -2
```

- This is **what happens** in a **Java program**:

```
int i;
short s;
byte b;


The Java COMPILER truncate the representaion
**ONLY** if you **beg** for it:
```

```
b = (byte) s;      ---->   move.w  s, d0
                           move.b  d0, b
```

```
b = (byte) i;      ---->   move.l  i, d0
                           move.b  d0, b
```

```
s = (short) i;     ---->   move.l  i, d0
                           move.w  d0, s
```

- **Example Program:** (Demo above code)

Example

  - Prog file: click here

How to run the program:

  - **Right click** on link and **save** in a scratch directory
  - To compile: `as255 ext3`
  - To run: `m680000`

- **Example Program:** (Demo above code)

Example

  - Prog file: click here (/home/cs255000/demo/asm/ext3.s)

How to run the program:

  - **Right click** on link and **save** in a scratch directory
  - To compile: `as255 ext3`
  - To run: `./simex`

- **Conversion error**: occurs when you **convert** a **value** that is **too big** for the ***smaller* representation**

Example:

```
s = 129;          (Stored as:  0000000100000001  --- represents value 129)

b = (byte) s;

   Jave will assign the lower 8 bits of s to b:

       b will contain the binary representation  00000001

   which represent the value 1
   and not the orignal value 129
```

- **Example Program:** (Demo above code)

Example

  - Prog file: click here (/home/cs255000/demo/asm/ext4.s)

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `as255 ext4`
- To run: `./simex`