Arithmetic expressions using integers of different sizes

- **Automatic conversion in *arithmetic* operations**
  - **Recall** that:
    - The **CPU** can ***only*** operate on **2 operands** of the ***same* size**
      - 2 **integers**
      - 2 **shorts**
      - 2 **bytes**
    - **Therefore:**
      - ***Mixed* size operands** must be ***converted*** before the **CPU** can perform the desired **operation** !!!
  - **Question:**
    - **How** show the **operands** of ***different* size** be **converted** ???
  - **Answer** from **CS170**:
    - When a ***lower*-precision type** and a ***higher*-precision type** are used in an **arithmetic operation**:
      - the ***lower*-precision representation** is ***converted*** to the ***higher*-precision representation** **first** before the **arithmetic operation** is **performed**

- **Example of mixed size operations**
  - **Integer = Integer + Integer**

```
In Java:

  int i, j;

  i = i + j;
```

In assembler: (**no** conversion is **needed** !)

```
  move.l i, d0      * Get 32 bits from i to D0
  move.l j, d1      * Get 32 bits from j to D1

  add.l  d0,d1      * Same szie, we can add

  move.l d1, i      * Store 32 bits result in i
```

  - **Integer = Integer + Short**

```
In Java:

  int i;
  short s;

  i = i + s;
```

In assembler:

```
  move.l i, d0      * Get 32 bits from i to D0
  move.w s, d1      * Get 16 bits from s to D1

  ext.l  d1         * Converts 16 bit repr to 32 bit repr

  add.l  d0,d1      * Add the two 32 bits numbers together
  move.l d1, i      * Store the 32 bit number in i
```

Demo: /home/cs255000/demo/asm/ext5.s

  - **Short = Short + Integer**

```
In Java:

  int i;
  short s;

  s = (short) (s + i);      // Casting means danger !
```

In assembler:

```
  move.l i, d0      * Get 32 bits from i to D0
  move.w s, d1      * Get 16 bits from s to D1

  ext.l  d1         * Converts 16 bit repr to 32 bit repr

  add.l  d0,d1      * Add the two 32 bits numbers together
  move.w d1, s      * Store the 16 bit number in s
```

- **NOTE:**

  - The **assembler instruction**:

    ```
      move.w d1, s      * Store the 16 bit number in s
    ```

    can **result** in **overflow** error

  - For this reason, in **Java**, the **programmer** must use an ***explicit* casting**:

    ```
       s = (short) (s + i) ;
    ```

    The **Java compiler** will ***not* allow** you to write **this**:

    ```
      s = s + i;         // Java will report error !!!
    ```

The **(short)** casting operation is to let you tell **Java** that **you know it's dangerous** and "please let me do it anyway".

- **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`

- **The assembler programmer: power and responsibility**
  - **Assembler programming:**
    - **Assembler programming** is the ***most*** **difficult way** to **program** a computer
      - It is ***assumed*** that **assembler programmers know** what they are **doing**.....
    - **Assembler programmers** do ***not*** ask ***permission*** to do ***anything*** !!!

  (If you **dare** to program in **assembler**, you **should be man enough** to take on the **responsibility** to know what to do... You are **not** in **Kansas** anymore (to quote Dorothy))

- **Summary**

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

How to perform all possible conversion between int, short and byte:

```
s = b;      --->   MOVE.B  b, D0
```

```
                    EXT.W   D0
                    MOVE.W  D0, s

b = s       --->    MOVE.W  s, D0
                    MOVE.B  D0,b        * Can result in overflow....
```

```
i = s;      --->    MOVE.W  s, D0
                    EXT.L   D0
                    MOVE.L  D0, i

s = i;      --->    MOVE.L  i, D0
                    MOVE.W  D0,s        * Can result in overflow....
```

```
i = b;      --->    MOVE.B  b, D0
                    EXT.W   D0          * First convert to WORD
                    EXT.L   D0          * THEN convert to LONG WORD !
                    MOVE.L  D0, i

b = i;      --->    MOVE.L  i, D0
                    MOVE.B  D0,b        * Can result in overflow....
```
