The MULS instruction - know when to use `ext.l` to convert into `long`

- **RECALL: Mulitply instruction in M68000**
  - **M68000** can ***only*** multiply two **16-bits integers** (due to the technological limitation at the time - 1980)
  - The syntax of the **multiply** instruction of M68000 is:

```
MULS <ea>, Dn        Multiply the 16 bit integer value in
                     the operand specified by <ea> to
                     the 16 bit value in data register Dn

                     The result is always 32 bits and it is
                     stored in data register Dn
```

  - Notice that you do **not** have any choice for operand size.

- **Multiple int data types**
  - **Facts:**
    - `int` data type uses **32 bytes** to **represent** the **value**
    - When an `int` is used in the `muls` instruction:
      - The `32 bit` representation is ***truncated*** to a **16 bit representation**
  - **Example:**

```
int i1, i2, i3;

i3 = i1 * i2;

 In assembler code:
                      move.l i1,D0      * get 32 bits value i1 in reg D0
                      move.l i2,D1      * get 32 bits value i2 in reg D1

                      muls   D1,D0      * D0 = D0*D1
                                        * We only use 16 bits in D0 and D1
                                        * So we have converted the int
```

```
                                    * into a short before we multiply !!
                                    * Note: result is correct as long as
                                    *       the values in D0 and D1 is small

                        move.l D0,i3        * Store i1*i2 to i3
                                            * The product is 32 bits !!!
```

- **Multiply with byte size operands**
  - **Important fact:**

    - `MULS` will ***always*** use **16 bits** operands !!!

    - So we **must *convert*** a **byte (8 bits) representation** into a **16 bit representation *before*** we use `MULS` !!!

    Example:

```
byte b1, b2, b3;

b3 = b1 * b2;

In assembler code:

                MOVE.B b1, D0     * D0 = b1 (8 bits)
                EXT.W  D0         * D0 now has a 16 bits representation !!

                MOVE.B b2, D1     * D1 = b2 (8 bits)
                EXT.W  D1         * D1 now has a 16 bits representation !!

                MULS   D1, D0     * D0 = b1 * b2 (32 bits)
                                  * The product is 32 bits !!!

                MOVE.B D0, b3     * Move byte value to b3
                                  (We have actually converted an int
                                   into a byte !)
```

  - Try out this demo program yourself: click here

- **Some more examples**
  - **More examples:**

```
int a;
short b;
```

```
byte c;

a = b * c;
                move.w b, d0      (16 bits valid in d0)
                move.b c, d1      (8  bits valid in d1)
                ext.w  d1         (16 bits valid in d1)

                * Now have two 16 bits values and can use muls !!

                muls   d0, d1     (32 bit result in d1)
                move.l d1, a      (store 32 bits in a)
```

```
b = a * c;
                move.l a, d0      (32 bits valid in d0)
                                  (We will only use 16 bits)
                move.b c, d1      (8  bits valid in d1)
                ext.w  d1         (16 bits valid in d1)

                * Now have two 16 bits values and can use muls !!

                muls   d0, d1     (32 bit result in d1)
                                  (We will only use 16 bits)
                move.w d1, b      (store 16 bits in b)
```

```
c = a * b;
                move.l a, d0      (32 bits valid in d0)
                                  (We will only use 16 bits)
                move.w c, d1      (16  bits valid in d1)

                * muls will only use 16 bits from d0

                muls   d0, d1     (32 bit result in d1)
                                  (We will only use 8 bits)
                move.b d1, c      (store 8 bits in c)
```