The DIVS instruction - know when to use `ext` to convert into `long`

- **Review: the divide instruction in M68000**
  - **M68000** can only divide a **32-bits integer number** by a **16-bits integer number**
  - The **syntax** of the **DIVS** instruction is:

```
   DIVS <ea>, Dn        Divides a 32 bit value in data register Dn
                        by a 16-bit value specified by <ea>

      (1) the quotient is stored in the lower 16 bits of data register Dn

      (2) the remainder is stored in the upper 16 bits of data register Dn
```

- **Review: SWAP**
  - **SWAP**:
    - The "`SWAP Dn`" instruction **exchanges** the **upper** and **lower halves** of the data register **Dn**.
  - **Example:**

```
       +----------+----------+----------+----------+
   D0 = | 00000000 | 00000001 | 00000000 | 00000010 |
       +----------+----------+----------+----------+

 After SWAP D0:

       +----------+----------+----------+----------+
   D0 = | 00000000 | 00000010 | 00000000 | 00000001 |
       +----------+----------+----------+----------+
```

- **Division with integer**
  - **Fact:**
    - The **quotient** and the **remainder** of `DIVS` are **16 bits** results

- An `int` value is represented in **32 bits**
- **Therfore:**
  - You **must** convert the **16 bits** (quotient or remainder) into **32 bits before** you can **store** it in an `int` variable

- **Example: quotient**

```
   int i, j, k;

   k = i / j;

In assembler:

      MOVE.L  i, D0          * Get 32 bits from i into D0
      MOVE.L  j, D1          * Get 32 bits from j into D0
      DIVS    D1, D0         * Divide 32 bits in D0 by 16 bits in D1
                               (We actually converted the int in D1 to a short
                                It will work as long as j is small)

      * The quotient is stored as 16 bits
      * We must convert to 32 bits before storing result to int variable k !!!

      EXT.L  D0              * The quotient is now stored as 32 bits

      MOVE.L D0, k           * Store 32 bits quotient to the int varibale k
```

- **Example: remainder**

```
   int i, j, k;

   k = i % j;

In assembler:

      MOVE.L  i, D0          * Get 32 bits from i into D0
      MOVE.L  j, D1          * Get 32 bits from j into D0
      DIVS    D1, D0         * Divide 32 bits in D0 by 16 bits in D1
                               (We actually converted the int in D1 to a short
                                It will work as long as j is small)

      SWAP D0                * Move the remainder to the lower 16 bits

      * The remainder is stored as 16 bits
      * We must convert to 32 bits before storing result to int variable k !!!

      EXT.L  D0              * The remainder is now stored as 32 bits
```

```
        MOVE.L D0, k           * Store 32 bits remainder to the int variable k
```

- **More examples**
    - **More examples:**

```
   int a;
   short b;
   byte c;

   a = b / c;
                  move.w b, d0      (16 bits valid in d0)
                  ext.l  d0         (You need 32 bits to divide)
                  move.b c, d1      (8  bits valid in d1)
                  ext.w  d1         (You need 16 bits valid to divide)

                  divs   d1, d0     (Quotient is 16 bits in d0)
                  ext.l  d0         (Make 32 bit representation)
                  move.l d0, a      (store 32 bits in a)

   b = a / c;
                  move.l a, d0      (32 bits valid in d0)
                  move.b c, d1      (8  bits valid in d1)
                  ext.w  d1         (You need 16 bits valid to divide)

                  divs   d1, d0     (Quotient is 16 bits in d0)
                  move.l d0, b      (store 16 bits in b)

   c = a / b;
                  move.l a, d0      (32 bits valid in d0)
                  move.w b, d1      (16 bits valid in d1)

                  divs   d1, d0     (Quotient is 16 bits in d0)
                                    (We will only use 8 bits)
                  move.b d0, c      (store 8 bits in c)
```

- Compute the **remainder**: (you just have to **swap** the result of the **division**)

```
   int a;
   short b;
   byte c;

   a = b % c;
                  move.w b, d0      (16 bits valid in d0)
                  ext.l  d0         (You need 32 bits to divide)
                  move.b c, d1      (8  bits valid in d1)
                  ext.w  d1         (You need 16 bits valid to divide)

                  divs   d1, d0     (Quotient is 16 bits in d0)
                  swap   d0         (Remainder is now the lower 16 bits)
                  ext.l  d0         (Make 32 bit representation)
                  move.l d0, a      (store 32 bits in a)

   b = a % c;
                  move.l a, d0      (32 bits valid in d0)
                  move.b c, d1      (8  bits valid in d1)
                  ext.w  d1         (You need 16 bits valid to divide)

                  divs   d1, d0     (Quotient is 16 bits in d0)
                  swap   d0         (Remainder is now the lower 16 bits)
                  move.l d0, b      (store 16 bits in b)

   c = a % b;
                  move.l a, d0      (32 bits valid in d0)
                  move.w b, d1      (16 bits valid in d1)

                  divs   d1, d0     (Quotient is 16 bits in d0)
                  swap   d0         (Remainder is now the lower 16 bits)
                                    (We will only use 8 bits)
                  move.b d0, c      (store 8 bits in c)
```

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

  Example

  - Prog file: click here