Mixed type operations with arrays

- **Important facts when using arrays**
  - **Important facts:**

    - An **address** is used to **identify** a **memory location**
      - An **address** is like a **social security number** for a **memory cell** !!!
    - An **address** is **always 32 bits**
      - A **SSN** for **humans** is ***always* 9 digits**:

        ```
        123-45-7890
        090-00-0012     <---- Leading 0's are significant !!!
        000-00-0001     <---- Leading 0's are significant !!!
        ```
    - **Therfore:**
      - You ***must*** use:

        ```
        move.l    #Label, ....       * Use .l for 32 bits address !!!
        ```

- **Assignment with arrays of mixed data type**
  - **Large = small**

```
  int   A[10];
  short B[10];

  A[2] = B[3];
```

```
In assembler code:

                movea.l  #B, a0            An address is always 32 bits,
                                           so use movea.l

                move.w   6(a0), d0         d0 = B[3] (16 bits)
                                           You are transfering B[3]
                                           which is a short, so move.w

                ext.l    d0                convert B[3] into 32 bits
```

```
                movea.l  #A, a0          An address is always 32 bits,
                                         so use movea.l
                move.l   d0, 8(a0)       You are transfering 32 bits
                                         so move.l
```

- **Small = large** (**dangerous conversion !**)

```
  int   A[10];
  short B[10];

  B[3] = A[2];
```

```
In assembler code:

                movea.l  #A, a0          An address is always 32 bits,
                                         so use movea.l
                move.l   8(a0), d0       d0 = A[2] (32 bits)

                movea.l  #B, a0          An address is always 32 bits,
                                         so use movea.l
                move.2   d0, 6(a0)       Transfer only the lower 16 bits of d0
                                         into B[3]
```

- **Calculations with arrays of mixed type: convert smaller to larger representation first**
  - **Example 1:** `int = int + short`

```
In Java:

     int A[10];
     short B[10];

     A[4] = A[7] + B[5];
```

```
In assembler:

        movea.l  #A, a0       * Get the base address of array A in A0
        move.l   28(a0), d0   * Get the int value A[7] in D0

        movea.l  #B, a1       * Get the base address of array B in A1
        move.w   10(a1), d1   * Get the short value B[5] in D1

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

        add.l    d1, d0       * Add two 32 bit numbers together

        move.l   d0, 28(a0)   * Store the 32 bits result in A[4]
```

  - **Example 2:** `short = int + short`

```
In Java:

     int A[10];
```

```
    short B[10];

    B[4] = (short) (A[7] + B[5]);
```

In assembler:

```
        movea.l  #A, a0       * Get the base address of array A in A0
        move.l   28(a0), d0   * Get the int value A[7] in D0

        movea.l  #B, a1       * Get the base address of array B in A1
        move.w   10(a1), d1   * Get the short value B[5] in D1

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

        add.l    d1, d0       * Add two 32 bit numbers together
                              * The result is 32 bits !!!

        move.w   d0, 8(a0)    * Store only 16 bits from D0 in B[4]
```