The ADD and SUBTRACT instructions

- **The ADD machine instruction**
  - **Fact:**

    - The **M68000** can perform the **add** operation in a **data register** or in an ***address* register**

    - The **CPU** can ***only*** add **values** of the ***same* representation** with each other

      In other words:

      - The **CPU** can **add** 2 **byte representations**, **or**
      - The **CPU** can **add** 2 **word (2 bytes) representations**, **or**
      - The **CPU** can **add** 2 **long word (4 bytes) representations**,

- **ADD: add in a data register**
  - **Instruction** to **add** in a **data register**:

```
    ADD.B  Src, Dn     * Add the  8 bits values: Dn = Dn + Src
    ADD.W  Src, Dn     * Add the 16 bits values: Dn = Dn + Src
    ADD.L  Src, Dn     * Add the 32 bits values: Dn = Dn + Src
```

  - **Example 1:** adding **integer** variables

```
   int x, y, z;         (We assumed x,y,z have been defined)

   z = x + y;

In assembler code:

        move.l   x,D0  * Get 32 bits representation of x in D0
```

```
        move.l   y,D1  * Get 32 bits representation of y in D1

        add.l    D0,D1 * Add the two 32 bits representations together
                       * The sum is stored in D1

        move.l   D1,z  * Store the 32 bits representation in z
```

- **Example 2:** adding **short** variables

```
   short x, y, z;         (We assumed x,y,z have been defined)

   z = x + y;

In assembler code:

        move.w   x,D0  * Get 16 bits representation of x in D0
        move.w   y,D1  * Get 16 bits representation of y in D1

        add.w    D0,D1 * Add the two 16 bits representations together
                       * The sum is stored in D1 (16 bits)

        move.w   D1,z  * Store the 16 bits representation in z
```

- **Example 3:** adding **integer array elements**

```
   int B[10]         (We assumed array B have been defined)

   B[4] = B[3] + B[7];

In assembler code:

        movea.l  #B,A0      * A0 = base address of array B

        move.l   12(A0),D0  * Get 32 bits representation of B[3] in D0
        move.l   28(A0),D1  * Get 32 bits representation of B[7] in D1

        add.l    D0,D1      * Add the two 32 bits representations together
                            * The sum is stored in D1

        move.l   D1,16(A0)  * Store the 32 bits representation in B[4]
```

- **Example 4:** adding **short array elements**

```
   short B[10;         (We assumed array B have been defined)
```

```
   B[4] = B[3] + B[7];

In assembler code:

        movea.l  #B,A0      * A0 = base address of array B
                            * An address is 32 bits, so we use .l !!!

        move.w    6(A0),D0  * Get 16 bits representation of B[3] in D0
        move.w   14(A0),D1  * Get 16 bits representation of B[7] in D1

        add.w     D0,D1     * Add the two 16 bits representations together
                            * The sum is stored in D1 (16 bits)

        move.w   D1,8(A0)   * Store the 16 bits representation in B[4]
```

- **ADDA: Adding in an address register**
  - **Fact:**

    - When the **destination** of the **add instruction** is an ***address* register**, the **instruction nmemonic** gets an **a** appended to the **tail**

    - When the **destination** is an ***address* register**, you can ***only*** use **word size** and **long word size** instructions

  - The **instruction** to **add** in an **address register** is:

```
    ADDA.W   Src, An    * Add the 16 bit value Src to the address reg An
                        * I.e.: An = An + Src
    ADDA.L   Sec, An    * Add the 32 bit value Src to the address reg An
                        * I.e.: An = An + Src
```

- **The SUB machine instruction**
  - **Subtract** in a **data register**:

```
    SUB.B  Src, Dn     * Subtract the  8 bits values: Dn = Dn - Src
    SUB.W  Src, Dn     * Subtract the 16 bits values: Dn = Dn - Src
```

```
SUB.L  Src, Dn     * Subtract the 32 bits values: Dn = Dn - Src
```

- **Note:**

  - The **subtract** instruction is ***very*** **similar** to the **add** instruction

  - You can use the **above examples** to **learn about** the **subtract instruction**

    The **examples** will **work** if you **replace**:

    ```
    ADD   --->   SUB
    ```

- **Subtract** in an **address register**:

```
SUBA.W  Src, An     * Subtract the 16 bits values: An = An - Src
SUBA.L  Src, An     * Subtract the 32 bits values: An = An - Src
```

- **Reminder: When to use data and address registers**

  - **Data registers:**

    - When you perform **calculation**: always use **data registers**

  - **Address registers:**

    - When you **access** an **array element** or a **field** in a **linked list**: use ***address*** **registers**

    Example:

```
   short B[10];         (We assumed array B have been defined)

   B[4] = B[3] + B[7];

In assembler code:

        /* ============================================================
           We want to access B[3]:
             ==> put base address in address register
           ============================================================ */
        movea.l  #B,A0      * A0 = base address of array B
                            * An address is 32 bits, so we use .l !!!

        /* ============================================================
           We want to compute B[3] + B[7]
             ==> put the values B[3] and B[7] in data register
           ============================================================ */
        move.w    6(A0),D0  * Get 16 bits representation of B[3] in D0
        move.w   14(A0),D1  * Get 16 bits representation of B[7] in D1

        add.w     D0,D1     * Add the two 16 bits representations together
                            * The sum is stored in D1 (16 bits)

        move.w   D1,8(A0)   * Store the 16 bits representation in B[4]
```