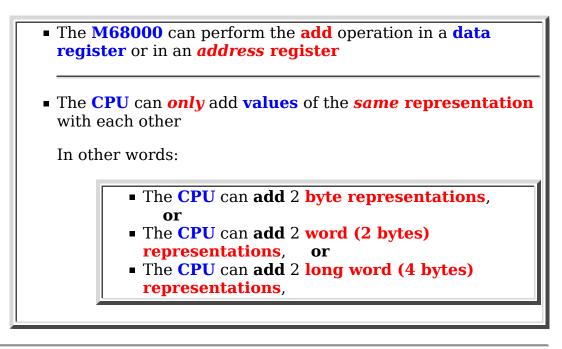
The ADD and SUBTRACT instructions

• The ADD machine instruction

• Fact:

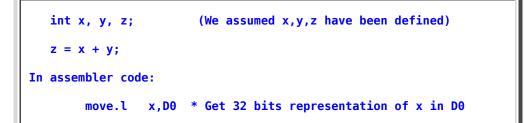


• ADD: add in a data register

• Instruction to add in a data register:

ADD.BSrc, Dn* Add the8 bits values: Dn = Dn + SrcADD.WSrc, Dn* Add the16 bits values: Dn = Dn + SrcADD.LSrc, Dn* Add the32 bits values: Dn = Dn + Src

• Example 1: adding integer variables



```
move.l y,Dl * Get 32 bits representation of y in Dl
add.l D0,Dl * 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
                28(A0),D1 * Get 32 bits representation of B[7] in D1
       move.l
        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 !!!
                            * Get 16 bits representation of B[3] in D0
        move.w
                  6(A0),D0
        move.w
                            * Get 16 bits representation of B[7] in D1
                 14(A0),D1
        add.w
                  D0,D1
                            * Add the two 16 bits representations together
                            * The sum is stored in D1 (16 bits)
                            * Store the 16 bits representation in B[4]
       move.w
                 D1,8(A0)
```

• ADDA: Adding in an address register

• Fact:

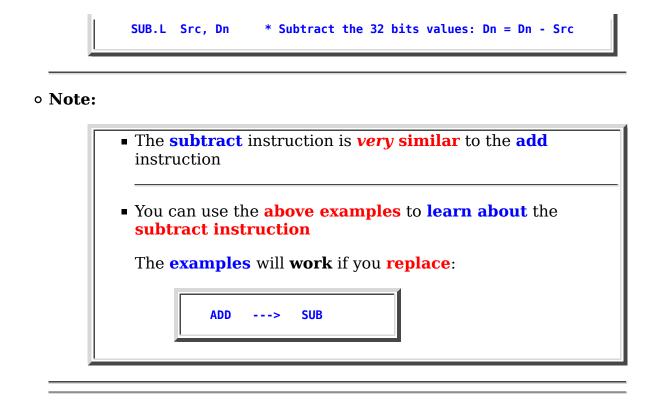
- When the destination of the add instruction is an address register, the instruction nmemonic gets an a appended to the tail
- When the destination is an address register, you can only use word size and long word size instructions

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

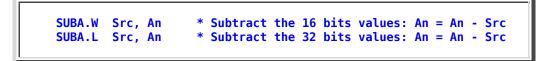
ADDA.W Src, An * Add the 16 bit value Src to the address reg An * I.e.: An = An + Src ADDA.L Sec, An * Add the 32 bit value Src to the address reg An * I.e.: An = An + Src

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

```
SUB.BSrc, Dn* Subtract the8 bits values: Dn = Dn - SrcSUB.WSrc, Dn* Subtract the16 bits values: Dn = Dn - Src
```



• Subtract in an address register:



• 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
                                                      -===== */
         _____
                        * A0 = base address of array B
      movea.l #B,A0
                        * 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
         _____
                                        =========================== */
               6(A0),D0 * Get 16 bits representation of B[3] in D0
      move.w
      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)
              D1,8(A0) * Store the 16 bits representation in B[4]
      move.w
```