A comprehensive example in Operand Manipulation

- Suppose we have the following variables defined:

```
int   x[100];
short y[100];
byte  z[100];

int   i;
short j;
byte  k;

class List
{
   int   value1;
   short value2;
   List next
}

List head;   (head contains the start of a linked list,
              assume the linked list has been created and is
              not empty)
```

- Write an equivalent assembler program for:

```
x[i + j] = y[i * k] + z[j / k] + head.value1 + head.next.value2;
```

- Steps needed to accomplish the statement:

1. Get `y[i * k]`
2. Get `z[j / k]`
3. Add them
4. Get `head.value1`
5. Add to sum
6. Get `head.next.value2`
7. Add to sum
8. Get the **address** of `x[i + j]`
9. Put the value of the sum computed previously in that address.

- The following is the solution, be very careful about the operand sizes !!!

(1) Get **y[i + k]**:

```
   MOVEA.L #y, A0          A0 = base address of array "y"
   MOVE.L  i, D0           D0 = i (32 bits)
   MOVE.B  k, D1           D1 = k (8 bits)

                           *** Can't add i + k yet !
   EXT.W   D1              D1 = k (16 bits)
[  EXT.L   D1              D1 = k (32  bits)     does not hurt...]
                           *** now we can multiply i * k !
   MULS    D1, D0          D0 = i * k (32 bits), index, NOT offset
```

```
      MULS   #2, D0            Because elements in array "y" are short

      MOVE.W 0(A0,D0.W), D7    D7 = y[i * k] (16 bits)

(2) Get z[j / k]:

      MOVEA.L #z, A0           A0 = base address of array "z"
      MOVE.W  j, D0            D0 = j (16 bits)
      MOVE.B  k, D1            D1 = k (8 bits)

                               *** Can't divide j / k yet !
      EXT.L   D0               Divident must be 32 bits
      EXT.W   D1               D1 = k (16 bits)
                               *** now we can divide j / k !

      DIVS    D1, D0           D0 = j / k (16 bits), index, NOT offset

      MULS   #1, D0            Because elements in array "z" are bytes
                               (You can omit this instruction....)

      MOVE.B 0(A0,D0.W), D6    D6 = z[j / k] (8 bits)

(3) Add them:
                               *** Can't D7 = y[i * k] (16 bits)
                               *** and D6 = z[j / k] (8 bits) yet !
                               *** because: WRONG SIZE !!!
      EXT.W   D6               D6 = z[j / k] (16 bits)
      ADD.W   D6, D7           D7 (16 bits) = y[i * k] + z[j / k]

              +++ NOTE: Do NOT use D7 in any computation !
              +++ You need it later !!!



(4) Get head.value1:

      MOVEA.L  head, A0        A0 points to the first element of list
      MOVE.L   (A0), D0        D0 contains the value head.value1

(5) Add to sum

      EXT.L    D7              D7 contains a word operand
                               We must convert it to long before adding
      ADD.L    D0, D7          D7 = y[i * k] + z[j / k] + head.value1

(6) Get head.next.value2:

      MOVEA.L  head, A0        A0 points to the first element of list
      MOVEA.L  6(A0), A0       A0 points to the second element of list
      MOVE.W   4(A0), D0       D0.w contains the value head.next.value2

(7) Add to sum

      EXT.L    D0              D0.w contains word size operand head.next.value2
                               Need to convert to long before adding
      ADD.L    D0, D7          D7 = y[i * k] + z[j / k] + head.value1 + head.next.value2


(8) Get the address of x[i + j]:

      MOVEA.L #x, A0           A0 = base address of array "x"
```

```
      MOVE.L  i, D0           D0 = i (32 bits)
      MOVE.W  j, D1           D1 = j (16 bits)

                              *** Can't add i + j yet !
      EXT.L   D1              D1 = j (32  bits)
                              *** now we can add i + j !
      ADD.L   D1, D0          D0 = i + j (32 bits), index, NOT offset

      MULS   #4, D0           Because elements in array "y" are short

                              *** Now 0(A0, D0.W) is the address of
                              *** x[i + j]

(9) Put the value of the sum computed previously in 0(A0, D0.W):

                              *** Can't do: MOVE.L D7, 0(A0, D0.W)
                              *** because:
                              *** D7 = y[i * k] + z[j / k] (16 bits)
                              *** and x[i + j] is 32 bits

      MOVE.L  D7, 0(A0, D0.W)
```

DONE... finally...

Now take a look back at the mess we made just to do the simple looking addition "`x[i + j] = y[i * k] + z[j / k] + head.value1 + head.next.value2`"..... Almost a **FULL PAGE** of assembler code...