M68000 (Address Register) Indirect with Index and Displacement Mode (a.k.a. the *indexed* mode)

- **Recall** the **syntax** of the **MOVE instruction**:

```
MOVE <EA>,<EA>              * Copy Src to Destination
      ^    ^
      |    |
      |    +--- Destination
      +-------- Source operand
```

- **The indexed mode**
  - **Syntax** to specified the **indexed** mode:

```
   x(An, Dm)          where An = A0, A1, A2, A3, A4, A5, A6, or A7
                             Dm = D0, D1, D2, D3, D4, D5, D6, or D7
                             x  = a number between -128 and 127

Examples:

   0(A1,D5)
  -8(A3,D7)
   4(A5,D4)
```

  - **Semantics (meaning):**
    - The **operand** specified by **x(An, Dm)** is **located** in ***memory*** at the **location (address)** given by the value of the expression

```
x + An (32 bits) + Dn (32 bits)
```

      The **address** used by the **indexed mode** is equal to:

      - **x** + the **32 bit value** in **register An** + the **32 bit value** in **register Dm**.

Note:

- The **registers An** and **Dm** used in the **addressing mode** will ***not* be altered**

- The **sum**:

```
x + An (32 bits) + Dn (32 bits)
```

is stored in a ***special*** **Memory Address Register (MAR)** and used to **access** the **memory**

- **Example:**

```
MOVEA.L #1000,A1              (set up address register)
MOVE.L  #4000,D4              (set up data register)

MOVE.L  #34, 4(A1, D4), D0     will store 34 (as binary number)
                               in memory location at address 5004
                               Beecause: A0 contains 1000, D0 = 4000,
                               So:       1000 + 4000 + 4 = 5004
```

- **Didactical comment**
  - Some demo programs may use:

```
x(An, Dm.W)
```

Then I am using the 16 bit number inside the register `Dm`

(That's because in my demo program, I used a very small value that can be represented by **16 bits**)

- **Most common application of the *index* mode**
  - **Most common situation** to use the **index mode**:
    - **Accessing** an **array variable** such as: `A[i]`, `A[i+j]`, etc.

- **Example 1:** accessing elements in an `int` array **B**

```
High level programming language:

       int B[10];
       int k;          // k has been initialized

       B[k] = 34;
```

```
Assembler code:

     MOVEA.L #B,A0           * A0 = base address of array B

     MOVE.L  k, D0           * D0 = k
                             * k is the index in the array
```

```
                              * You need to multiply by the size
                              * of each array element to get
                              * the offset !

      MULS #4, D0             * This instruction multiply D0 by 4
                              * and store the result in D0
                              * --- Now D0 contains the offset of B[k]
                              * --- from the base address of array B

      MOVE.L #34,0(A0,D0)     * Move 34 into element B[k]

How to defined the array B and k::
   B:  DS.L 10            An integer array: int B[10]
   k:  DS.L 1            An integer variable: int k
                         (used as index into array A)
```

- **Example Program:** (Demo above code) *Example*
  - Prog file: click here

How to run the program:

- **Right click** on link(s) and **save** in a scratch directory
- To compile: `as255 indexed`
- To run, use: `m68000`

- **Example 2:** accessing elements in a `short` array **B**

```
High level programming language:

       short B[10];
       int k;          // k has been initialized

       B[k] = 34;
```

```
Assembler code:

      MOVEA.L #B,A0           * A0 = base address of array B
                              * The address of any array type is 32 bits !!

      MOVE.L  k, D0           * D0 = k
                              * k is the index in the array
                              * You need to multiply by the size
                              * of each array element to get
                              * the offset !

      MULS #2, D0             * D0 = 2*D0
                              * (because a short occupies 2 bytes memory)
                              * --- Now D0 contains the offset of B[k]
                              * --- from the base address of array B

      MOVE.W #34,0(A0,D0)     * Move 34 into element B[k]
                              * We need to use .W because B[k] is a short
```

```
How to defined the array B and k:
   B:  DS.W 10          A shirt array: short B[10]
   k:  DS.L 1           An integer variable: int k
                       (used as index into array A)
```

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

  - Prog file: click here

How to run the program:

- **Right click** on link(s) and **save** in a scratch directory
- To compile: `as255 indexed1`
- To run, use: `m68000`

- **Quiz**: **why** is the following **program *not* correct** ???

```
High level programming language:

       short B[10];
       int k;          // k has been initialized

       B[k] = 34;
```

```
Assembler code:

     MOVEA.L #B,A0          * A0 = base address of array B
                            * The address of any array type is 32 bits !!

     MOVE.L  k, D0          * D0 = k
                            * k is the index in the array
                            * You need to multiply by the size
                            * of each array element to get
                            * the offset !

     MULS #2, D0            * D0 = 2*D0
                            * (because a short occupies 2 bytes memory)
                            * --- Now D0 contains the offset of B[k]
                            * --- from the base address of array B

     MOVE.L #34,0(A0,D0)    * Error !!!
                            * Can you predict what this instruction will do ???
```

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

  - Prog file: click here

How to run the program:

- **Right click** on link(s) and **save** in a scratch directory
- To compile: `as255 indexed2`
- To run, use: `m68000`
