
M68000 Assembler Directives

- M68000 Assembler Directives:
 - ORG (Organize)
 - EQU (Equal)
 - DS (Define Storage)
 - DC (Define Constant) - Note: a misnomer
 - EVEN
 - END (Exit)
-

- **ORG** (Organize)

- Syntax:

ORG *address*

- Effect: tell assembler to translate the next assembler instruction at memory location *address*
- The use of **ORG** is **obsoleted** by the use of the "**virtual memory**" **technique** that allow **multiple programs** to **start** at **address 0**

(**Virtual memory** will be discussed in **CS355**)

- **EQU** (Equal)

- Syntax:

Label **EQU** *constant*

- Effect: tell assembler to set the symbolic name *label* equal to the value of the constant *constant*
- *Label*
 - Consists of letter, digits and `_` (underscore)
 - Must begin with a letter
 - First character of *Label* must be at column 1 or *Label* must be followed by colon (`:`)

- Examples:

```

123456789 (<---- column position in file)

MAX     EQU 100           - good EQU directive
  MAX:  EQU 100           - good EQU directive
MAX:    EQU 100           - good EQU directive
  MAX   EQU 100           - BAD EQU directive

```

- Any one of the good EQU directive above defines the symbolic name `MAX` to be equal to 100. This capability to define symbolic constant can also be found in high level languages, like:

```

JAVA:   final int MAX = 100;
C:      #define MAX 100

```

- The assembler (is a computer program) has a data structure named **symbol table** used to map symbolic names to their defined values.

- Example, get this program, compile it and look at the output listing file `a.lst`: [click here](#)

The output listing looks like this:

```

MAX      EQU      100
MIN      EQU      10

          SYMBOL TABLE
          *****
MAX = 000064      MIN = 00000A      (Hint: hexadecimal numbers)

```

The output listing indicate that the assembler has recorded the following 2 symbolic constant (that the assembler program can use):

- MAX which is equal to $64_{(16)}$ or $6 \times 16 + 4 = 100_{(10)}$
 - MIN which is equal to $A_{(16)}$ or $10_{(10)}$
- You can define constant in other number systems as follows:
 - `$`-prefix indicates a hexadecimal constant, e.g.: `$FA1F`
 - `@`-prefix indicates a octal constant, e.g.: `@70167`
 - `%`-prefix indicates a binary constant, e.g.: `%11011011`

• DS (Define Storage)

- Syntax:

```

(1) Label DS.B n - reserves n bytes of memory space
(2) Label DS.W n - reserves n words of memory space
(3) Label DS.L n - reserves n long words of memory space

```

- Effect: tell assembler to

- **reserve** space to store `n` bytes, words or long words (depending on the size following the `DS` directive)
- **set** the symbolic name `Label` equal (like `EQU`) to the address of the first memory location of the reserved space

- The `DS` directive is used to define **uninitialized static** variables (static variables have lifetime from start of program till end)

- Check out the effect of `DS` in this demo program: [click here](#).

Assemble the program with "`as255 ds`" and take a look at the listing file "`a.lst`":

`a.lst`:

```

1 000000      * Demonstrate the effect of DS directive
2 000000      * Assemble with: as255 ds
3 000000      * Look at the output file a.lst
4 001000      ORG $1000
5 001000 1200      move.b d0, d1      (1200 (hex) is the code
6 001002 1200      move.b d0, d1      for "move.b d0, d1" !)
7 001004      A: ds.b 10
8 00100E 1200      move.b d0, d1
9 001010 1200      move.b d0, d1
10 001012      B: ds.w 10
11 001026 1200      move.b d0, d1
12 001028 1200      move.b d0, d1
13 00102A      C: ds.l 10
14 001052 1200      move.b d0, d1
15 001054 1200      move.b d0, d1
16 001056      end

```

SYMBOL TABLE

A	001004	B	001012	C	00102A
---	--------	---	--------	---	--------

- Notice that: (use calctool !)

100E(hex)	-	1004(hex)	=	10(decimal)	10 bytes reserved at A
1026(hex)	-	1012(hex)	=	20(decimal)	20 bytes reserved at B
1052(hex)	-	102A(hex)	=	40(decimal)	40 bytes reserved at C

- Examples of usage:

Construct in Java/C/C++	Equivalent in M68000
-----	-----
int i;	i: DS.L 1
short s;	s: DS.W 1
byte b;	b: DS.B 1

Construct in C/C++	Equivalent in M68000
-----	-----
int A[10];	A: DS.L 10
short B[100];	B: DS.W 100

Construct in C/C++	Equivalent in M68000
-----	-----
class MyClass	Remembers:
{	1. a MyClass object occupies 10 bytes
int x;	2. x's offset is 0
int y;	3. y's offset is 4
short z;	4. z's offset is 8
};	
MyClass A;	A: DS.B 10

Note: A marks the start of the memory location for the object
 You need to add the offset to A to get to the members
 x, y and z. That's why assembler remembers the offset
 for each member variable in an object

- DC (Define Constant)

- Syntax:

- (1) Label **DC.B** <list of constant values (byte size)>
- (2) Label **DC.W** <list of constant values (word size)>
- (3) Label **DC.L** <list of constant values (long word size)>

- Effect: tell assembler to

- **reserve** space to store the list of constant values in memory,
- **initialize** the reserved space with the specified values
- **set** the symbolic name `Label` equal (like EQU) to the address of the first memory location of the reserved space

- The DC directive is used to define **initialized static** variables

- **Important program note:**

- The **assembler** on the **Linux machines (in the lab)** **do not** allow **spaces** between the **values** in the **list of constants**

Example:

```
A: dc.b 10, 1, 2, 3      BAD !!!
```

You will get **warning:**

```
filename.s:xx: Warning: zero assumed for missing expression
```

Note:

- My **teaching notes** on this **website** are based on the **assembler** on **Solaris** that **does allow** a **space** between **constants**.

- So you **may** see some **space** in my **webpages**
 - But my **test programs** that run on a **lab machine** will **not** have the **space** (I removed them).

- Check out the effect of DS in this demo program: [click here](#).
Assemble the program with "as255 dc" and take a look at the listing file "a.lst":

a.lst:

```

1 000000      * Demonstrate the effect of DC directive
2 000000      * Assemble with: as255 dc
3 000000      * Look at the output file a.lst
4 001000                ORG $1000
5 001000 1200                move.b d0, d1
6 001002 1200                move.b d0, d1
7 001004 0A  A:      dc.b 10, 1, 2, 3
                        01
                        02
                        03
8 001008 1200                move.b d0, d1
9 00100A 1200                move.b d0, d1
10 00100C 000A  B:      dc.w 10, 1, 2, 3
                        0001
                        0002
                        0003
11 001014 1200                move.b d0, d1
12 001016 1200                move.b d0, d1
13 001018 0000  C:      dc.l 10, 1, 2, 3
                        000A
                        0000
                        0001
                        0000
                        0002
                        0000
                        0003
14 001028 1200                move.b d0, d1
15 00102A 1200                move.b d0, d1
16 00102C                end

```

SYMBOL TABLE

A 001004

B 00100C

C 001018

- Notice that: (use calctool !)

```
1008(hex) - 1004(hex) = 4(decimal)    4 bytes reserved at A
1014(hex) - 100C(hex) = 8(decimal)    8 bytes reserved at B
1028(hex) - 1018(hex) = 16(decimal)   16 bytes reserved at C
```

- EVEN

- **EVEN** tells the **assembler** to **skip forward** until it reaches an **even valued address**

- This **directive** is **needed** due to the **alignment requirement** imposed by the **computer manufacturer** on **how** certain **types** of **variables** are **stored**

- **Alignment requirements** in the **M68000**:

- A **short typed** variable (= 2 bytes in length) must be **located** at an **even address**
- A **int typed** variable (= 4 bytes in length) must be **located** at an **even address**

(A **byte typed** variable (= 1 byte in length) must be **located** *anywhere* in memory)

- Using the **even** directive:

- **Problem:**

- Define a **int typed** variable named **x**

- This is **not completely correct**:

```
x: ds.1 1      * reserve 4 bytes for the int variable
```

Reason:

- The **variable x** **may not** be located on an **even valued address**
(Remember the **alignment requirement** of **int typed** variables (see above))

- **Correct solution:**

```
even      * Tell assembler to skip to an even address
x: ds.1 1  * Now we are sure that x is stored at an even address
```

- **END**

- End the assembling process
 - Any text following the **end** directive will be ignored...
-
-
-

- **Final comment**

- **Make sure that you:**

- **Do not** put any **directive** at the **start** of a **line**

- **Because** the **assembler** will **consider** the **directive** as a **label** !!!
