Assembly-Language Statements

- Assembly language is made up of two types of statements
  - Executable Statements
    - Valid processor instructions
    - Translated into machine language by the assembler
  - Assembler Directives
    - Are not instructions to the processor
    - Cannot be translated into machine language
    - Tell the assembler what it needs to know about the program and its environment

68000 Program Structure

```
*Our first assembly-language program
*Notice that full-line comments start with an '*' in the 1st column
```

Label Field

```
CR          EQU         %1101           ;ASCII carriage return
LF          EQU         %1010           ;ASCII line-feed
            ORG         $8000           ;program entry point
START       MOVE.W      #10,D6          ;repeat 10x
LOOP        MOVEA.W     #MESSAGE,A1     ;A1 points to message
            TRAP        #2              ;send message
            SUB         #1,D6           ;decrement count
            BNE         LOOP            ;if not 0, do it again
            TRAP        #14             ;return to MON68K
MESSAGE     DC.B        'Hello World'
            DC.B        CR,LF
            END         START
```

1 **Columns** 80

68000 Program Structure

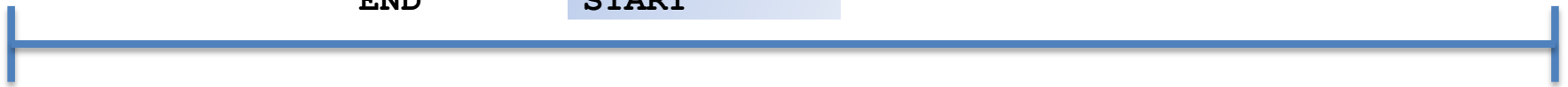
```
*Our first assembly-language program
*Notice that full-line comments start with an '*' in the 1st column

                MNEMONIC
CR              EQU     %1101           ;ASCII carriage return
LF              EQU     %1010           ;ASCII line-feed
                ORG     $8000           ;program entry point
START           MOVE.W  #10,D6          ;repeat 10x
LOOP            MOVEA.W #MESSAGE,A1     ;A1 points to message
                TRAP    #2              ;send message
                SUB     #1,D6           ;decrement count
                BNE     LOOP            ;if not 0, do it again
                TRAP    #14             ;return to MON68K
MESSAGE         DC.B    'Hello World'
                DC.B    CR,LF
                END     START
```

68000 Program Structure

```
*Our first assembly-language program
*Notice that full-line comments start with an '*' in the 1st column

                        OPERAND
CR          EQU         %1101       ;ASCII carriage return
LF          EQU         %1010       ;ASCII line-feed
            ORG         $8000       ;program entry point
START       MOVE.W      #10,D6      ;repeat 10x
LOOP        MOVEA.W     #MESSAGE,A1 ;A1 points to message
            TRAP        #2          ;send message
            SUB         #1,D6       ;decrement count
            BNE         LOOP        ;if not 0, do it again
            TRAP        #14         ;return to MON68K
MESSAGE     DC.B        'Hello World'
            DC.B        CR,LF
            END         START
```



68000 Program Structure

```
*Our first assembly-language program
*Notice that full-line comments start with an '*' in the 1st column

CR          EQU       %1101          ;ASCII carriage return
LF          EQU       %1010          ;ASCII line-feed
            ORG       $8000          ;program entry point
START       MOVE.W    #10,D6         ;repeat 10x
LOOP        MOVEA.W   #MESSAGE,A1    ;A1 points to message
            TRAP      #2             ;send message
            SUB       #1,D6          ;decrement count
            BNE       LOOP           ;if not 0, do it again
            TRAP      #14            ;return to MON68K
MESSAGE     DC.B      'Hello World'
            DC.B      CR,LF
            END       START
```

COMMENT

1 **Columns** 80

Today

- We will consider the following aspects of the 68000's ISA
  - Assembly Language
    - Assembly-language statements
    - Program structure
    - Assembler Directives
  - Assembler directives and C
  - Assembler files

The Most Important Assembler Directives

Directive	Operation	Syntax
ORG	Set value (address) of location counter	`ORG address`
END	End of source program	`END label`
EQU	Equate value to symbol	`symbol EQU value`
DS	Reserve space in RAM for data generated at runtime	`<label> DS count`
DC	Define data in RAM at the current location	`<label> DC item, <item> ...`

Equate Directive

- Links a name to a value, making programs easier to read
  - Format:
    - `<name> EQU <value>` (Does NOT reserve space in RAM)

Example 1

```
sunday      EQU   1
monday      EQU   2
...
saturday    EQU   7
```

```
move.b   #sunday,d0
```

Equate Directive

- Links a name to a value, making programs easier to read
  - Format:
    - `<name> EQU <value>` (Does NOT reserve space in RAM)

Example 2

```
sunday     EQU   1
monday     EQU   sunday + 1
...
saturday   EQU   sunday + 6
```

```
move.b   #monday,d1
```

The RHS can contain an assemble-time expression, as long as all of the symbols have been pre-defined

Origin Directive

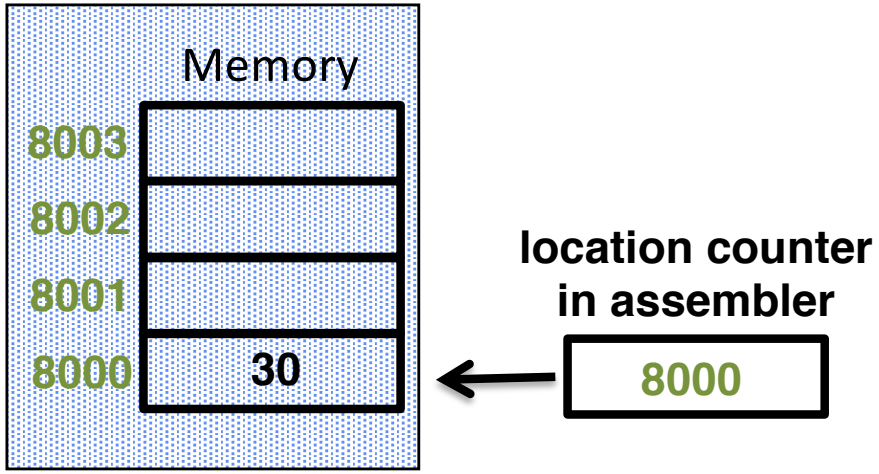
- Sets up the value of the location counter and keeps track of where the next item is to be located in the processor's memory
  - Format:
    - `ORG <address>` (location counter = address)

```
code    equ     $8000

        org     code
        move    #100,d0
```

30 3C 00 64

machine language



Origin Directive

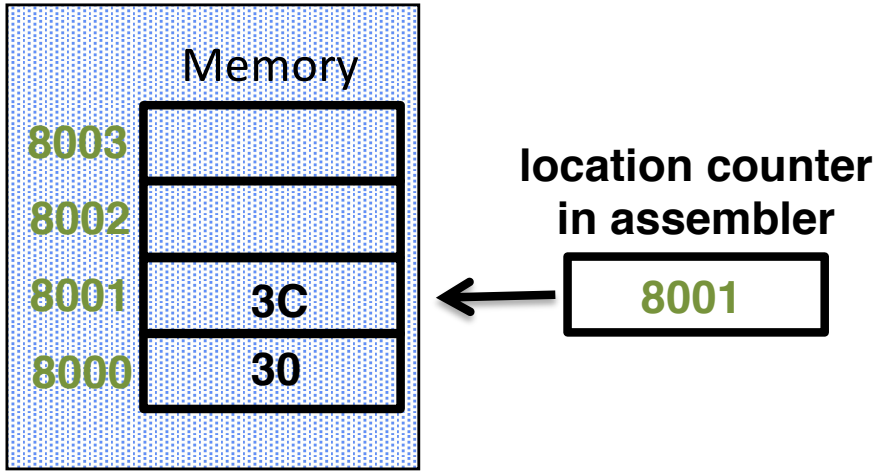
- Sets up the value of the location counter and keeps track of where the next item is to be located in the processor's memory
  - Format:
    - `ORG <value>` (location counter = value or address)

```
code    equ     $8000

        org     code
        move    #100,d0
```

30 3C 00 64

machine language



Origin Directive

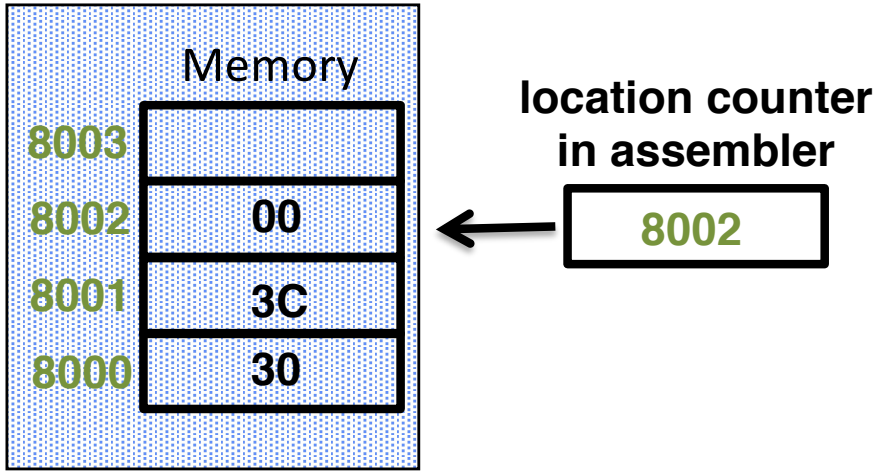
- Sets up the value of the location counter and keeps track of where the next item is to be located in the processor's memory
  - Format:
    - `ORG <value>` (location counter = value or address)

```
code    equ     $8000

        org     code
        move    #100,d0
```

30 3C 00 64

machine language



Origin Directive

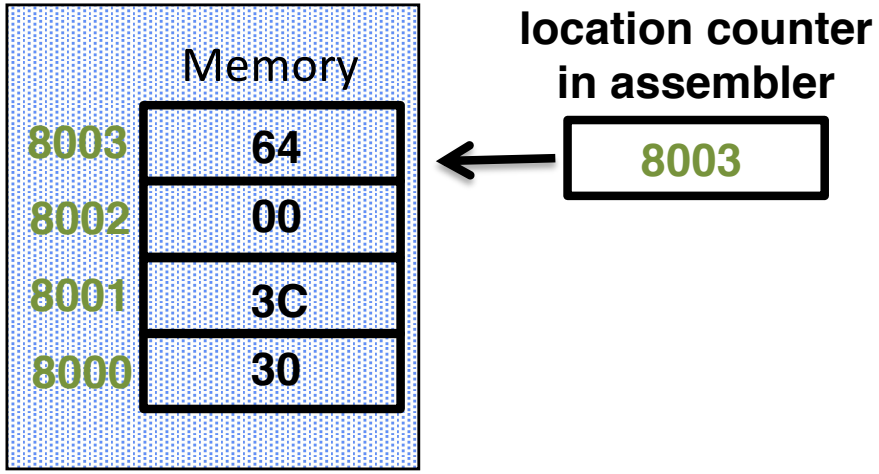
- Sets up the value of the location counter and keeps track of where the next item is to be located in the processor's memory
  - Format:
    - `ORG <value>` (location counter = value or address)

```
code    equ     $8000

        org     code
        move    #100,d0
```

30 3C 00 64

machine language



Origin Directive

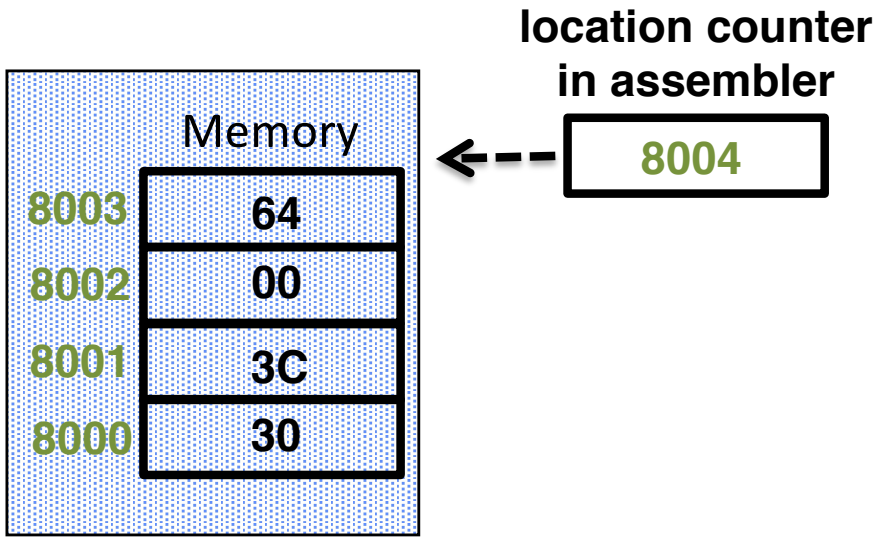
- Sets up the value of the location counter and keeps track of where the next item is to be located in the processor's memory
  - Format:
    - `ORG <value>` (location counter = value or address)

```
code    equ     $8000

        org     code
        move    #100,d0
```

30 3C 00 64

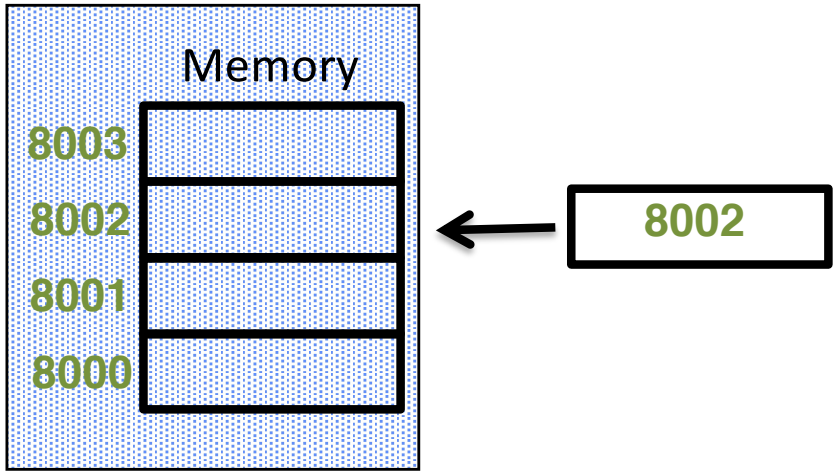
machine language



Alignment and the Location Counter

```
code    equ     $8001

        org     code
        move    #100,d0
```



- Other things:
  - An ORG directive can be located at any point in the program
  - The M68000 ISA reserves the first 1024 bytes of memory
  - RAM starts at hexadecimal $8000 on the 68K mini-board

Define Constant Directive

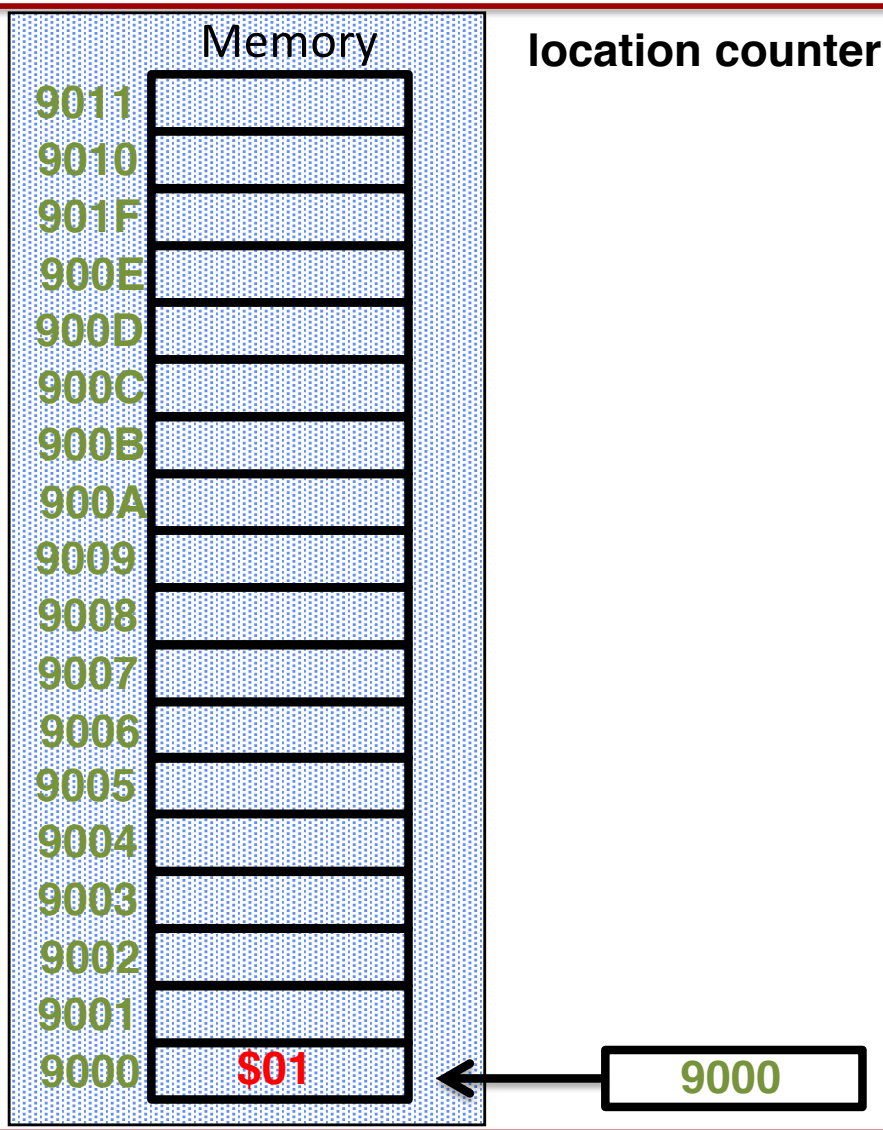
- The define constant directive allows you to place one or more data items in memory when the program is first loaded
  - Format:
    - `<label>  DC.<size>  <item>, <item> …`
  - The (optional) **`label`** will be defined to equal to the address of the start of the list of items
  - The **`size`** specifies that a list of bytes (.B), words (.W), or long-words (.L) is being defined
  - Each **`item`** may be a decimal number, hexadecimal number ($), a binary (%) number, an ASCII string enclosed in single quotes, or an assemble-time expression

Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000

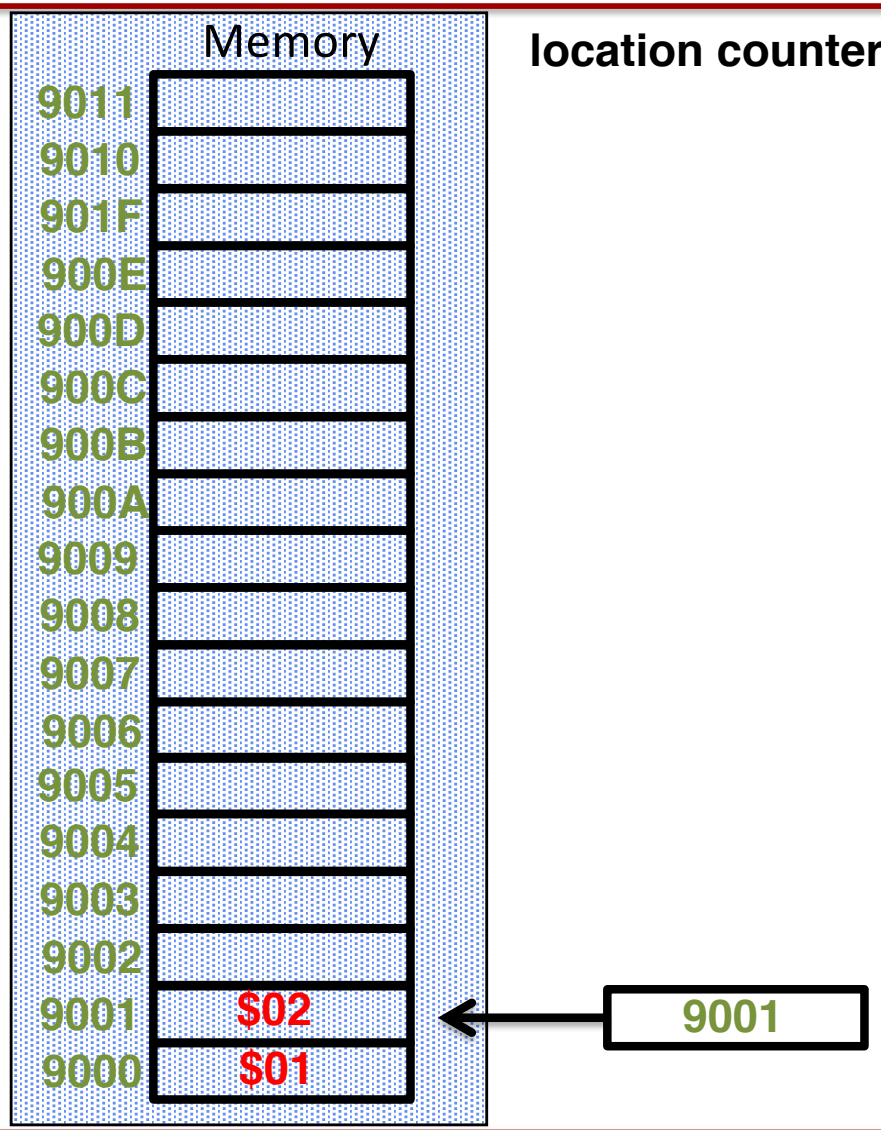


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000

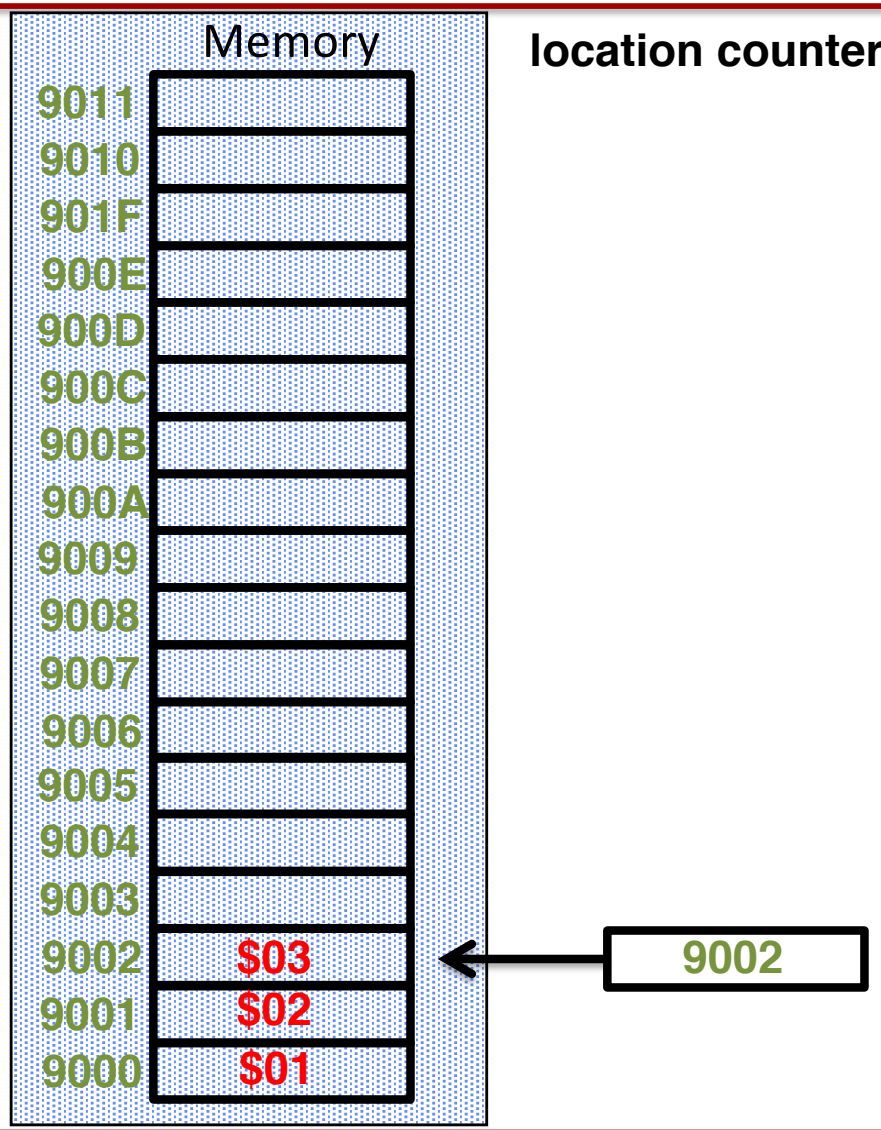


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000

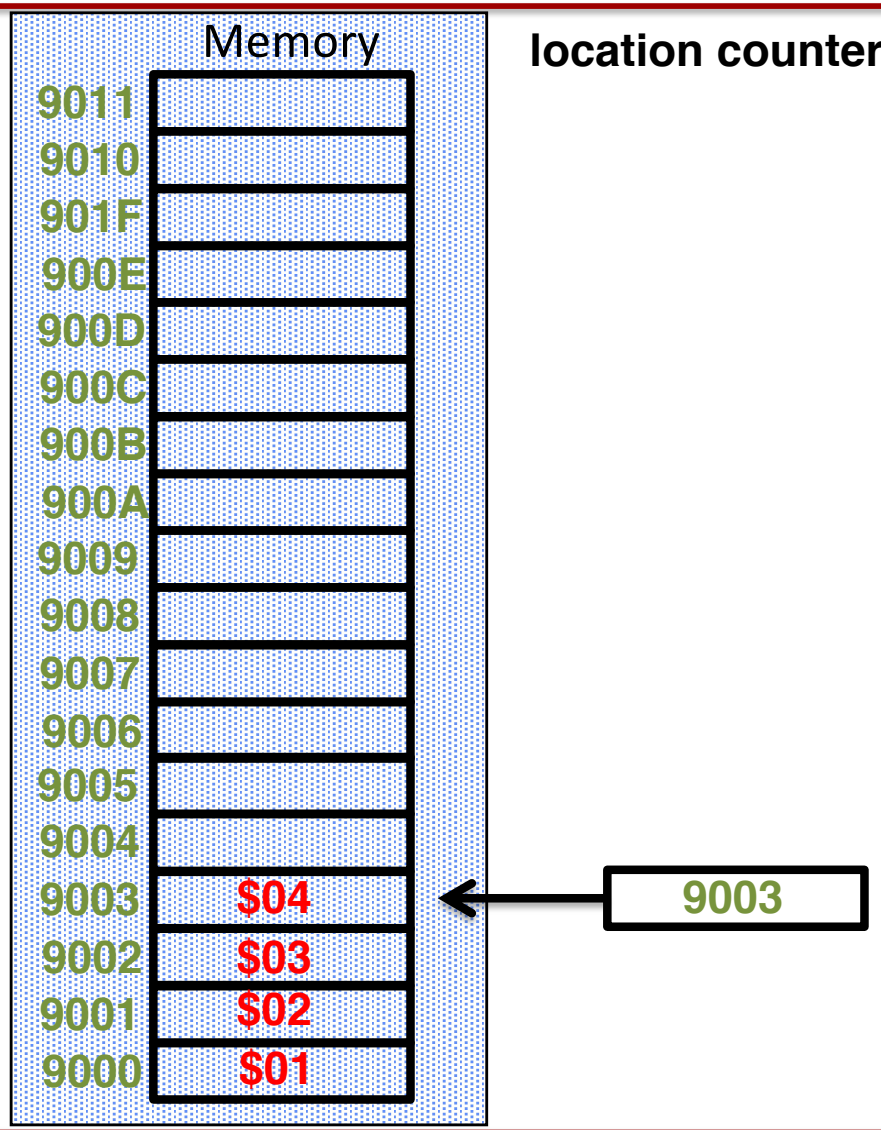


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000

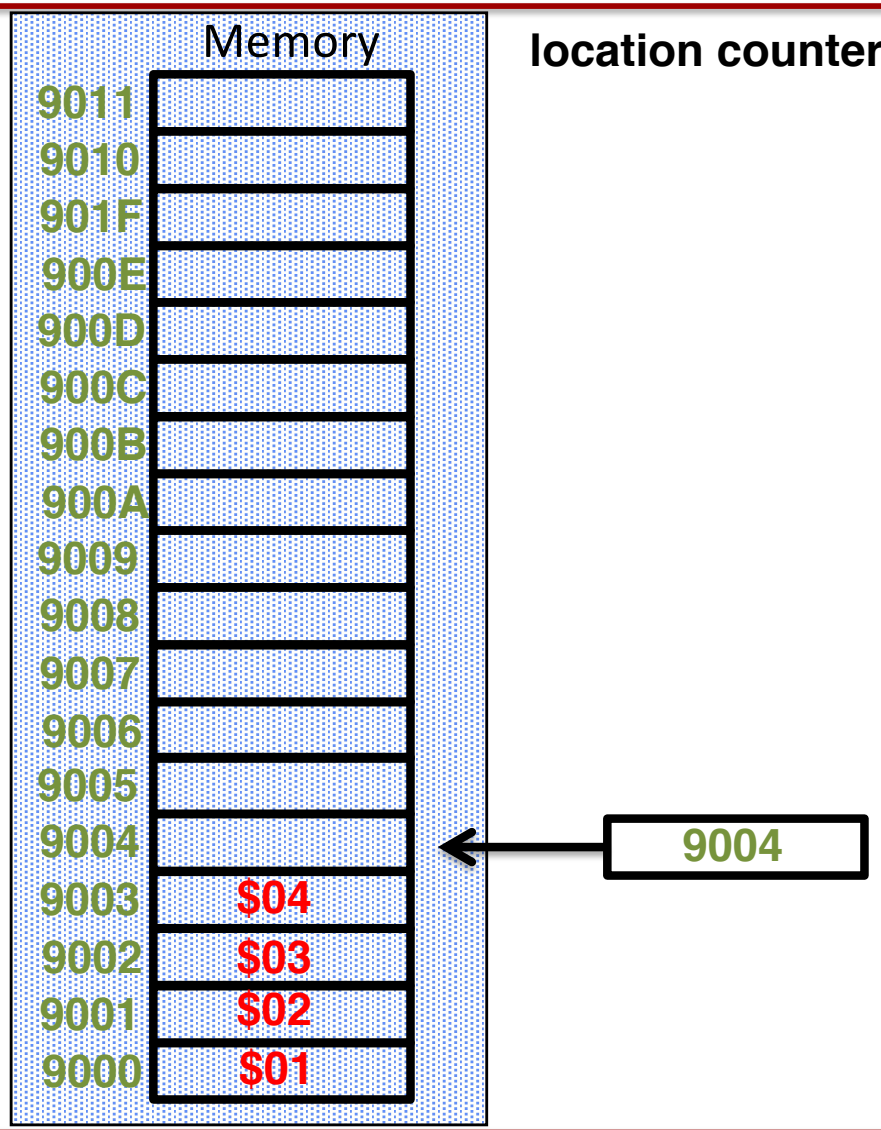


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000

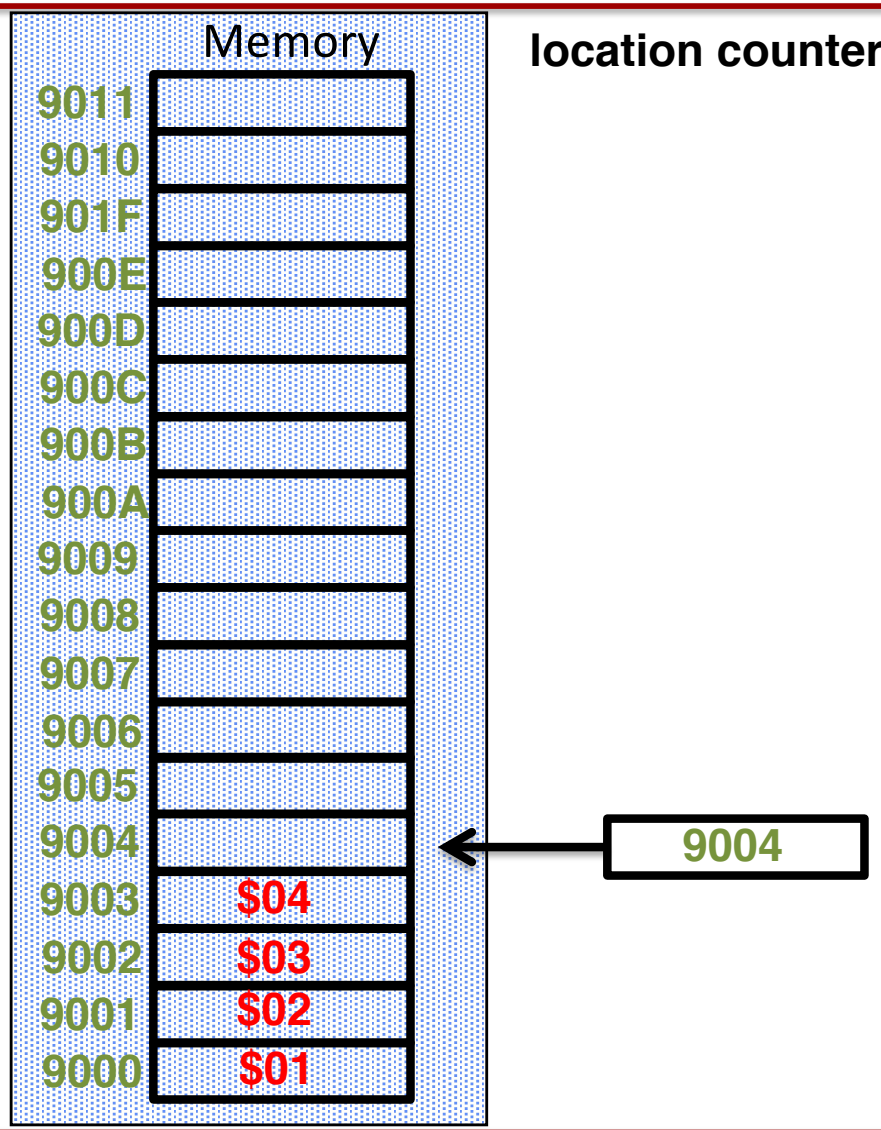


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004

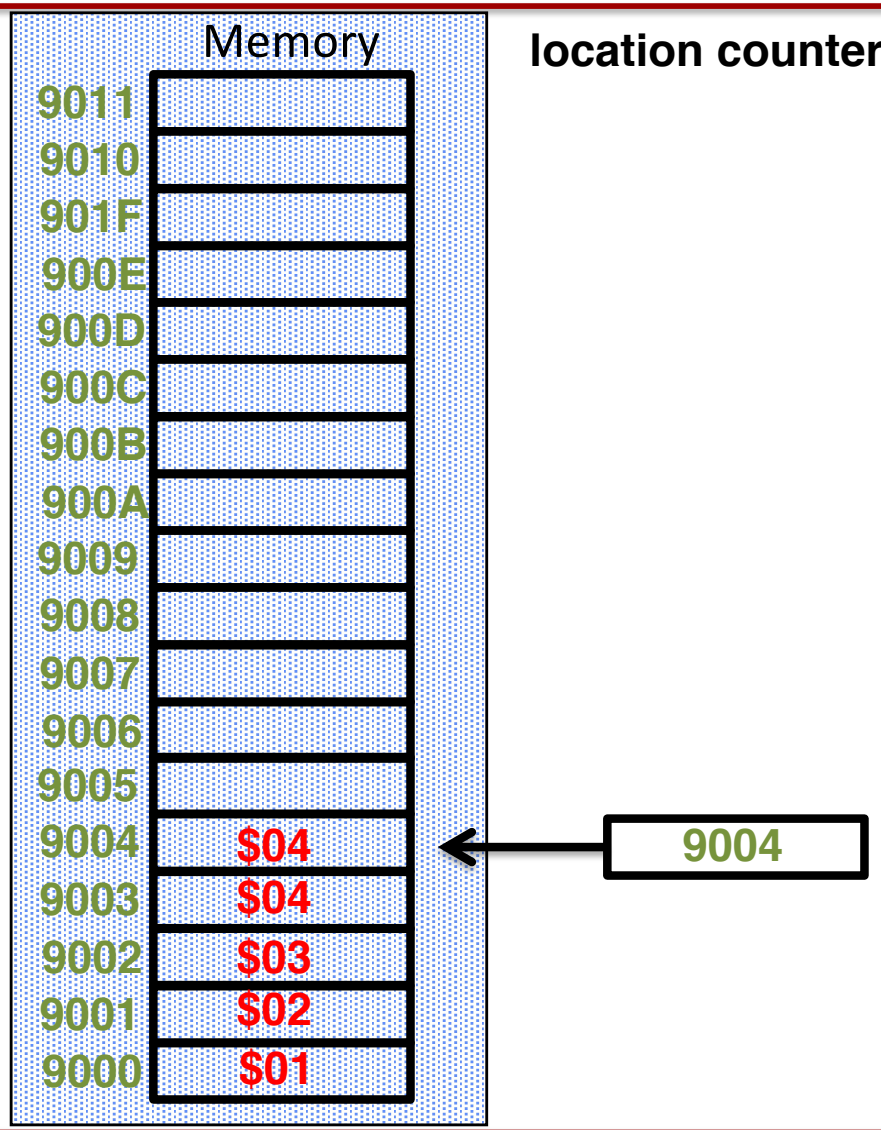


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004

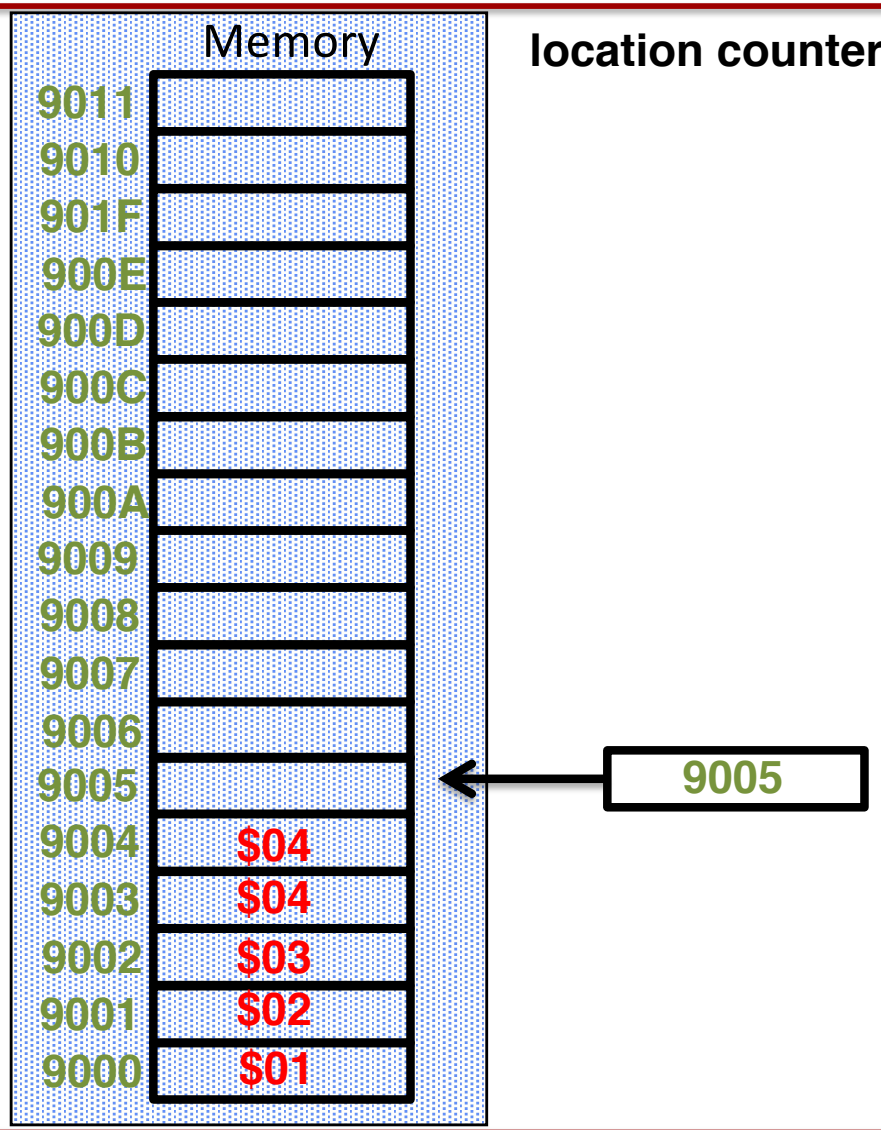


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004

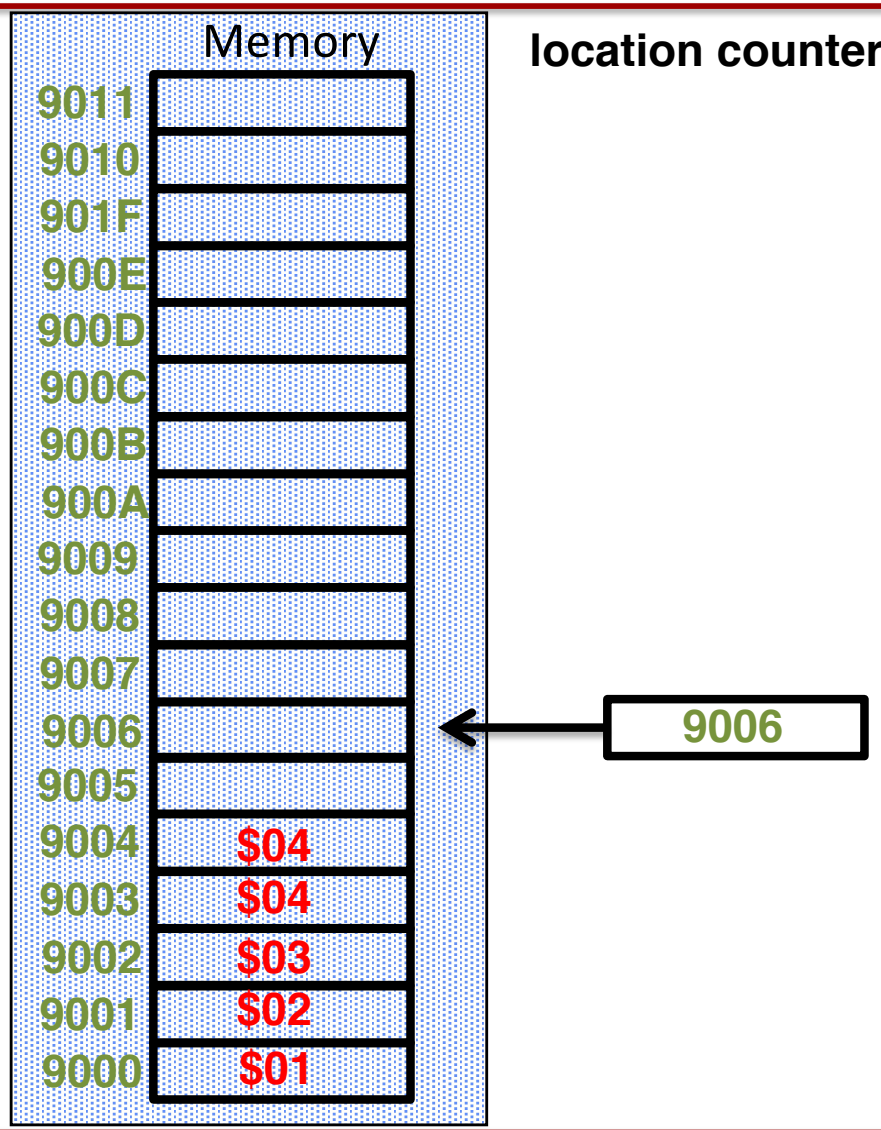


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006

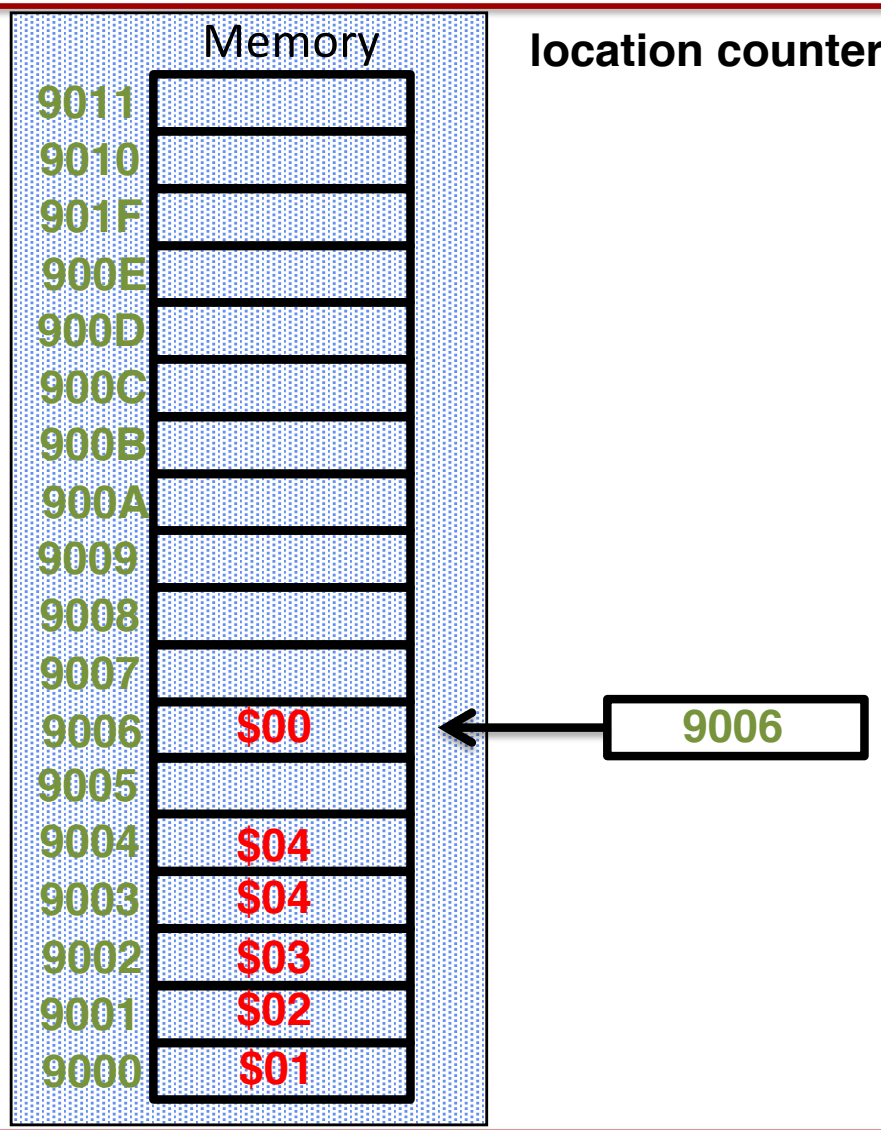


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006

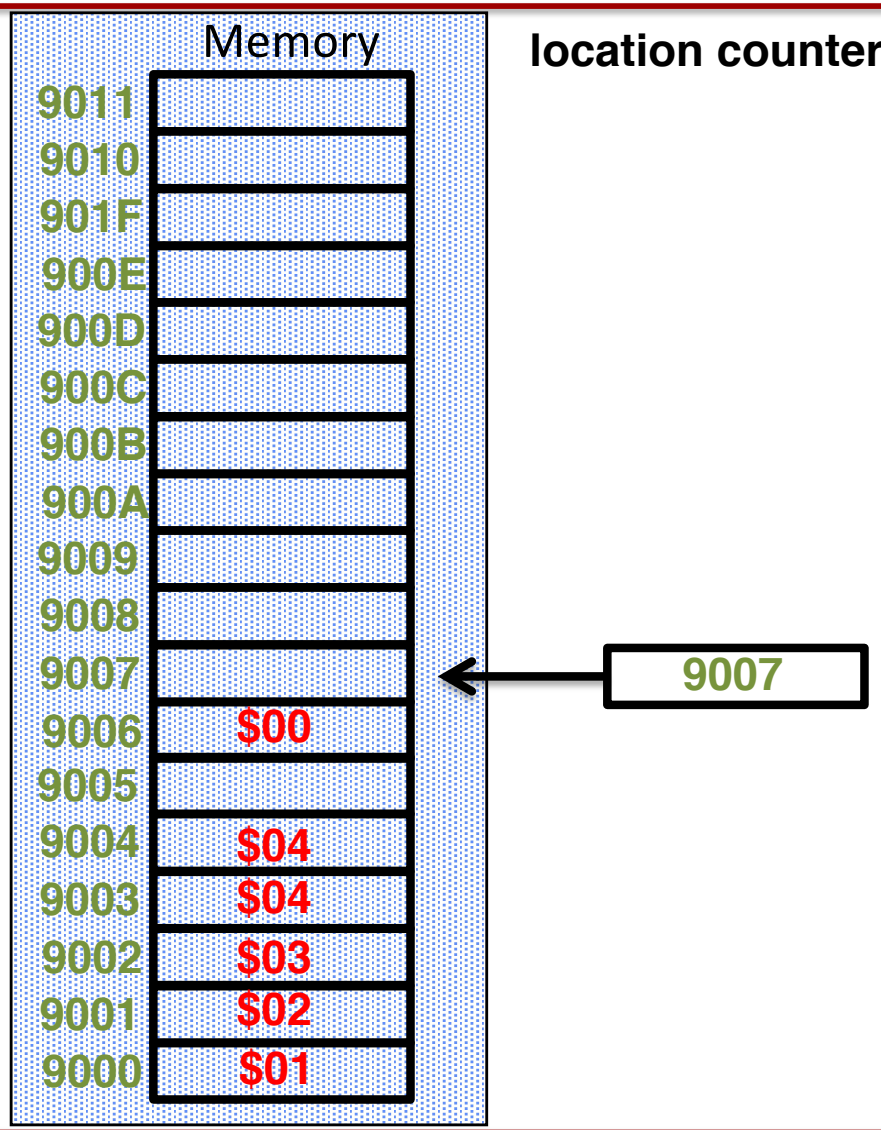


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006

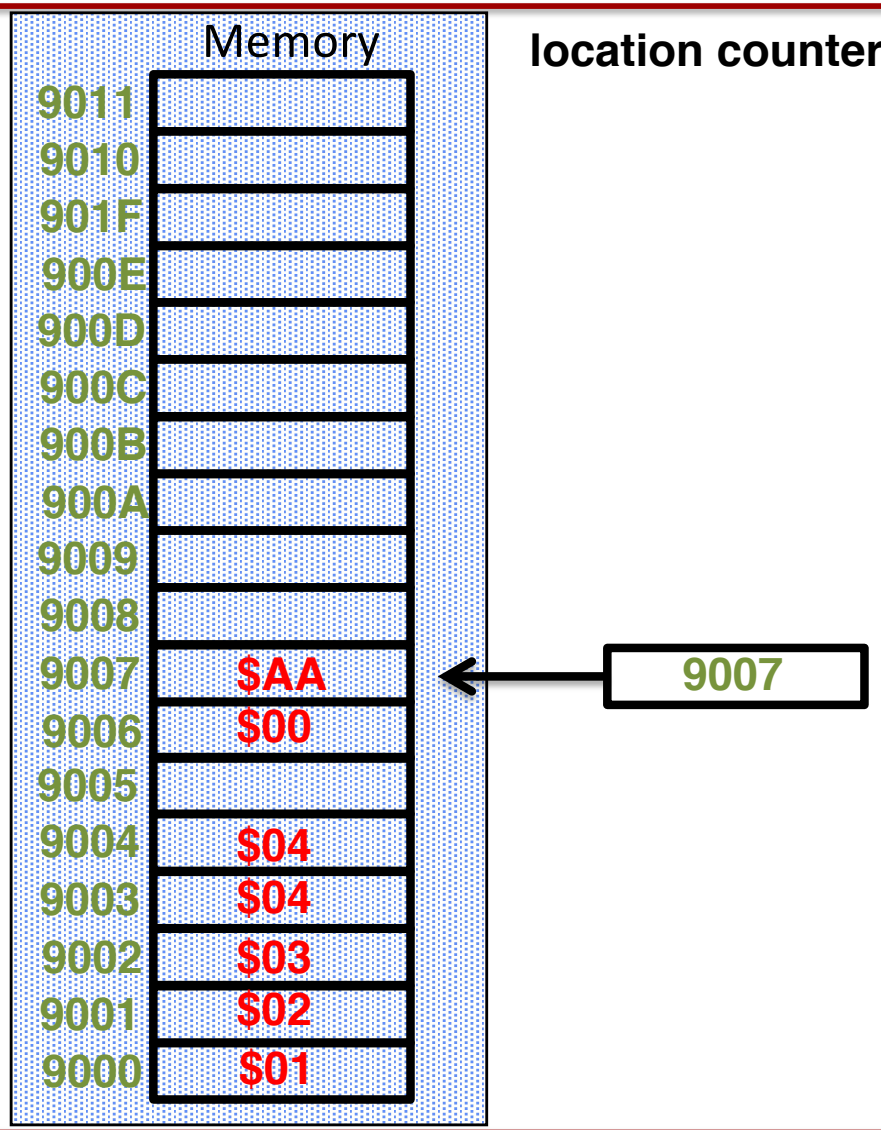


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006

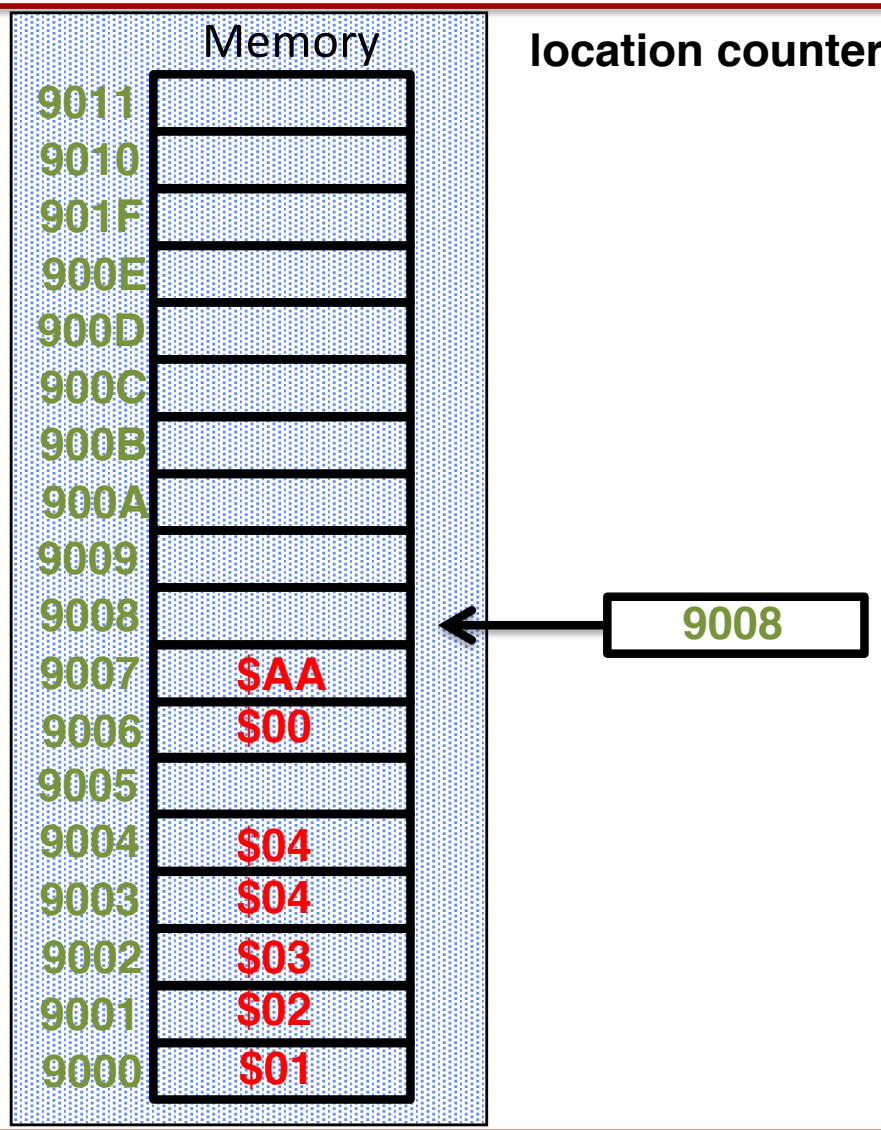


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008

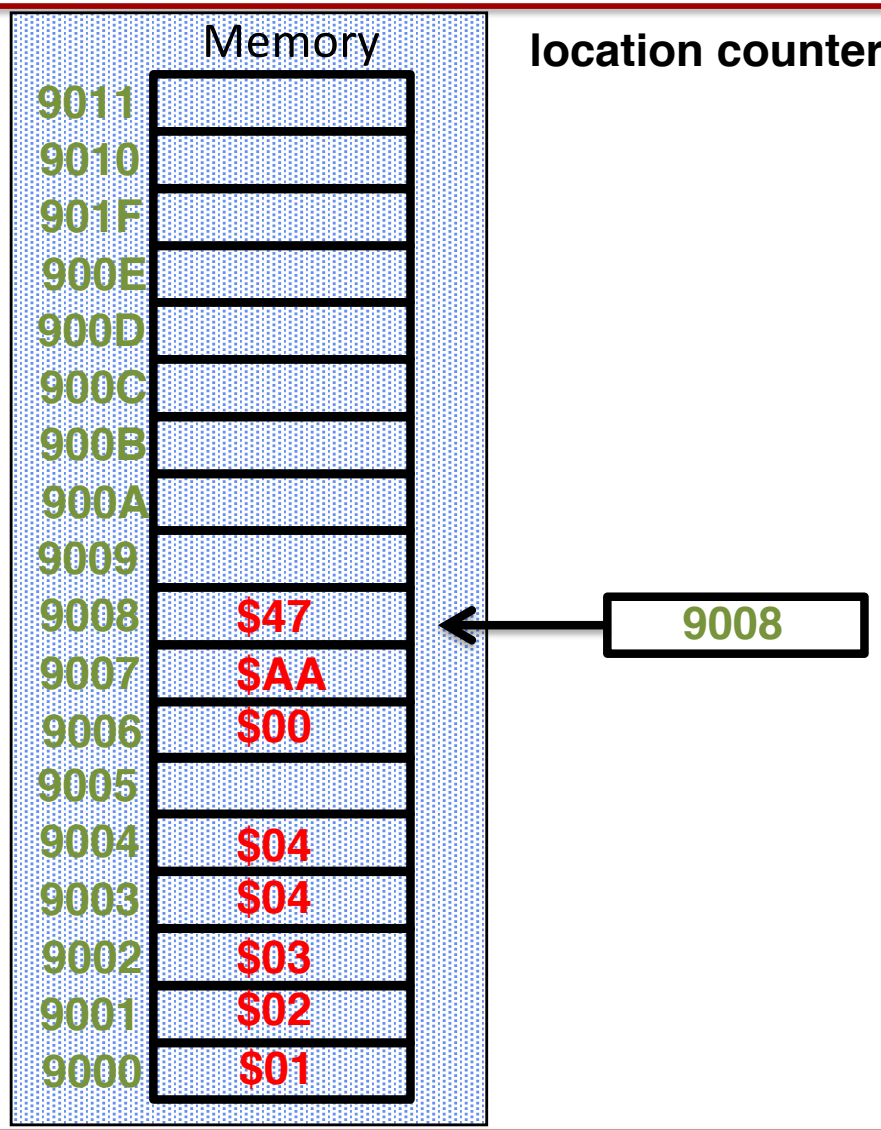


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008

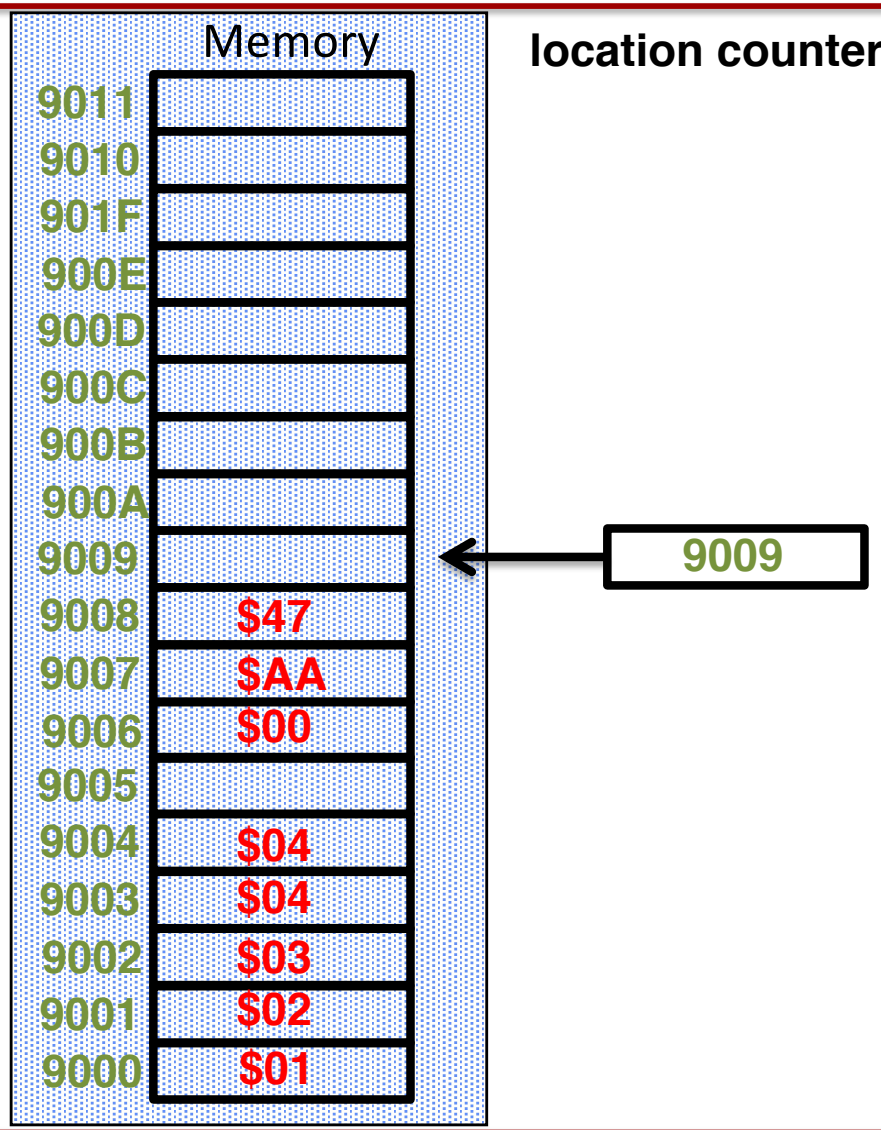


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008

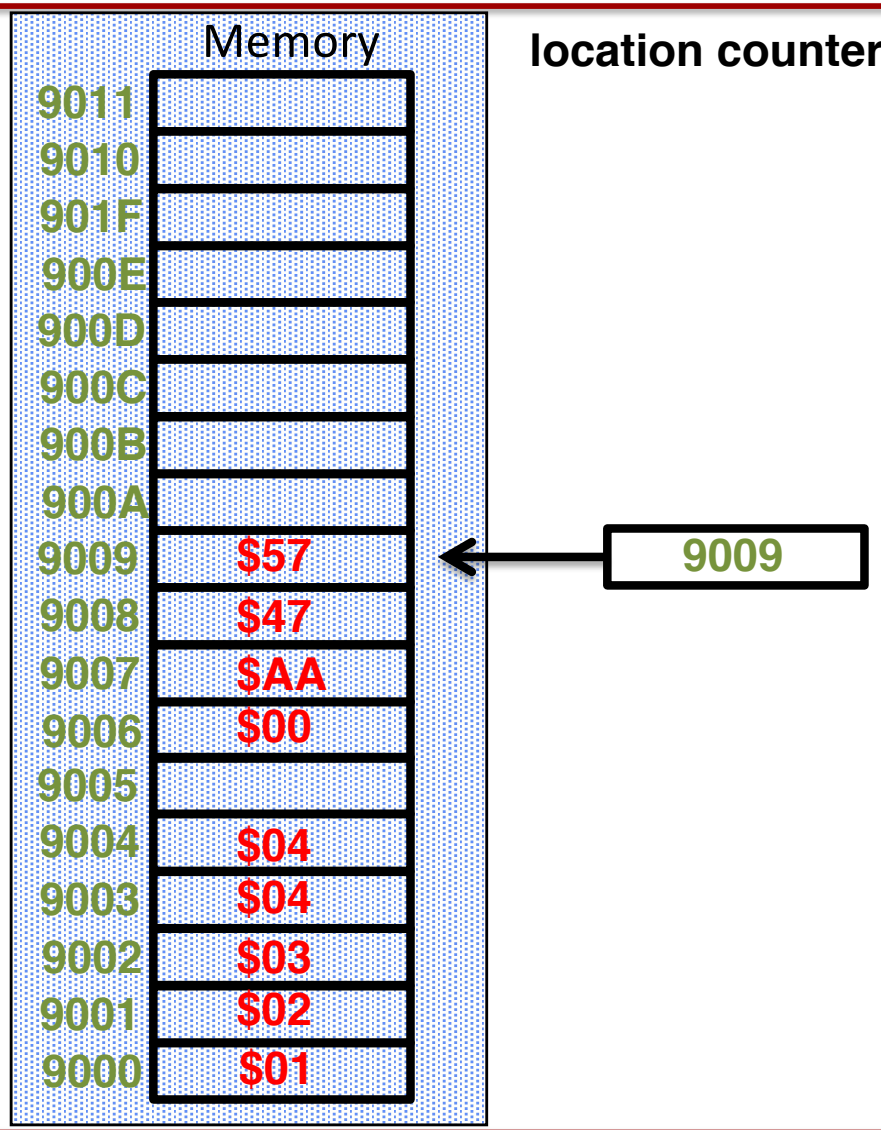


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008

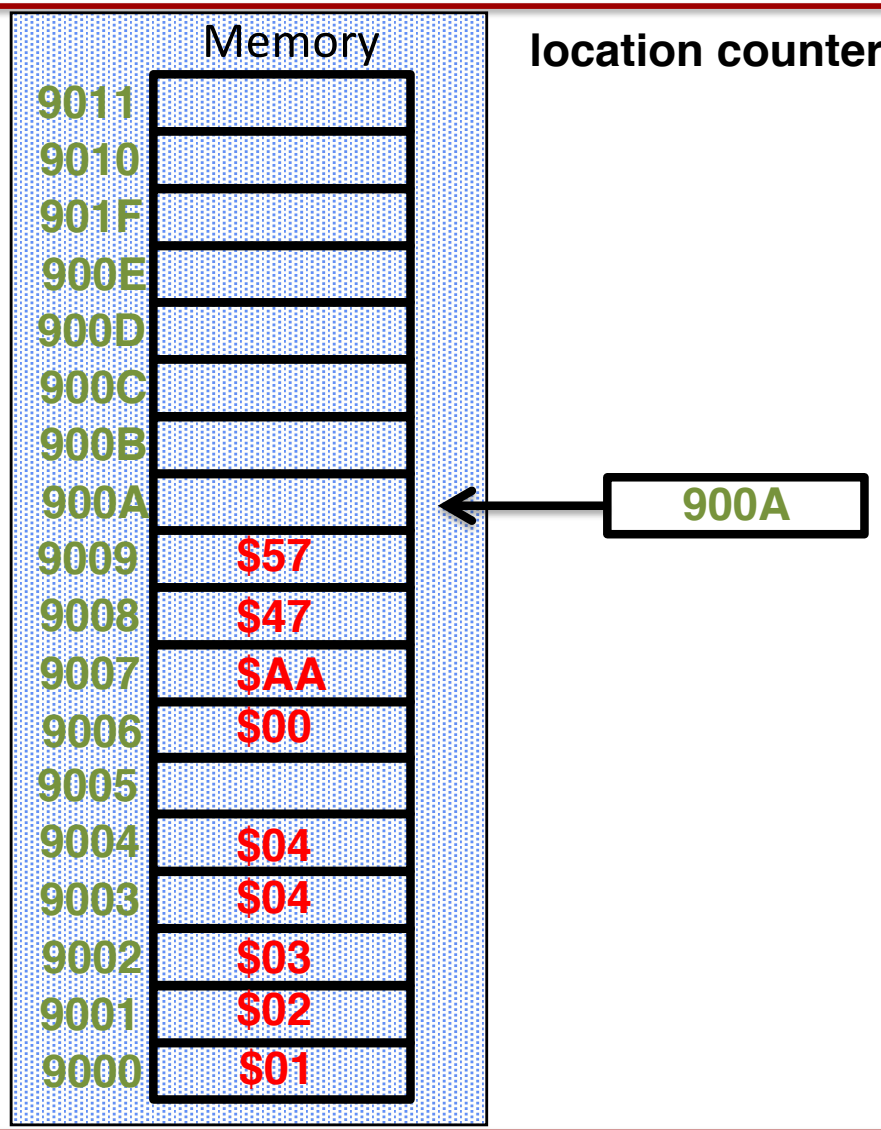


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008

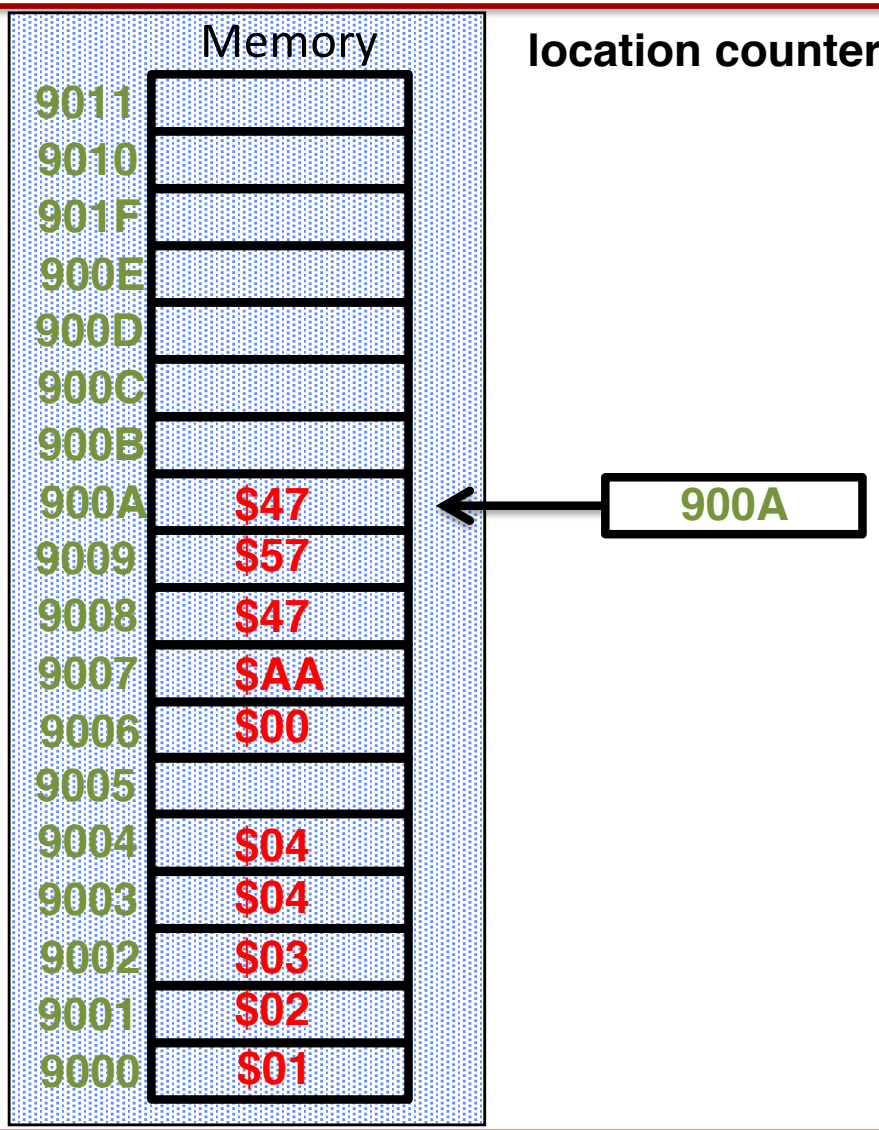


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008

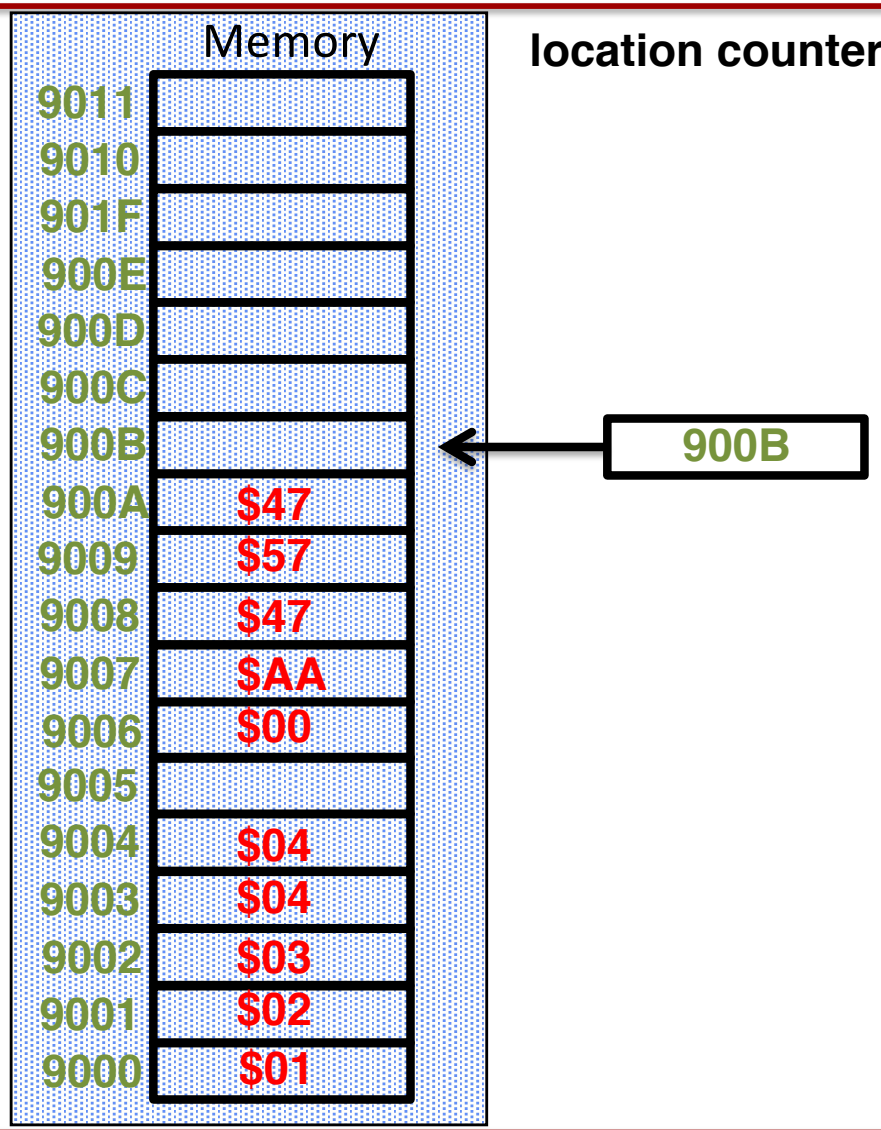


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008

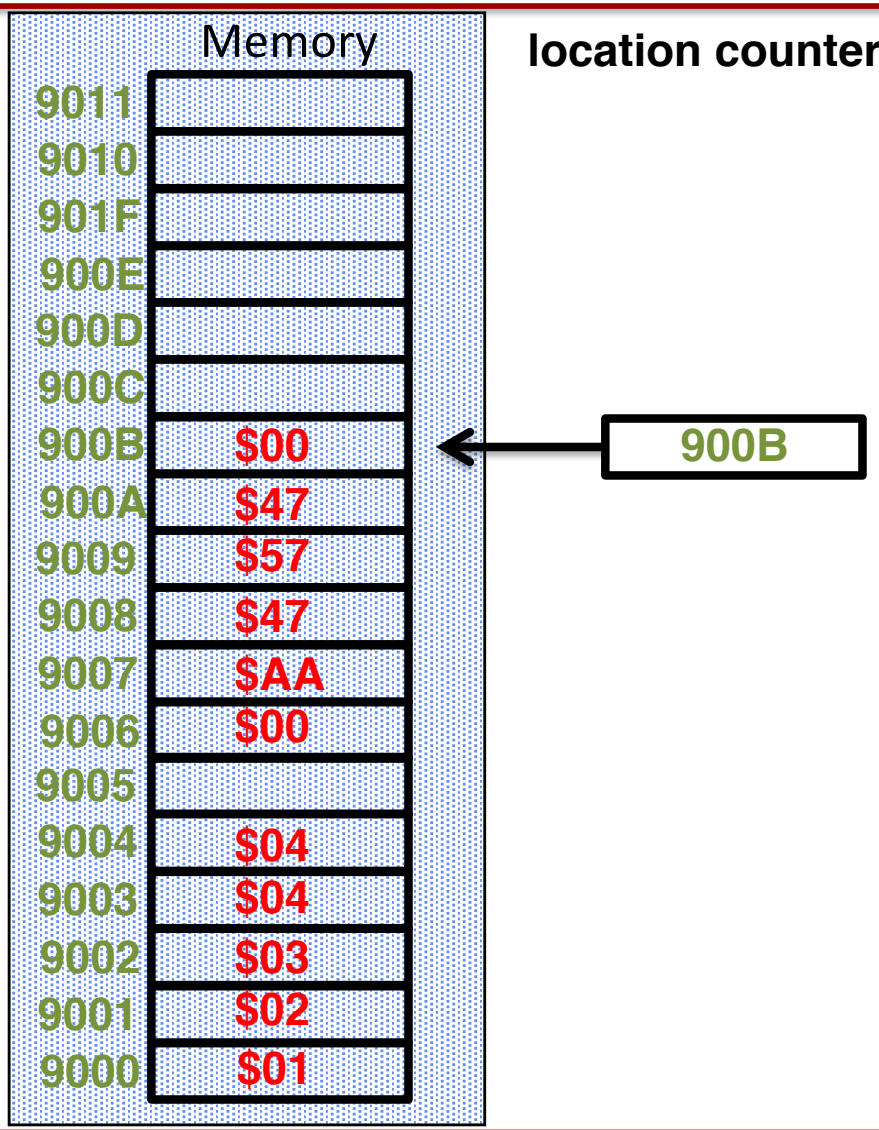


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008

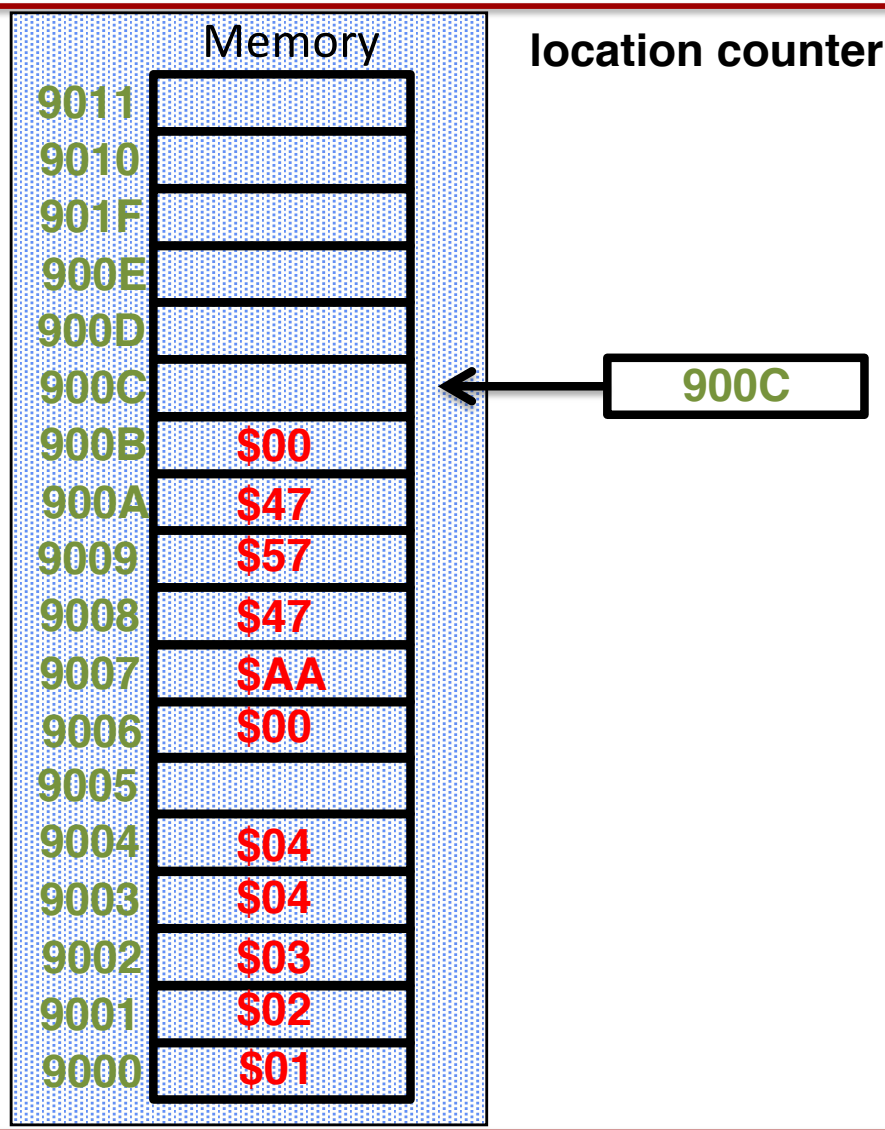


Example

```
        org     $9000
list    dc.b    1,$2,%11,4
val1    dc.b    val1-list
val2    dc.w    $AA
name    dc.l    'GWG'
```

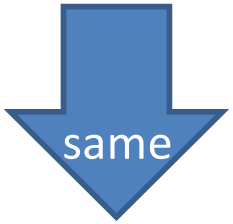
Symbol Table

Symbol	Address
list	00009000
val1	00009004
val2	00009006
name	00009008



Using Labels to Access Data

```
move.w val2,d0
```



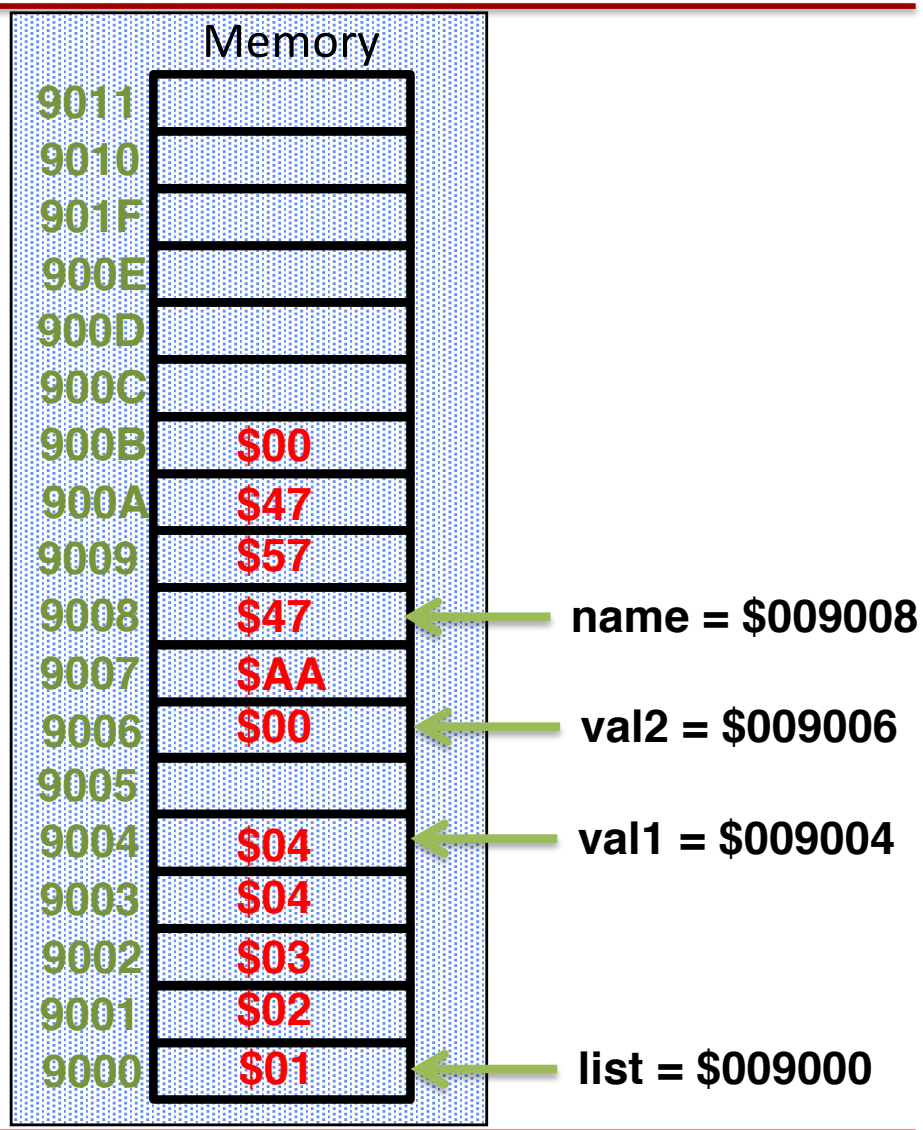

```
move.w $9006,d0
```

D0	$12345678

BEFORE

D0	$123400AA

AFTER



Common Mistake!!!

```
move.w #val2,d0
```

same

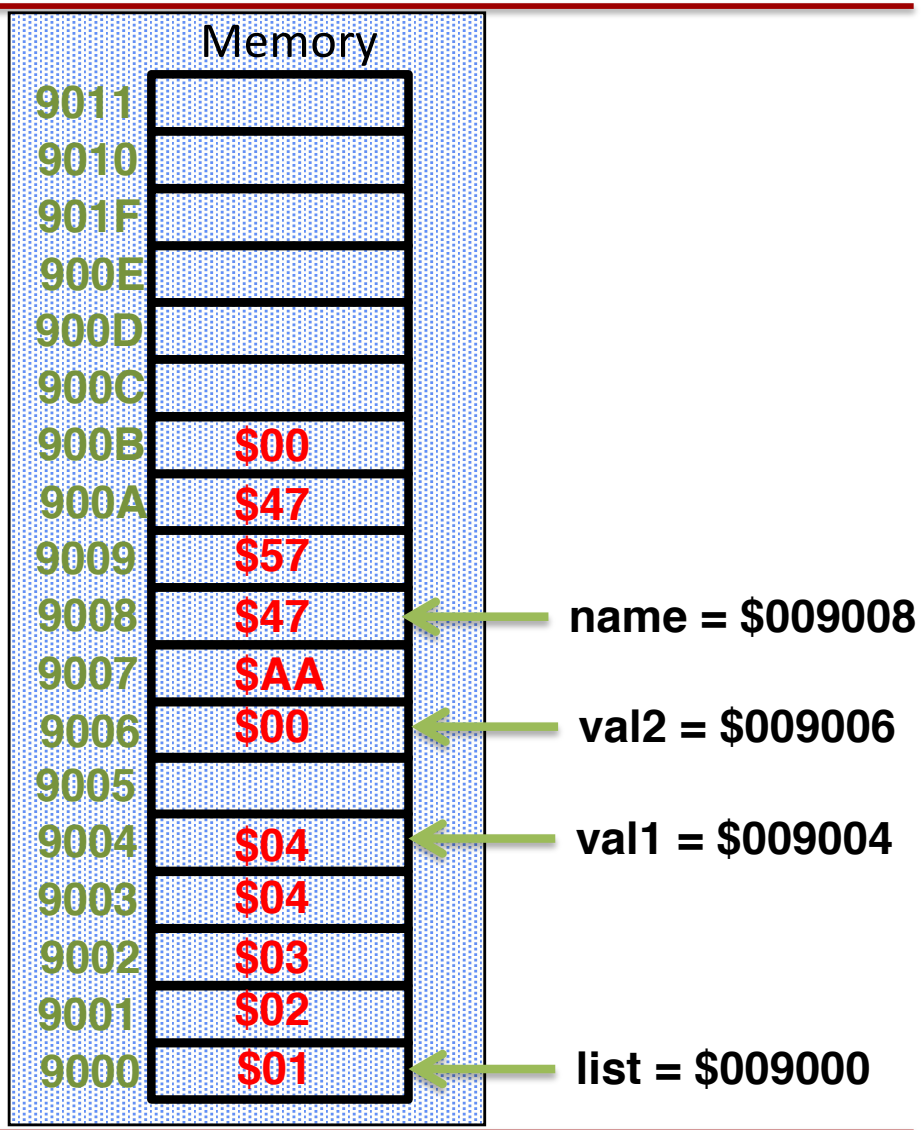
```
move.w #$9006,d0
```

D0	$12345678

BEFORE

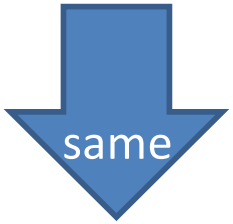
D0	$12349006

AFTER



Assemble-Time Expressions and Labels

```
move.b list+2,d0
```



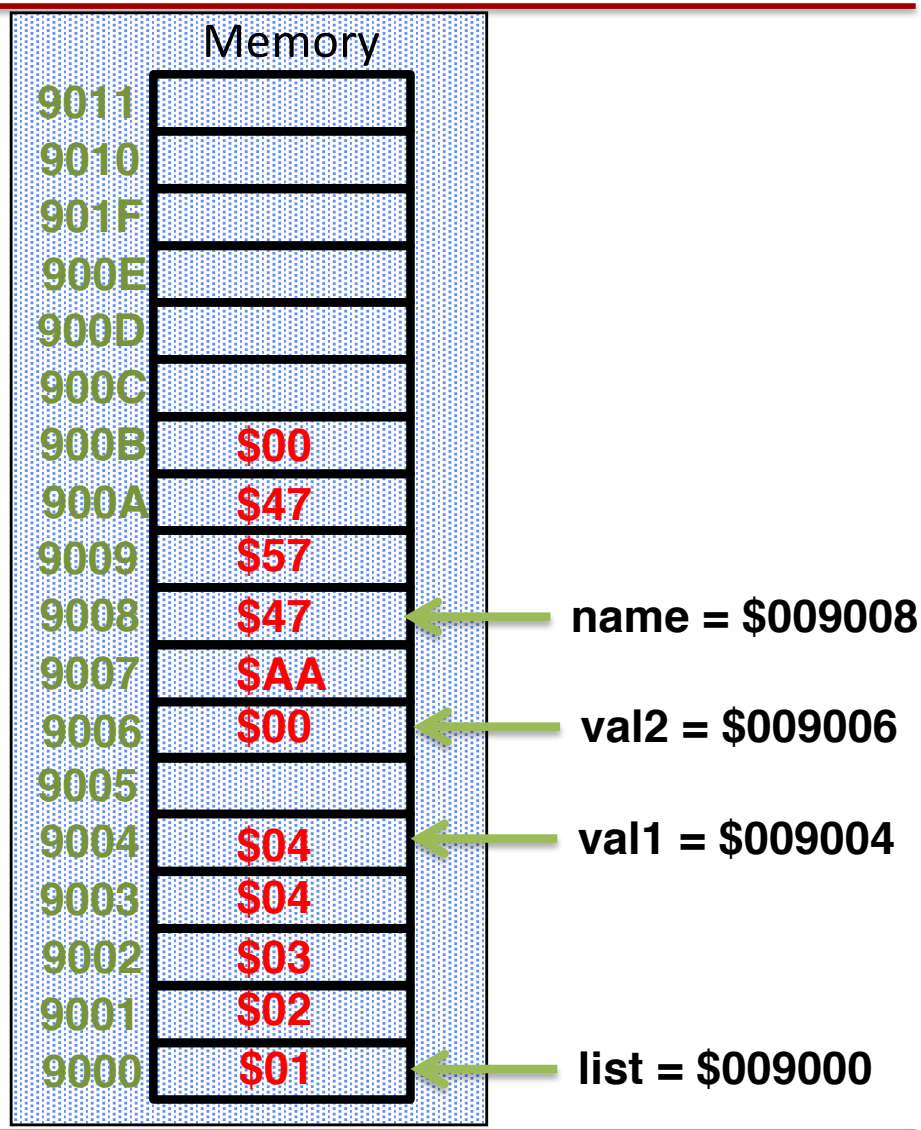

```
move.b $9002,d0
```

D0	$12345678

BEFORE

D0	$12345603

AFTER



Define Storage Directive

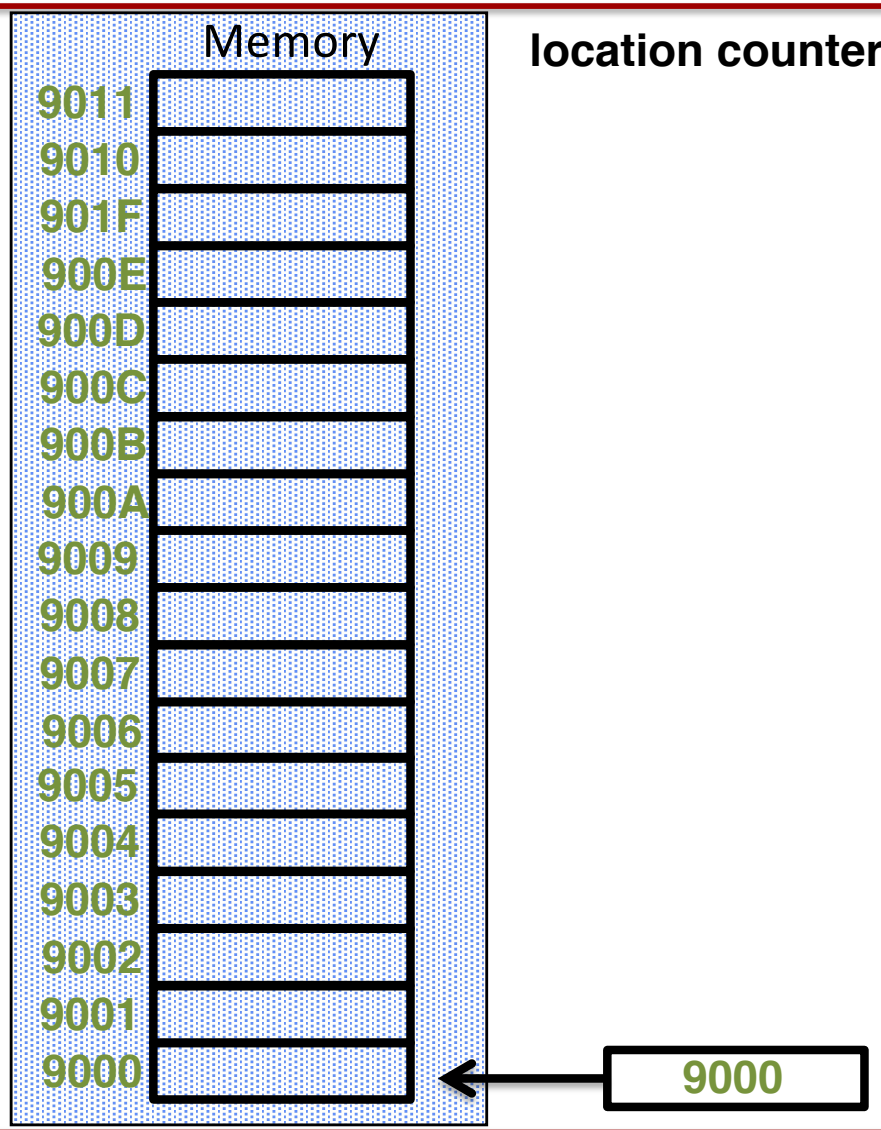
- The define storage directive allows you to reserve an uninitialized block of bytes, words, or long-words in memory
  - Format:
    - `<label>  DS.<size>  <count>`
  - The **`label`** will be defined to equal to the address of the start of the memory block
  - The **`size`** specifies that a block of bytes (.B), words (.W), or long-words (.L) is being reserved
  - **`count`** is the number of bytes, words, or long-words that will be in the block, and may be specified as a decimal number, hexadecimal number ($), a binary (%) number, or an assemble-time expression

Example

```
        org     $9000
list    ds.b    3
table   ds.w    2
addrs   ds.l    1
```

Symbol Table

Symbol	Address
list	00009000

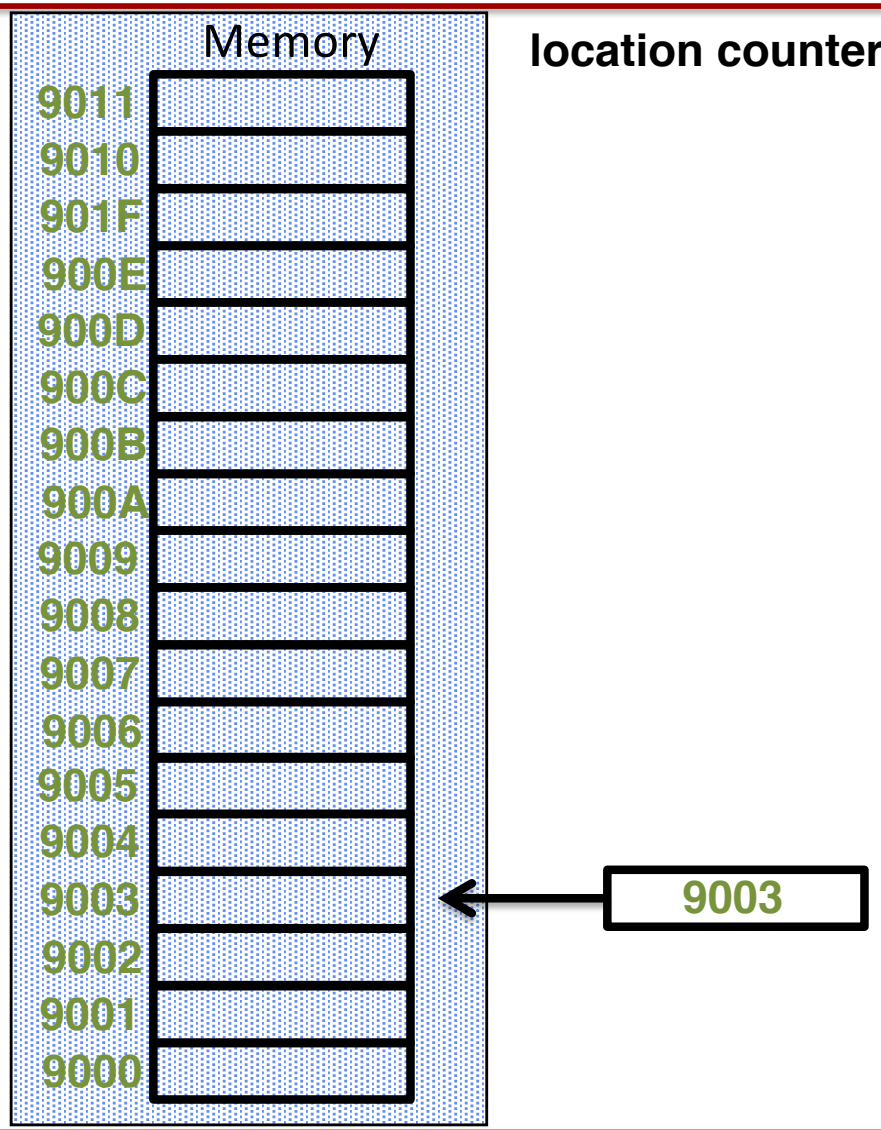


Example

```
        org     $9000
list    ds.b    3
table   ds.w    2
addrs   ds.l    1
```

Symbol Table

Symbol	Address
list	00009000

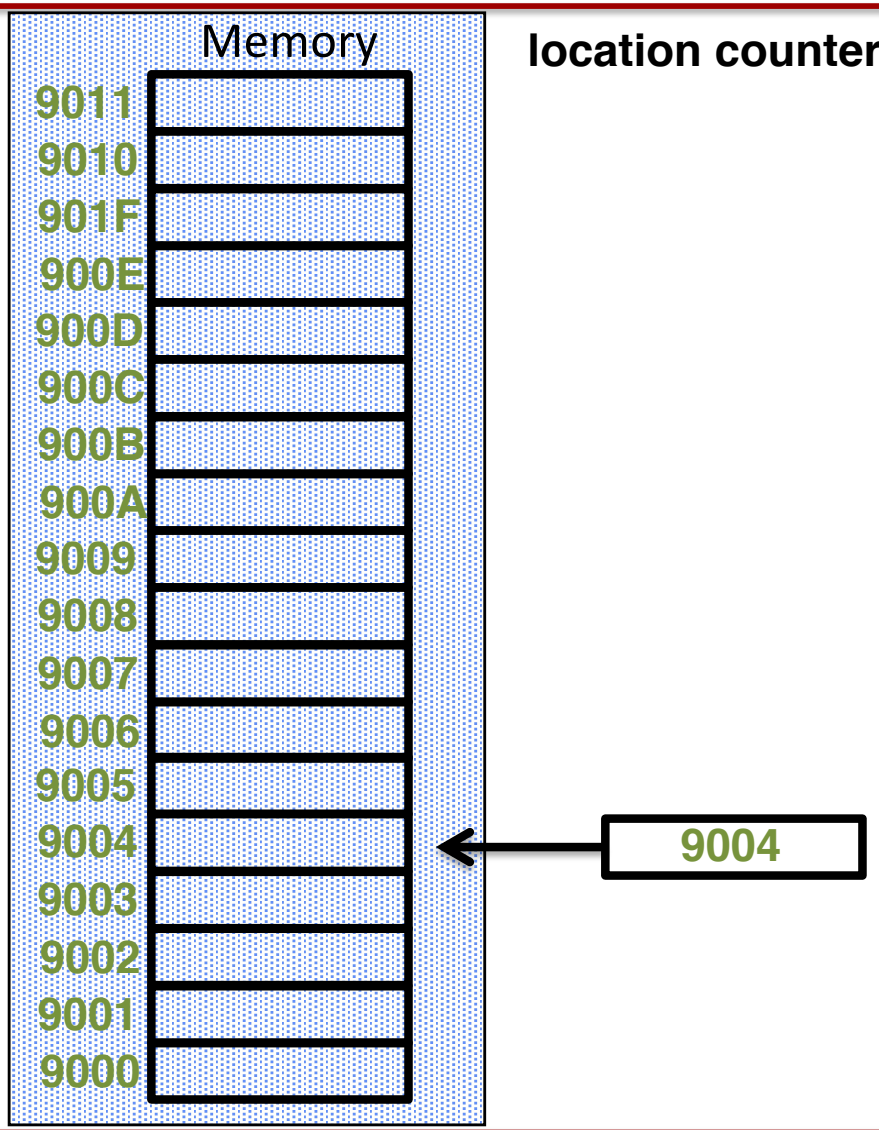


Example

```
        org     $9000
list    ds.b    3
table   ds.w    2
addrs   ds.l    1
```

Symbol Table

Symbol	Address
list	00009000
table	00009004

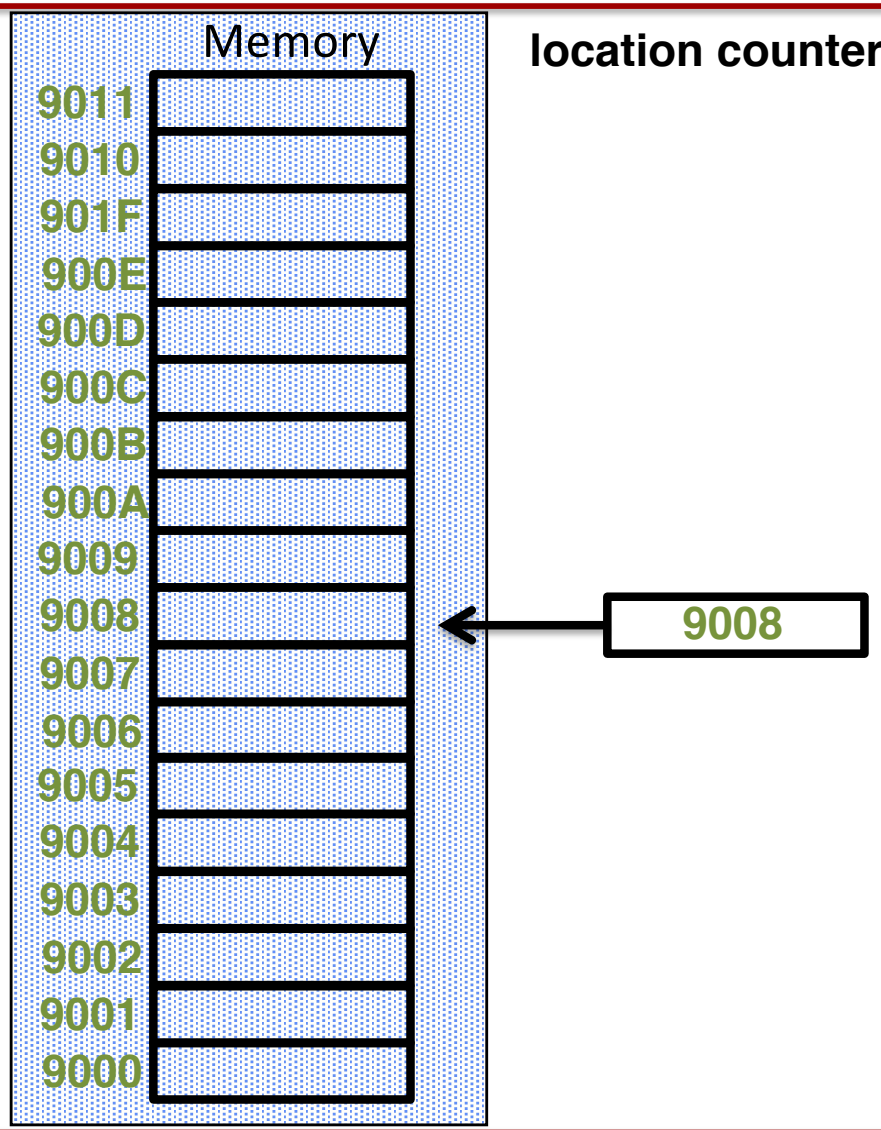


Example

```
        org     $9000
list    ds.b    3
table   ds.w    2
addrs   ds.l    1
```

Symbol Table

Symbol	Address
list	00009000
table	00009004
addrs	00009008

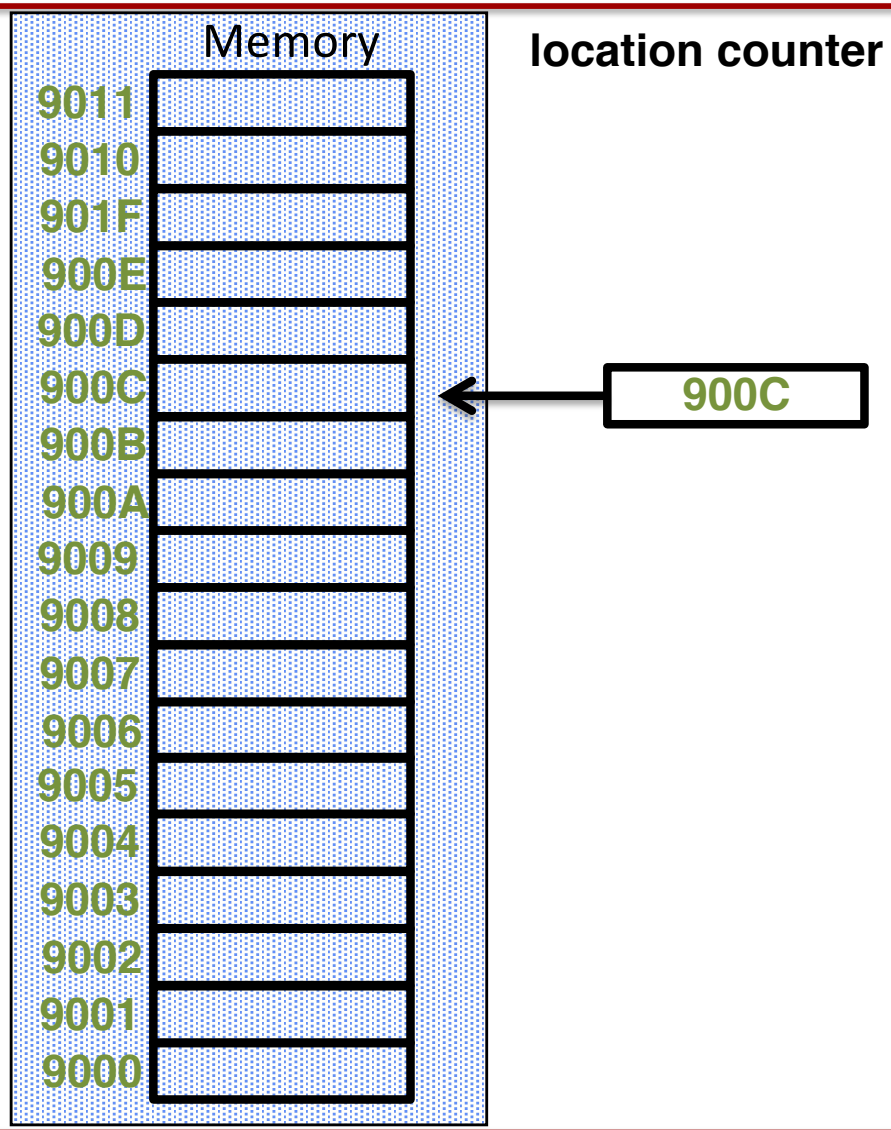


Example

```
        org     $9000
list    ds.b    3
table   ds.w    2
addrs   ds.l    1
```

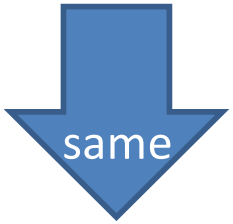
Symbol Table

Symbol	Address
list	00009000
table	00009004
addrs	00009008



Writing Memory using Labels

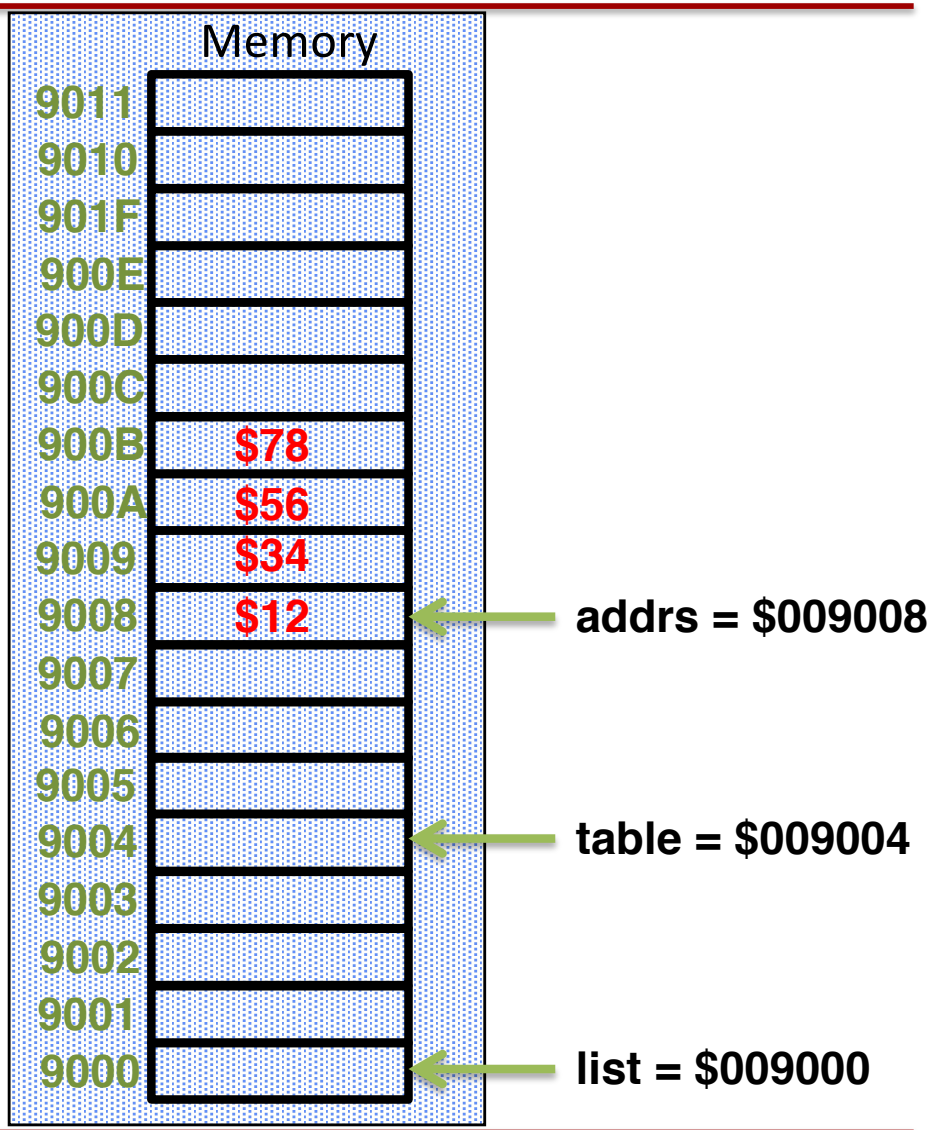
```
move.l d0,addrs
```




```
move.l d0,$009008
```

D0	$12345678

BEFORE



End Directive

- The end directive marks the end of the source file
  - Format:
    - **END <start address of program>**

```
start  move   #100,d0    ;1st program instruction
       ...
       ...
       end    start
```

DC, DS, and Global Variables in C

```
        SECTION    idata,,"data"
        XDEF _a
_a      DC.B 97
        DS.B 1     word align
        XDEF _b
_b      DC.W 3645
        XDEF _r
_r      DS.W 1
        XDEF _c
_c      DC.B 45
        DS.B 1     word align
        XDEF _d
_d      DC.L 234567
        XDEF _j
_j      DC.B 71
        DC.B 97
        DC.B 114
        DC.B 121
XDEF _k
_k      DC.B 71
        DC.B 97
        DC.B 114
        DC.B 121
        DC.B 0
        DS.B 1     word align
        XDEF _i
_i      DC.W 1
        DC.W 2
        DC.W 3
        DC.W 4
        DC.W 5
        DC.W 6
*1      char a = 'a' ;
*2      int  b = 3645;
*3      int r;
*3      short int c = 45;
*4      long int  d = 234567;
*5
*6      char j[] = {'G', 'a', 'r', 'y'};
*7      char k[] = "Gary";
*8      int  i[]  = {1,2,3,4,5,6};
*9
*10     void main(void) {

        SECTION    S_main,,"code"

_main
*11
*12     }
        RTS
        END
```

Assembler Flow