SEE Microprocessors

7: Arrays & Loops

Muhammad Mun’im Ahmad Zabidi (munim@utm.my)

cc BY NC SA

Module 7: Address Registers & Array Processing

- Special instructions for address registers
  - MOVEA, ADDA, SUBA
  - CMPA
  - LEA
- Understanding arrays
- Array applications

Using Address Registers

- Data registers support byte, word, and longword operations.
- Address registers support word and longword operations.
  - 32-bit address stored in an address register is a "single entity"
  - Effect of a word operation to the content of an address register is a longword operation.
- In the case of a word operation, the source operand is sign extended to a long word,
- All MC68000' s addresses are sign-extended to 32 bits for word operations.
  - ADDA.L #$FFF4,A0 will add $0000FFF4 to A0.
  - ADDA.W #$FFF4,A0 will have $FFF4 sign extended to $FFFFFFF4 before addition happens.

The 68000 Address Bus

- The 68000 has 32-bit address registers and program counter (PC)
- Because of packaging, A24 to A31 are not used
- The A00 is used to select the byte or word addressing
- The 68000 has actually 24-bit addressing !

  ⇒ can access only 2^{24} (16M) bytes in memory

  ⇒ bits 24-31 in any address register are don't cares
- 68020 has a full 32-bit address bus

⇒ How many bytes in memory can be accessed?

Special Instructions for Address Registers

`MOVEA,          ADDA,          SUBA,          CMPA`

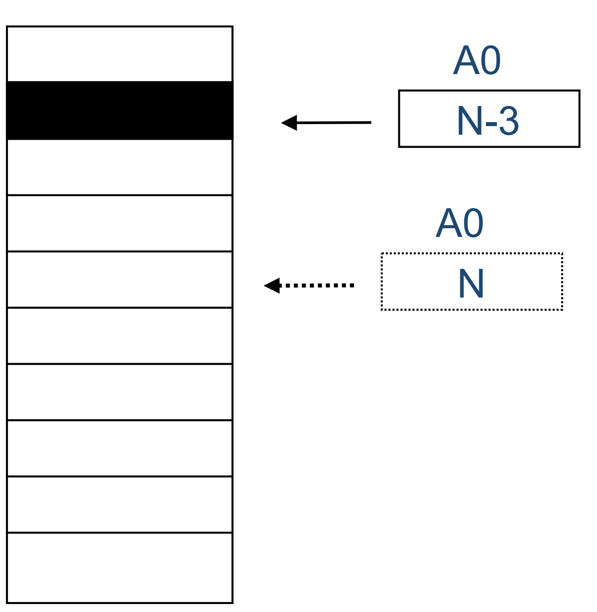
- If an address register is specified as the destination operand, then the following address register instructions, MOVEA, ADDA, SUBA and CMPA, will be used instead of MOVE, ADD, SUB and CMP, respectively. The mnemonics may be the same but the assembler will generate different machine code.
- LEA is a more powerful version of MOVEA that allows performs address calculation while loading an address register
- MOVEA, ADDA, SUBA, LEA do not affect the CCR.
- 68000 has an extra ALU specifically for address calculations
  - This ALU is not connected to the CCR

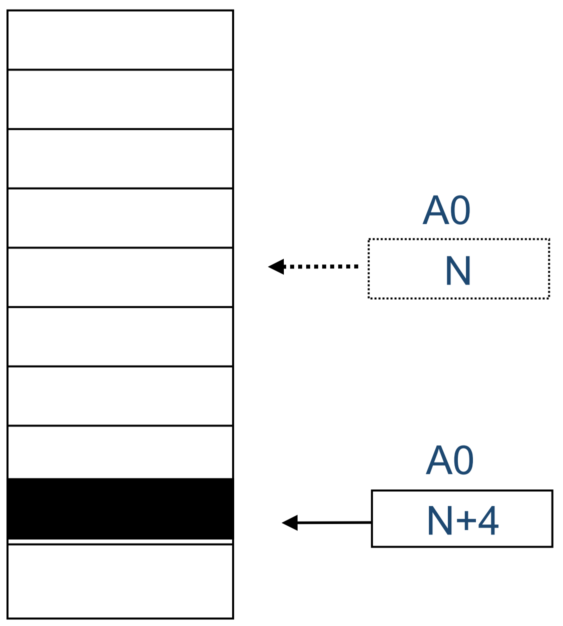
LEA (Load Effective Address)

- Computes the effective address of an operand and loads it into an address register
- Intrinsically a longword operation
  - => .L is not required
- # symbol not needed
- More powerful than MOVEA instruction

```
LEA    $0010FFFF,A5        [A5] ← $0010FFFF
LEA    (A0),A5             [A5] ← [A0]
LEA    (12,A0),A5          [A5] ← [A0]+12
LEA    (12,A0,D4.L),A5     [A5] ← 12+[A0]+[D4]
```

Modification of the Address Register





```
ADDA.W     #-3,A0          ADDA.W     #4,A0
```

Adding 5 Words

- A simple instruction sequence to add 5 numbers stored beginning at $1010 and store the sum in $2000:

```
MOVE.W $1010,D0
ADD.W  $1012,D0
ADD.W  $1014,D0
ADD.W  $1016,D0
ADD.W  $1018,D0
MOVE.W D0,$2000
```

1010	1
1012	2
1014	5
1016	7
1018	2

5 words

- What if you have 100 numbers?

```
MOVE.W $1010,D0
ADD.W  $1012,D0
… 97 more ADD.W instructions …
ADD.W  $10C6,D0
MOVE.W D0,$2000
```

A Better way to Add 5 or 100 Words

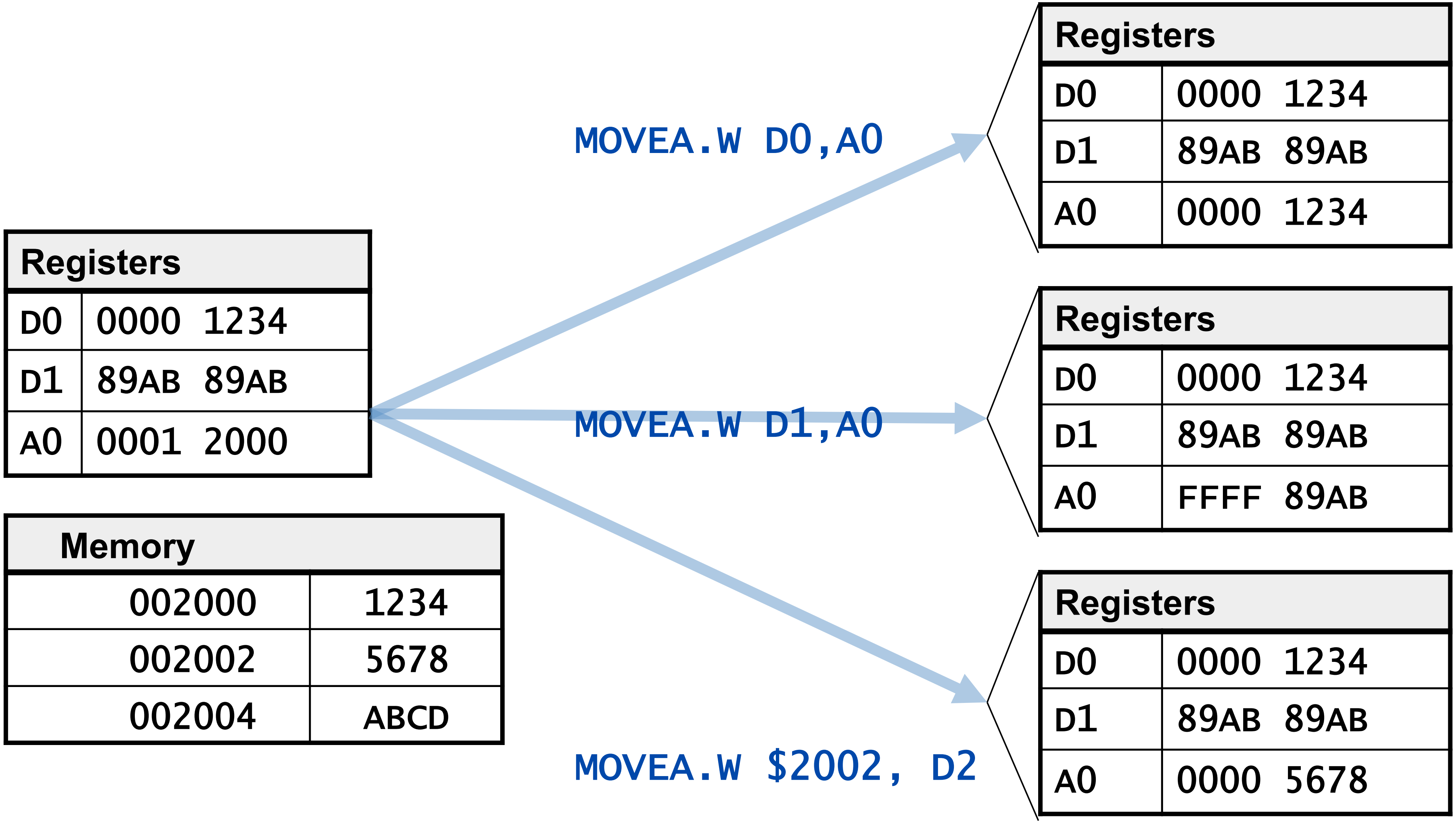
```
        MOVE.B   #5,D0          ; Five numbers to add
        MOVEA.L  #$1010,A0      ; A0 points at the numbers
        CLR.B    D1             ; Clear the sum
Loop    ADD.B    (A0)+,D1       ; REPEAT Add number to total
        SUB.B    #1,D0
        BNE      Loop           ; UNTIL all numbers added
        STOP     #$2700
```

- What if you have 100 numbers?

```
        MOVE.B   #100,D0        ; 100 numbers to add
        MOVEA.L  #$1010,A0      ; A0 points at the numbers
        CLR.B    D1             ; Clear the sum
Loop    ADD.B    (A0)+,D1      ; REPEAT Add number to total
        SUB.B    #1,D0
        BNE      Loop           ; UNTIL all numbers added
        STOP     #$2700
```

- Wasn’t that easy? Interested in making your life easier? Read on.

Address Register Instructions



Registers	
D0	0000 1234
D1	89AB 89AB
A0	0001 2000

Memory	
002000	1234
002002	5678
002004	ABCD

MOVEA.W D0,A0

Registers	
D0	0000 1234
D1	89AB 89AB
A0	0000 1234

MOVEA.W D1,A0

Registers	
D0	0000 1234
D1	89AB 89AB
A0	FFFF 89AB

MOVEA.W $2002, D2

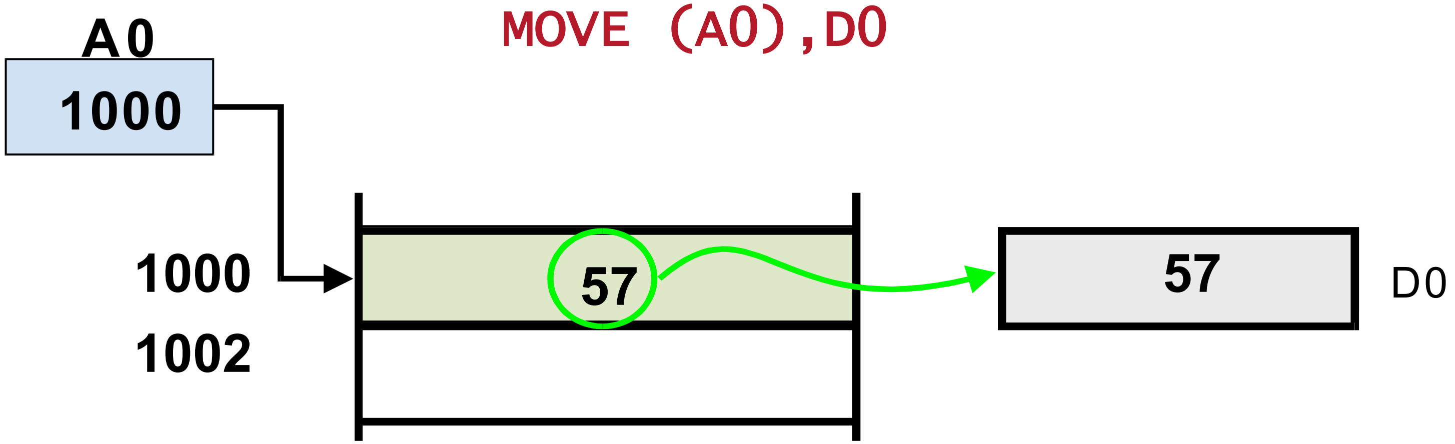
Registers	
D0	0000 1234
D1	89AB 89AB
A0	0000 5678

Address Register Indirect Addressing

- In address register indirect addressing, the instruction specifies one of the 68000's address registers; for example, MOVE.B (A0),D0.
- The specified address register contains the address of the operand.
- The processor then accesses the operand pointed at by the address register.

Address Register Indirect Addressing

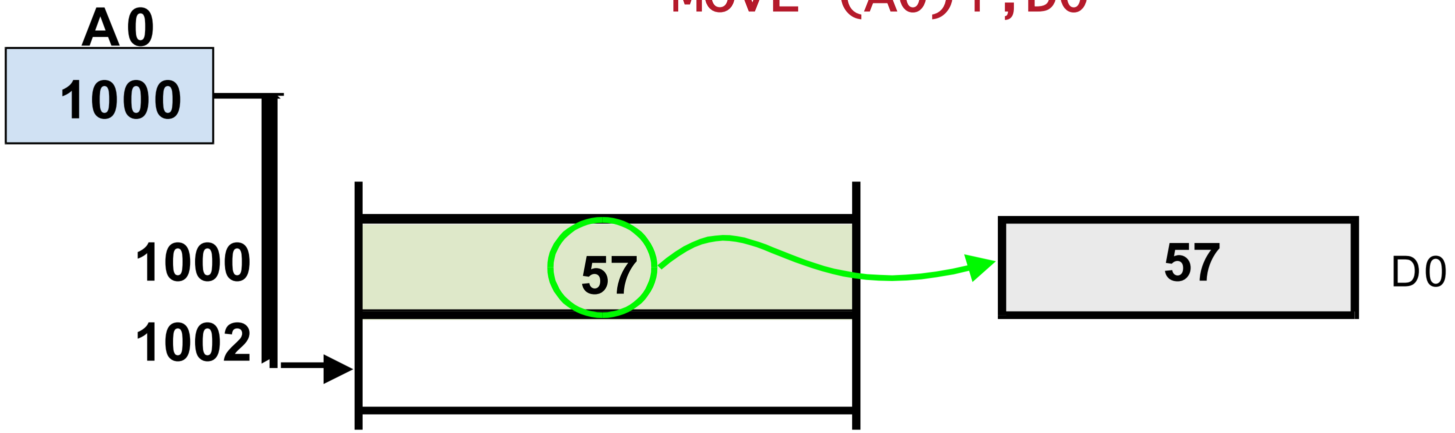
- This instruction means load D0 with the contents of the location pointed at by address register A0



Post-incrementing

- If the addressing mode is specified as (A0)+, the contents of the address register are incremented after they have been used.

MOVE (A0)+,D0



Address A0 register is used to access memory location 1000 and the contents of this location (i.e., 57) are added to D0

Byte, Word, and Longword Arrays

- One-dimensional array is stored in consecutive memory locations.
- Elements in arrays of different sizes have different step sizes

2000	1	2000	1	2000	1
2001	2	2002	2	2004	2
2002	5	2004	5	2008	5
2003	7	2006	7	200C	7
2004	2	2008	2	2010	2

Byte array
Step size = 1

Word array
Step size = 2

Longword array
Step size = 4

- Other examples:

```
Array1  DS.B 9            ; array of 9 bytes
Array2  DS.W 5            ; array of 5 words
Array3  DC.B 2,4,5,6,1,3  ; array of 6 bytes with initial values
```

Arrays

- Where can we locate an element of a 1-D array in memory?
- The location of the ith element a_i is
  - Base + (i-1) x element_size

- The 5th element is in location
  1000 + (5-1) x 1 = 1004
- Is this element Array1[5]
  or Array1[4]?

1000	4
1001	2
1002	7
1003	1
1004	3
1005	7
1006	4
1007	3
1008	1

```
Array1 DC.B    4,2,7,1,3,7,4,3,1
```

Note: Element_size is measured in byte

Example – Adding *n* integers

```
       MOVE.W  N,D1
       MOVE.W  #NUM,A2
       CLR.W   D0
LOOP   ADD.W   (A2)+,D0
       SUB.W   #1,D1
       BGT     LOOP
       MOVE.W  D0,SUM
       END
       ORG $001000
N      DC.W    7
NUM    DC.W 3,5,8,10,5,12,14
SUM    DS      1
```

MEMORY

$1000	7	N
$1002	3	NUM
$1004	5	
$1006	8	
$1008	10	
$100A	5	
$100C	12	
$100E	14	
$1010	?	SUM

Adding 5 Signed Words

- The sum may overflow if a word variable is used.
- Use longword for the sum
- Sign-Extend the current element before adding

```
        ORG     $1000
START   CLR.L   D0             ; Initialize sum
        MOVE.B  #5,D1          ; Set counter to 5
        LEA     ARRAY,A0
ULANG   MOVE.W  (A0)+,D2       ; Copy to temporary location
        EXT.W   D2             ; Extend to 32 bit
        ADD.L   D2,D0          ; then add to running sum
        SUB.B   #1,D1          ; Decrement counter
        BNE     ULANG
        STOP    #$2700
ARRAY   DC.W    1,2,-3,4,-10
        END     START
```

Adding 5 Unsigned Words

- Slightly different technique
- Use longword for the sum
- Clear the temporary register before entering the loop

```
        ORG     $1000
START   CLR.L   D0              ; Initialize sum
        MOVE.B  #5,D1           ; Set counter to 5
        CLR.L   D2              ; Clear temporary register
        LEA     ARRAY,A0
ULANG   MOVE.W  (A0)+,D2        ; Copy to temporary reg
        ADD.L   D2,D0           ; then add to running sum
        SUB.B   #1,D1           ; Decrement counter
        BNE     ULANG
        STOP    #$2700
ARRAY   DC.W    1,2,-3,4,-10
        END     START
```

Post-Increment

- This post-increment facility is similar to that in C/C++/Java, which is useful when a list of operands are to be accessed in sequence.
- Example: Suppose we have an array holding eight values 1, 2, 3, 4, 5, 6, 7, 8. A C program which adds all elements of the **array** could be written as:

```
main() {
    int array[ ] = {1, 2, 3, 4, 5, 6, 7, 8};
    int sum = 0;
    int index = 0;
    int count = 8;
    for(; ;) {
           sum += array[index++];      // post-increment
           count - -;
           if(count > 0) continue;
           else break;
           }
    }
```

The corresponding assembly program:

```
        ORG     $400

START   LEA     ARRAY, A1      * A1 points to ARRAY
        MOVE.B  #8, D1         * set up the count
        CLR.W   D2             * clear D2 for the sum
LOOP    ADD.W   (A1)+,D2       * add array element to D2
        SUB.B   #1, D1         * decrement the count
        BNE     LOOP           * back to LOOP if D1>0
        MOVE.W  D2, SUM        * result into memory

        STOP    #$2700

        ORG     $4000
ARRAY   DC.W    1, 2, 3, 4, 5, 6, 7, 8 * the word array
SUM     DS.W    1              * space for the sum

        END     START
```

7-Segment LED and Lookup Table

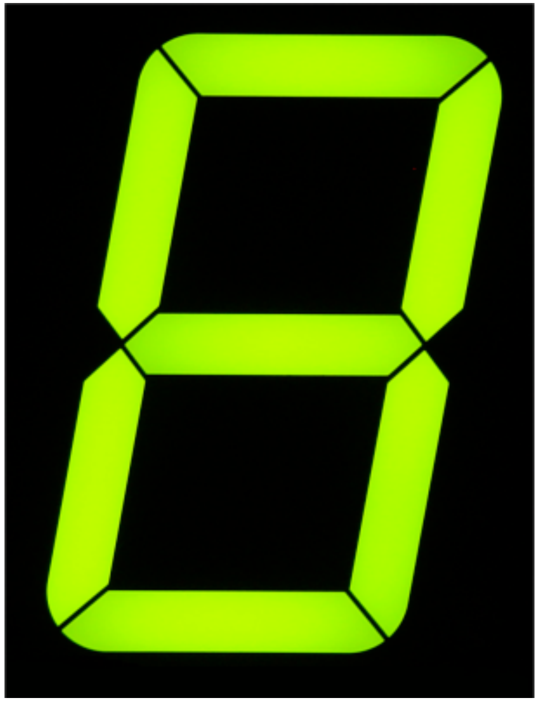
```
SEVEN     EQU       $E011     ; IDE68k 1st digit

          ORG       $1000

MAIN      MOVEA.L   #TAB,A0
START     MOVE.B    (A0)+,D0
          BEQ       MAIN
          MOVE.B    D0,SEVEN
          BSR       DELAY
          BRA       START

DELAY     MOVE.L    #$2FFFF,D1
LOOP      SUB.L     #1,D1
          BNE       LOOP
          RTS

TAB       DC.B      %00111111,%00000110,%01011011,%01001111
          DC.B      %01100110,%01101101,%01111101,%00000111
          DC.B      %01111111,%01100111,0
```



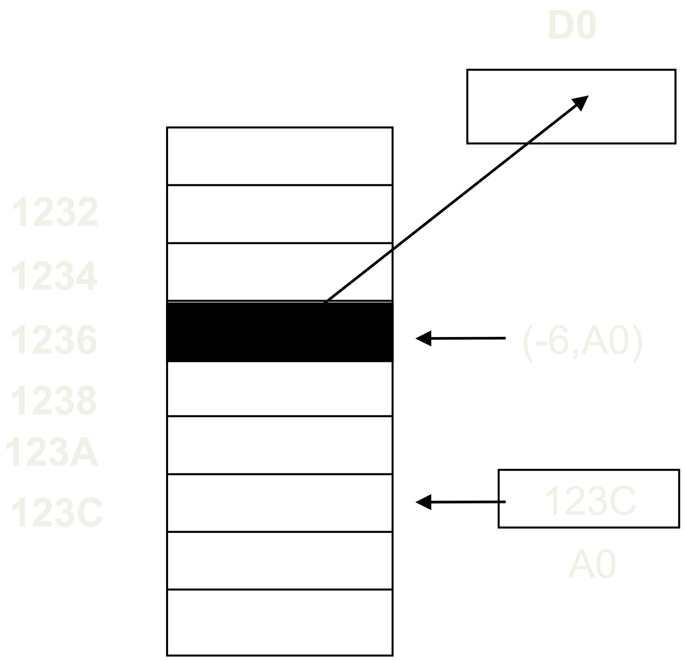
Example: Comparing Memory Blocks

```
* This program compares two blocks of memory.
* If the memory is equal, then FF is stored in address register D0,
* otherwise, 00 is stored.
        ORG     $400          ; Program origin
        LEA     Block1,A0     ; Point to beginning of memory block 1
        LEA     Block2,A1     ; Point to beginning of memory block 2
        MOVE.W  #Size,D0      ; Store the long word count in size
LOOP    CMPM.L  (A0)+,(A1)+   ; Compare the long words
        BNE     NotEq         ; Branch if not equal
        SUBQ.W  #1,D0         ; Otherwise, decrement the count
        BNE     LOOP          ; Go back for another comparison
        CLR.L   D0            ; Two strings are equal so set
        MOVE.B  #$FF,D0       ;       D0 to FF
        BRA     Exit
NotEq   CLR.L   D0            ; Otherwise, set D0 to 00
Exit    STOP    #$2700
Size    EQU     2             ; Compare 2 words
        ORG     $600
Block1  DC.L    'Bloc','1234';      Block 1
        ORG     $700
Block2  DC.L    'Bloc','1234 '      ;       Block 2
        END     $400
```

Compare Memory Blocks for Equality

```
BLOCK1      EQU      <address1>
BLOCK2      EQU      <address2>
SIZE        EQU      <# of words in block>
            LEA      BLOCK1,A0
            LEA      BLOCK2,A1
            MOVE.W   #SIZE,D0
LOOP        CMPM.W   (A0)+,(A1)+
            BNE      NOT_SAME
            SUBQ.W   #1,D0
            BNE      LOOP
ALL_SAME    …
NOT_SAME    ...
```

Register Indirect with Displacement



```
MOVE.W     (-6,A0),D0
```

or

```
MOVE.W     -6(A0),D0
```

Character Translation Using Lookup Table

```
        MOVE.B   #8,D1
LOOP    ROL.L    #4,D2
        MOVE.B   D2,D3
        ANDI.L   #$0000000F,D3
        MOVEA.L  D3,A0
        MOVE.B   TRANS(A0),D0
        BSR      PRINT_CHAR
        SUB.B    #1,D1
        BNE      LOOP
        ...
TRANS   DC.B     ‘0123456789ABCDEF’
```

```
D2:0101 1010 0011 0100 1110 1111 0110 1101
Printed as:
5A34EF6D
```

Offset	Hex	ASCII
0	$30	‘0’
1	$31	‘1’
2	$32	‘2’
3	$33	‘3’
4	$34	‘4’
5	$35	‘5’
6	$36	‘6’
7	$37	‘7’
8	$38	‘8’
9	$39	‘9’
10	$41	‘A’
11	$42	‘B’
12	$43	‘C’
13	$44	‘D’
14	$45	‘E’
15	$56	‘F’

Searching for Minimum Value

```
* Assume all numbers are unsigned words

        ORG     $1000

START   MOVEA   #ARRAY,A0       ; Set PTR to array
        MOVE    #65535,D0       ; Initialize MIN
        MOVE    #10,D1          ; Set counter to element count

LOOP    CMP     (A0),D0         ; Compare MIN with current
        BLS     SKIP            ; If MIN is lower/same, skip
        MOVE    (A0),D0         ; Else copy current to MIN
SKIP    ADDA    #2,A0           ; Manually increment pointer
        SUB     #1,D1           ; Decrement counter
        BNE     LOOP

        STOP    #$2700

ARRAY   DC.W    10000,32,12,33,4,10,50,1000,22,33

        END     START
```

Searching for Minimum Value V.2

```
* Assume all numbers are unsigned words

        ORG     $1000

START   LEA     ARRAY,A0        ; Set PTR to array
        MOVE    (A0)+,D0        ; Initialize MIN with 1st elt
        MOVE    #9,D1           ; Counter <- element count - 1

LOOP    CMP     (A0)+,D0        ; Compare MIN with current
        BLS     SKIP            ; If MIN is lower/same, skip
        MOVE    -2(A0),D0       ; Else copy current to MIN
SKIP    SUB     #1,D1           ; Decrement counter
        BNE     LOOP

        STOP    #$2700

ARRAY   DC.W    10000,32,12,33,4,10,50,1000,22,33

        END     START
```

Counting Elements Of Specified Range

```
* Assume ARRAY has LENGTH unsigned word elements
* This program counts elements < 10

        ORG     $1000

START   LEA     ARRAY,A0        ; Set PTR to array
        CLR     D0              ; Initialize COUNT
        MOVE    #LENGTH,D1      ; Initialize LOOPCTR

LOOP    MOVE    (A0)+,D1        ; Fetch current element
        CMP     #10,D1          ; Is it < 10
        BHS     SKIP            ; If > or =, get next elt
        ADD     #1,D0           ; Else increment COUNT
SKIP    SUB     #1,D1           ; Continue until all done
        BNE     LOOP

DONE    STOP    #$2700

ARRAY   DC.W    10000,32,12,33,4,10,50,1000,22,33
LENGTH  EQU     (*-ARRAY)/2
        END     START
```

Splitting an Array

```
* Copy odd values to ODD array
        ORG     $1000

START   LEA     SOURCE,A0       ; Set ptr1 to source
        LEA     DEST,A1         ; Set ptr2 to dest
        MOVE    #LENGTH,D0      ; Initialize LOOPCTR

LOOP    MOVE    (A0)+,D1        ; Fetch current element
        BTST    #0,D1           ; Is it ODD?
        BEQ     SKIP            ; If EVEN, then skip
        MOVE    D1,(A1)+        ; Else copy to dest array
SKIP    SUB     #1,D0           ; Continue until all done
        BNE     LOOP

DONE    STOP    #$2700
        ORG     $1080
SOURCE  DC.W    10000,32,12,33,4,10,31,11,22,33
LENGTH  EQU     (*-ARRAY)/2
        ORG     $1100
DEST    DS.W    LENGTH
        END     START
```

Sorting

```
* Assume ARRAY has LENGTH unsigned word elements
        ORG     $1000
START   MOVE    #LENGTH,D0     ; Outer Loop Ctr <- Length
        SUB     #1,D0
OLOOP   LEA     ARRAY,A0       ; Set PTR to array
        MOVE    D0,D1                  ; Inner Loop Ctr <- Length
ILOOP   MOVE    (A0)+,D2       ; Fetch current element
        CMP     (A0),D2        ; Is A(i) > A(i+1)
        BHS     SKIP                   ; If > skip, else swap
        MOVE    (A0),-2(A0)
        MOVE    D2,(A0)
SKIP    SUB     #1,D1                  ; Continue until all scanned
        BNE     ILOOP
        SUB     #1,D0                  ; Rpt until LENGTH-1 passes
        BNE     OLOOP
DONE    STOP    #$2700
ARRAY   DC.W    10000,32,12,331,4,10,50,1000,22,33
LENGTH  EQU     (*-ARRAY)/2
        END     START
```

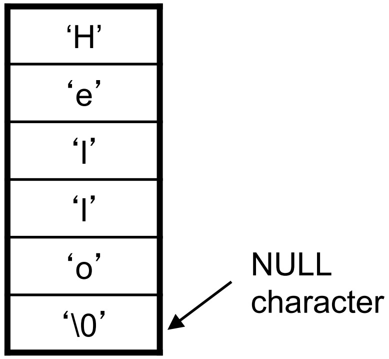
Example: Counting 6’s in An Array

```
*
* A region of memory starting at location $1000 contains
*  an array of  20 one-byte values.
* This program counts the number of 6’s in this array
*  and stores the count in register D1.
*
        ORG     $400                Program origin
        LEA     Array,A0            A0 points to the start of the array
        MOVE.B  #20,D0              20 values to examine
        CLR.B   D1                  Clear the 6’s counter
Next    MOVE.B  (A0)+,D2            Pick up an element from the array
        CMP.B   #6,D2               Is it a 6?
        BNE     Not_6               IF not 6 THEN skip counter increment
        ADD.B   #1,D1               IF 6 THEN bump up 6’s counter
Not_6   SUB.B   #1,D0               Decrement loop counter
        BNE     Next                Repeat 20 times
        STOP    #$2700              Halt processor at end of program
        ORG     $1000
Array   DC.B    1,6,4,5,5,6,2,5,6,7,6,6,6,1,3,5,9,6,7,5
        END     $400
```

Introduction to Strings

- Strings are arrays that contains only ASCII characters
- Two types of strings:
  - Null-terminated (also used by C language)
  - Started by a character count (also used by Pascal language)
- C-style strings are generally easier to use

The string “Hello” in C-style string.

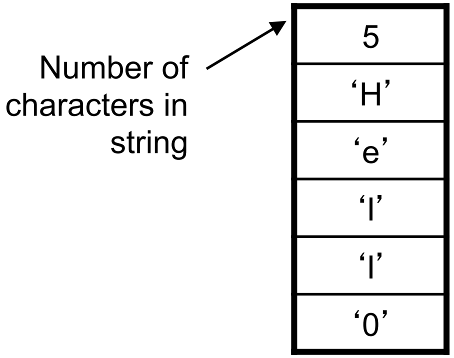


‘H’
‘e’
‘l’
‘l’
‘o’
‘\0’

NULL character

```
STR1      DC.B      ‘Hello’,0
```

The string “Hello” in Pascal-style string.



Number of characters in string

5
‘H’
‘e’
‘l’
‘l’
‘0’

```
STR2      DC.B      5,‘Hello’
```

Converting a C-String to All-Uppercase

```
* Convert character by character until NULL is found

        ORG     $1000

START   LEA     STRING,A0     ; Set PTR to array

LOOP    MOVE.B  (A0)+,D1      ; Fetch current element
        BEQ     DONE          ; If NULL, we’re done
        CMP.B   #‘a’,D1
        BLO     LOOP          ; If < ‘a’ get next char
        CMP.B   #‘z’,D1
        BHI     LOOP          ; If > ‘z’ get next char
        SUB.B   #$20,D1       ; Else convert to uppercase
SKIP    BRA     LOOP          ; Continue until done

DONE    STOP    #$2700

STRING  DC.B    ‘qUiCk BrOwN fOx JuMpS oVeR ThE lAzY DoG’,0
        END     START
```

Converting a Pascal-String to All-Lowercase

```
* Get the character count, and use it as loop counter

        ORG      $1000

START   LEA      STRING,A0     ; Set PTR to array
        MOVE.B   (A0)+,D1      ; Fetch loop counter

LOOP    BEQ      DONE          ; If NULL, we’re done
        CMP.B    #‘A’,D1
        BLO      SKIP          ; If < ‘A’ skip it
        CMP.B    #‘Z’,D1
        BHI      SKIP          ; If > ‘Z’ skip it
        OR.B     #$20,D1       ; Else convert to lowercase
SKIP    SUB.B    #1,D0
        BRA      LOOP          ; Continue until done

DONE    STOP     #$2700

STRING  DC.B     39            ; Isn’t there an easier way?
        DC.B     ‘qUiCk BrOwN fOx JuMpS oVeR ThE lAzY DoG’
        END      START
```

Concatenating Strings

```
* Concatenating means joining two strings to become one.

         ORG      $1000

START    LEA      SRC1,A0           ; Set ptr1 to first string
         LEA      DEST,A1           ; Set ptr2 to destination
* Copy first string to destination
LOOP1    MOVE.B   (A0)+,D0          ; Copy to D0
         BEQ      COPY2             ; if NULL don’t copy it
         MOVE.B   D0,(A1)+
         BRA      LOOP1
* Copy second string to destination
COPY2    LEA      SRC2,A0
LOOP2    MOVE.B   (A0)+,(A1)+
         BNE      LOOP2             ; Copy until NULL found
DONE     STOP     #$2700

SRC1     DC.B     ‘String 1’,0
SRC2     DC.B     ‘STRING 2’,0
DEST     DS.B     100
         END      START
```

Trimming Strings

```
* Trimming means removing excess spaces.
* This program performs left-trim only.
        ORG     $1000

START   LEA     SOURCE,A0       ; Set ptr1 to source
        LEA     DEST,A1         ; Set ptr2 to destination
* Skip spaces until non-space character is found
*  Assume some non-space characters exist in the string
LOOP1   MOVE.B  (A0)+,D0        ; Copy to D0
        BEQ     DONE            ; If NULL, non-space not found
        CMP.B   #’ ‘,D0
        BEQ     LOOP1
* Copy all characters until end of string
LOOP2   MOVE.B  (A0)+,(A1)+
        BNE     LOOP2           ; Copy until NULL found
DONE    CLR.B   A1              ; Add NULL to end of string
        STOP    #$2700

SOURCE  DC.B    ‘          Hello’,0
DEST    DS.B    100
        END     START
```

Finding a Specific Character in a String

```
* If found, D0 = $FF. Else, D0 = $00.
        ORG     $1000

START   LEA     STRING,A0       ; Set ptr to string
        CLR.B   D0              ; Assume not found
* Skip spaces until non-space character is found
LOOP    MOVE.B  (A0)+,D1        ; Copy to D0
        BEQ     DONE            ; If NULL, we’re done
        CMP.B   #’Z’,D1
        BNE     LOOP            ; If NULL, non-space not found
        MOVE.B  #$FF,D0

DONE    STOP    #$2700

STRING  DC.B    ‘qUiCk BrOwN fOx JuMpS oVeR ThE lAzY DoG’,0
        END     START
```

Program Counter Relative Addressing

- Special case of register indirect
  - □ displacement: EA = [pc] + d_{16}
  - □ index: EA = [PC] + [Xn] + d_8

- When the code is assembled, the assembler uses the offset TABLE (or relative address of TABLE) to calculate (memory_loc_of_TABLE - [PC])
- When the instruction is executed, the offset is added to current PC to give the address of TABLE

```
         MOVE.B   TABLE(PC),D2
         …
TABLE    DC.B  Value1
         DC.B  Value2
```

Program Counter Relative Addressing

```
00001000                                 ORG      $1000
00001000  143900002000                   MOVE.B   TABLE,D2
00001006  4E722700                       STOP     #$2700

00002000                                 ORG      $2000
00002000  0F                   TABLE     DC.B      $0F
00001000                                 END      $1000
```

2000 - 1002

```
00001000                                 ORG      $1000
00001000  143A0FFE                       MOVE.B   TABLE(PC),D2
00001004  4E722700                       STOP     #$2700

00002000                                 ORG      $2000
00002000 0F                    TABLE     DC.B     $0F
               00001000                  END      $1000
```

Relative Addressing

- [PC]=1002 after read a word from 1000, 1001.
- The location for TABLE is at 2000.
- The offset for TABLE is 2000 - 1002 = 0FFE.
- The program counter relative addressing enables position independent coding (PIC).
- The program can be placed anywhere in the memory, or be relocated.