7. Complex Data Types
Arrays

An array is a set of related items that are all of the same type (words, bytes, etc.). Arrays allow the programmer to set-up simple loops to access these related items.

The following assembler directives can be used when setting up arrays:

- *DUP* initialize an area of memory with a repeated value or list of values.

- A variable defined as a byte can be initialized with the ASCII code of characters by surrounding the character string with single quote marks ('').

Array elements can be accessed by using displacement or by indexing.
EXAMPLE:

.model small
.stack 200h
.data
    message db 'Hello World!!!',$'
    buffer db 80 dup(?)
.code
.start:
    ; set-up ds to be able to access our data
    mov ax,@data
    mov ds,ax

    ; copy message to buffer (max 80 characters)
    mov si,0
    mov cx,80
.copy:
    mov al,message[si]
    mov buffer[si],al
    inc si
    cmp al,'$
    loopne copy
    je print

    ; message is too big - truncate it
    mov buffer[79],$

.print:
    ; use DOS interrupt 21h, Service 09h to print buffer
    ; (requires ascii string terminated by '$')
    lea dx,buffer
    mov ah,09h
    int 21h

.exit:
    ; Use DOS interrupt 21h, service 4ch to exit program
    mov ax,4c00h
    int 21h

.end start
Assembler Directives

- *TYPE* returns the number of bytes on each element of an array.

- *LENGTH* returns the number of data units on an array.

- *SIZE* returns the total number of bytes on an array (same as *TYPE * *LENGTH*).
EXAMPLE (same as before, but added directives):

.model small
.stack 200h
.data
    message db 'Hello World!!!'','$
    buffer db 80 dup(?)
.code
start:
    ; set-up ds to be able to access our data
    mov ax,@data
    mov ds,ax

    ; copy message to buffer (max 80 characters)
    mov si,0
    mov cx,length buffer
copy:
    mov al,message[si]
    mov buffer[si],al
    add si,type buffer
    cmp al,'$
    loopne copy
    je print

    ; message is too big - truncate it
    mov buffer[size buffer - 1],'$
print:
    ; use DOS interrupt 21h, Service 09h to print buffer
    ; (requires ascii string terminated by '$')
    lea dx,buffer
    mov ah,09h
    int 21h
exit:
    ; Use DOS interrupt 21h, service 4ch to exit program
    mov ax,4c00h
    int 21h
end start
Array Exercise

Assume there is an array of employee pay scales. There are 100 employees and a 16-bit value is used to represent each employee pay scale. Write a program that adds 50 dollar to each employee. Use assembler directives (TYPE, LENGTH, and SIZE) as much as possible.
Structures

Structures provide a method to group data that may not be of the same type.

*STRUC* is an assembler directive used to indicate a structure definition (not memory allocation).

*ENDS* indicates the end of the structure.

<, > are used to allocate memory and initialize values.

The dot operator (.) is used to access members of a structure.

Nested structures can be defined.
Structure Example

.model small

Personal_Data struc
  initials db 2 Dup(?)
  age dw ?
  weight dw ?
.ends

Employee struc
  employee_number dd ?
  data Personal_Data <?,?,?> ;nested struc
.ends

.stack 200h

.data
  john_doe Personal_Data <'JD',26,150>
  friends Personal_Data 100 Dup(<>)
  employee1 Employee  <12345678h>

.code
  ...
  ...
  mov  ax,john_doe.age
  ...
  ...
  mov  employee1.data.age,39
  ...
  ...
  mov  friends[bx].initials[si],'L'
  ...
  ...
.end start
Structure Exercise

Define & Allocate storage for an array of 100 structures. Each should contain:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>last_name</td>
<td>10 bytes</td>
</tr>
<tr>
<td>first_name</td>
<td>10 bytes</td>
</tr>
<tr>
<td>middle_init</td>
<td>1 byte</td>
</tr>
<tr>
<td>division</td>
<td>1 Word</td>
</tr>
<tr>
<td>dept</td>
<td>1 Word</td>
</tr>
</tbody>
</table>

Write a program to make each employee’s division equal to 14.