THE UNIVERSITY OF ARIZONA
UA South

CYBV 471 Assembly Programming for Security Professionals
Week 14

Data Structures-1

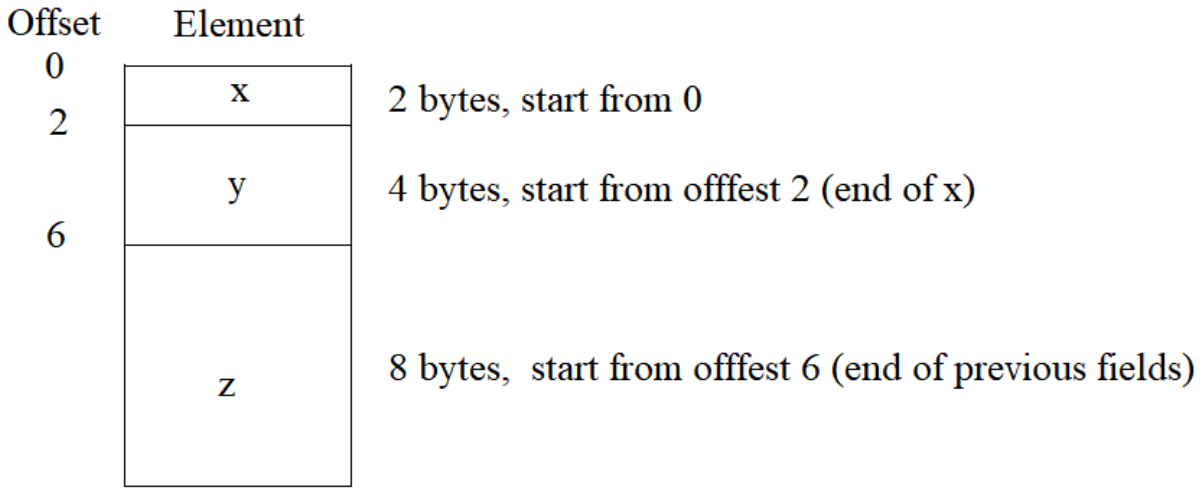
Agenda

- **Data Structure**
  - What is a data structure?
  - Define Structures
  - Declare Structures
  - Referencing Structure Variables
  - Access Structure's Members
  - Aligning Structure Fields
  - ALLGN Instruction
  - Structures Containing Structures
  - Array of Structures

What is a data structure?

- It is used to define and group different related data types into one variable (structure type)
- You can consider a structure as an array with elements of different data types and sizes
- The variables in a structure are called *fields*
- Program statements can access the structure as a single entity or access individual fields
- Example

```
struct   st1 {
        short int x;        // two bytes integer
        int         y;      // four bytes integer
        double    z;        // 8 bytes integer
    };
```



What is a data structure?

- Structures provide an easy way to cluster data and pass it from one procedure to another
  - Instead of passing multiple variables separately, they can be passed as a single unit.

- To access an element within structure, we need to know the starting address of the structure and the relative offset of that element from the beginning of the structure.

- Unlike an array where this offset can be calculated by the index of the element, the element of a structure is assigned an offset by the compiler.

- Using a structure involves three sequential steps:
  1. Define the structure.
  2. Declare one or more variables of the structure type, called *structure variable*
  3. Write runtime instructions that access the structure fields.

Defining Structures in Assembly

- A structure is defined using the STRUC and ENDSTRUC directives above data section
- Global structure (above data section)
- Inside the structure, you define fields as ordinary variables.

```
STRUC name
        .field-1:  RESx    ; reserve x bytes for field-1
        .field-2:  RESy    ; reserve y bytes for field-2
         ----
        .field-n: RESz     ; reserve z bytes for field-n
ENDSTRUC
```

- EXAMPLE

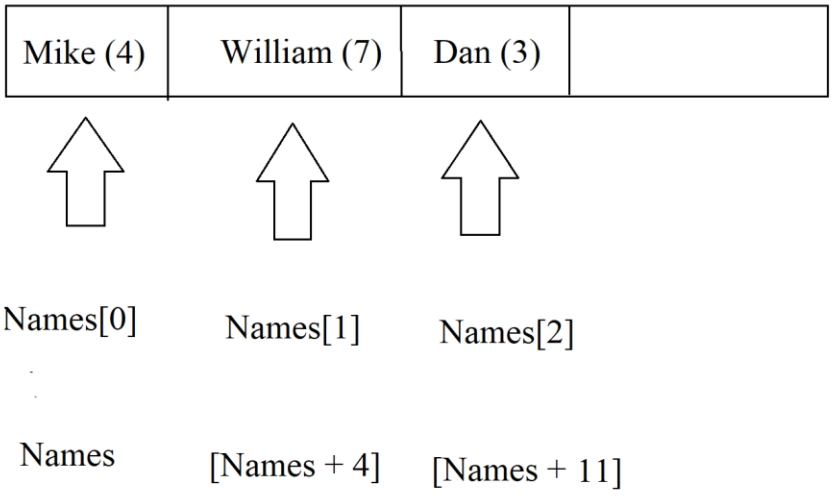
```
STRUC Point          ; define point (name) structure
    .x:  RESD 1      ; reserve 4 bytes for x direction
    .y:  RESD 1      ; reserve 4 bytes for y direction
ENDSTRUC
```

Defining Structures

Name of the array is a pointer to the first element in the array
Name of the structure is a pointer to the first element in the structure

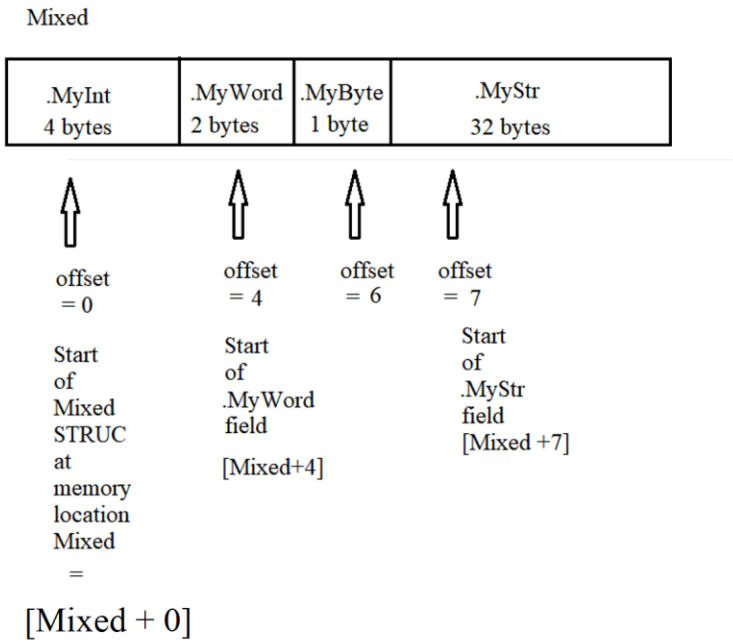
Array

Names = ["Mike", "William", " Dan"]



Structure

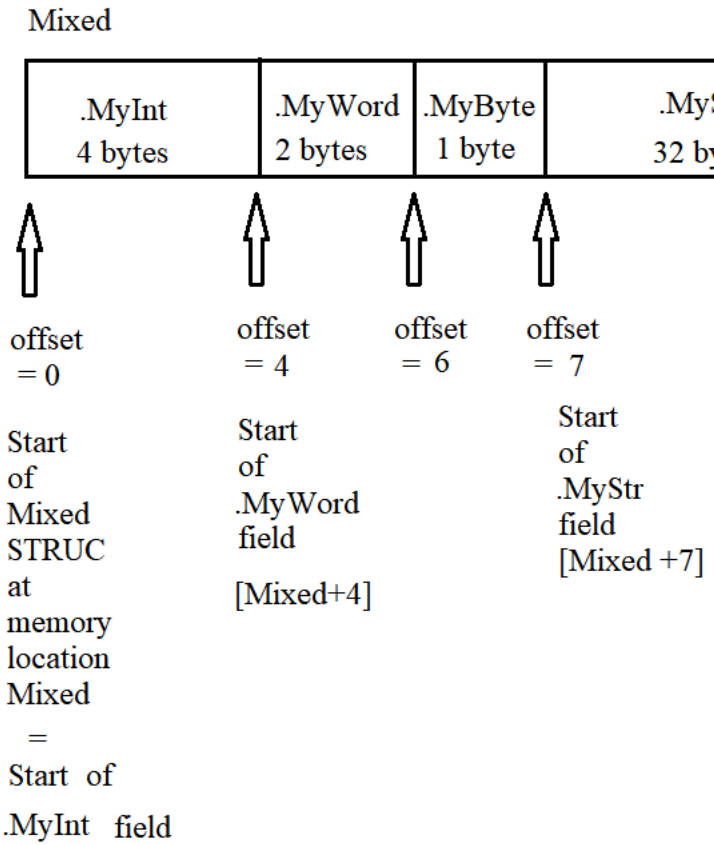
```
STRUC Mixed       ; define Mixed structure
    .MyInt:    resd  1    ; reserve 4 bytes for my_int field
    .MyWord:   resw  1    ; reserve 2 bytes for my_word field
    .MyByte:   resb  1    ; reserve 1 byte for my_byte field
    .MyStr:    resb  32   ; reserve 32 bytes for my_str field
ENDSTRUC
```



Defining Structures

- EXAMPLE

```
STRUC Mixed       ; define Mixed structure
    .MyInt:    resd  1    ; reserve 4 bytes for my_int field
    .MyWord:   resw 1     ; reserve 2 bytes for my_word field
    .MyByte:   resb  1    ; reserve 1 byte for my_byte field
    .MyStr:    resb  32   ; reserve 32 bytes for my_str field
ENDSTRUC
```



Size of the Structure

- EXAMPLE

```
STRUC Mixed       ; define Mixed structure
    .MyInt:     resd  1     ; reserve 4 bytes for my_int field
    .MyWord:    resw 1      ; reserve 2 bytes for my_word field
    .MyByte:    resb  1     ; reserve 1 byte for my_byte field
    .MyStr:     resb  32    ; reserve 32 bytes for my_str field
ENDSTRUC
```

- What is the size of the above structure?
- Size = 4 bytes + 2 + 1 + 32 = 39 bytes
- The value of structure size is saved in "Mixed.size" symbol
- In general, the size of a structure "MyStruc" is saved in "MyStruc.size" symbol
- The above structure has 6 symbols: Structure name (Mixed), 4 fields (.MyInt, .MyWord, .MyByte, and .MyStr), and Mixed.size.

Declare Structures

- After defining a global structure, you can declare instances of it and initialize its fields in the data section

```
STRUC Point          ; define point (global name) structure
                .x:  RESD 1       ; reserve 4 bytes for x direction
                .y:  RESD 1       ; reserve 4 bytes for y direction
 ENDSTRUC


section .data
      P1: ISTRUC Point          ;declare an instance of point structure and initialize its fields
      AT  Point.x,  dd 5        ; initialize the fields with different values
      AT  Point.y,  dd 7
     IEND                       ; end of initialization
```

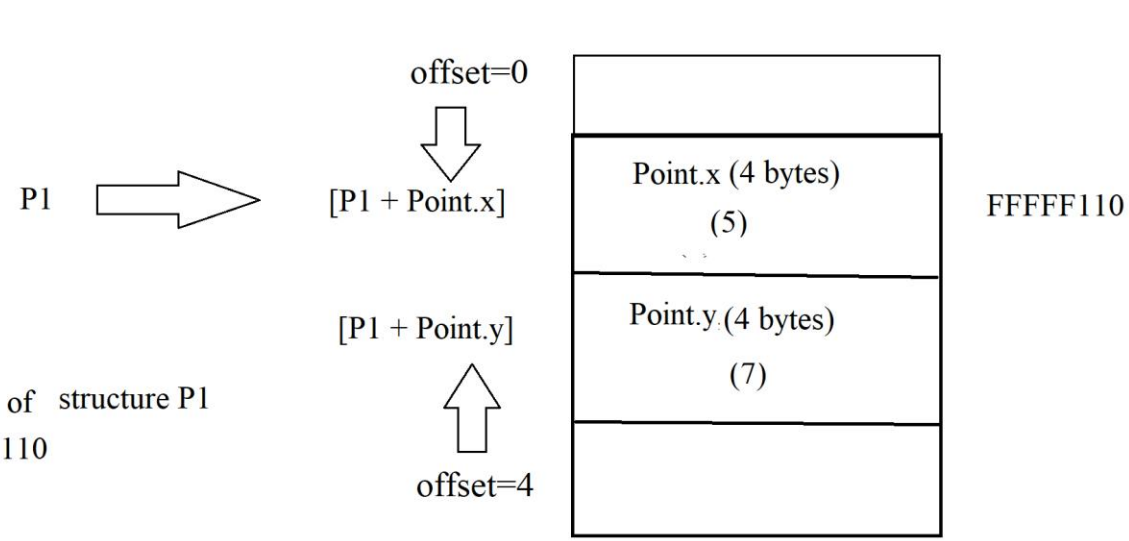
- How could you reserve memory space for structure in .bss section?

```
section    .bss
      P2: RESB  Point.size
```

Referencing (Accessing) Structure Variables

```
P1: ISTRUC Point
                at  Point.x,  dd 5          ; initialize the fields with different values
                at  Point.y,  dd 7
              IEND                          ; end of initialization
section .text
 mov eax, P1                    ; eax = address of structure = name of the instance = P1
                                ; points to the first element in the structure
```

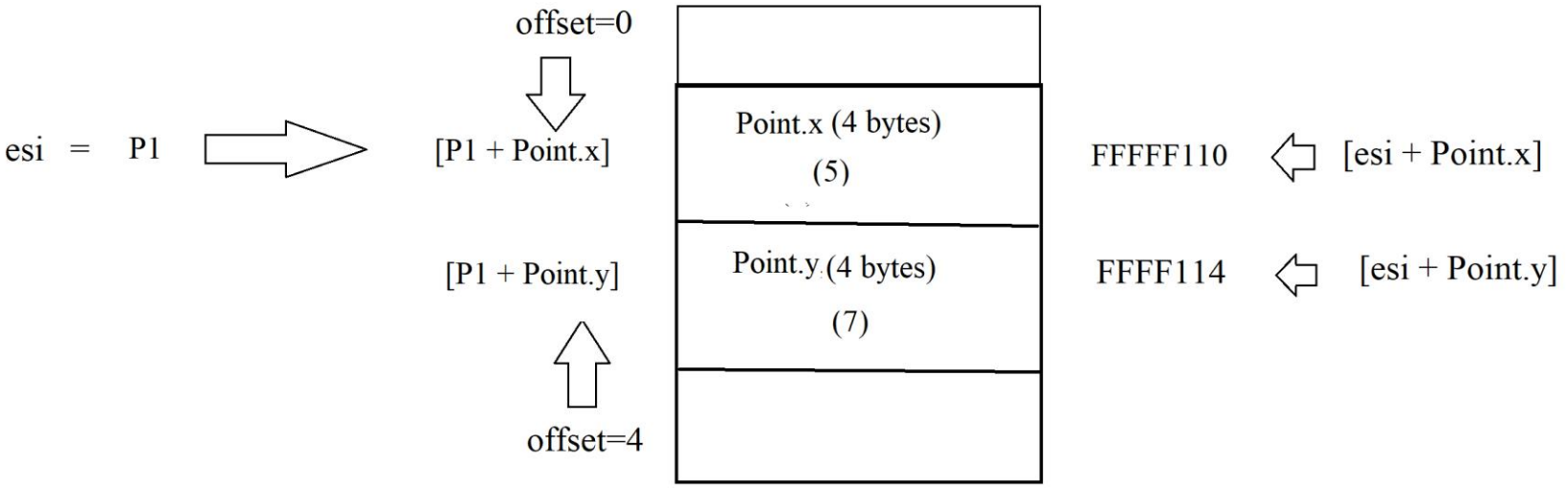
```
P1: ISTRUC Point
      at  Point.x,  dd 5
      at  Point.y,  dd 7
    IEND
```




```
section .text
      mov eax, P1
                ; eax = address of   structure P1
                ; eax = FFFFF110

    section .text
    mov eax, [P1 + Point.x]      ; eax = 5

    mov eax, [P1 + Point.y]      ; eax = 7
```

Referencing Structure Variables

```
P1: ISTRUC Point
      at  Point.x,  dd 5
      at  Point.y,  dd 7
    IEND
```




```
section .text

    mov eax,   [P1 + Point.x]    ; eax = 5

    mov eax,   [P1 + Point.y]    ; eax = 7
```

```
section .text

    mov esi, P1

    mov eax,  [esi + Point.x]    ; eax = 5

    mov eax,  [esi + Point.y]    ; eax = 7
```

Structure Example1

```
; STRU1.asm    create a structure

STRUC Point          ; define point structure
   .x:  RESD 1       ; reserve 4 bytes for x coardinate
   .y:  RESD 1       ; reserve 4 bytes for y coardinate
   .size:
ENDSTRUC

SECTION .data        ; Data section
  msg1:     db "The point coardinate is ",10, 0
  msgL1:    equ $-msg1

  msg2:     db "The new point coardinate is ",10, 0
  msgL2:    equ $-msg2

  msg3:     db "Size of the Point strucure is ",10, 0
  msgL3:    equ $-msg3

  msg4:     db "Memory locations for P1 strucure, offest for X and Y ",10, 0
  msgL4:    equ $-msg4

  msg5:     db "Memory locations for P1 strucure, X and Y ",10, 0
  msgL5:    equ $-msg5

 ;declare an instance of point structure and intialize its fields
 P1: ISTRUC Point
   AT  Point.x,  dd 5          ; initilaize the fields with different values
   AT  Point.y,  dd 7
IEND
```

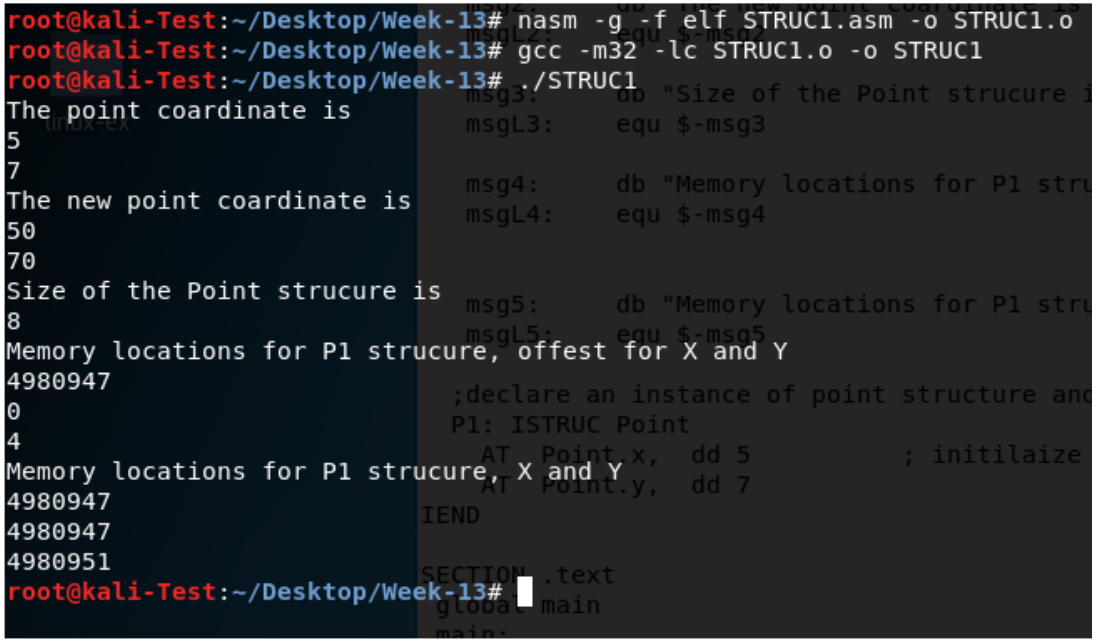
```
SECTION .text
 global main
 main:
       push ebp
       mov ebp, esp

       mov ecx,msg1                ; print coardinate message
       mov edx,msgL1
       call PString

       mov eax,  [P1 + Point.x]    ; Point.x is an offset w.r.t P1
       call printDec
       call  println

       mov eax, [P1 + Point.y]     ; Point.y is an offset w.r.t P1
       call printDec
       call  println

      ; Change the point values
       mov ecx,msg2                ; print coardinate message
       mov edx,msgL2
       call PString

       mov esi, P1                 ; indirect access
       mov word [esi + Point.x], 50   ; indirect access
       mov eax,  [esi + Point.x]
       call printDec
       call  println

       mov word [P1 + Point.y], 70    ; direct access
       mov eax, [P1 + Point.y]
       call printDec
       call  println
```

Structure Example1

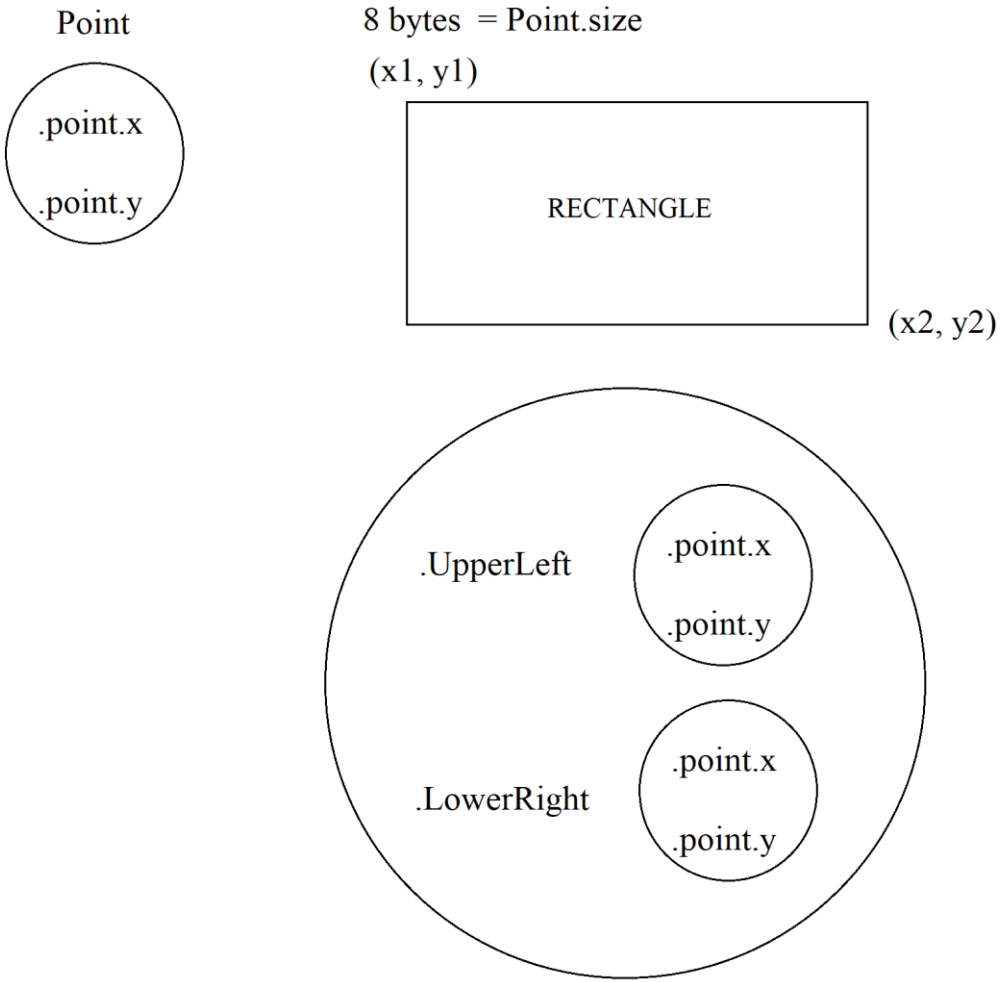
```
mov ecx,msg3                ; print size message
mov edx,msgL3
call PString

mov eax,  Point.size
call printDec
call  println

mov ecx,msg4                ; print offset message
mov edx,msgL4
call PString

 mov eax,  P1
 call printDec
 call  println

 mov eax,  Point.x
 call printDec
 call  println

 mov eax,  Point.y
 call printDec
 call  println
```

```
mov ecx,msg5                ; print memory message
mov edx,msgL5
call PString

 mov eax,  P1
 call printDec
 call  println

 mov eax,  P1 + Point.x
 call printDec
 call  println

 mov eax,  P1 + Point.y
 call printDec
 call  println

 ; exit the program and cleaning
 mov esp, ebp
 pop ebp
 ret
```

Structure Example1



Structures Containing Structures



Structures Containing Structures

- *Structures* can contain instances of other structures.
- For example, a **RECTANGLE** structure can have two instances of POINT structures that represent its upper-left and lower-right corners

```
STRUC RECTANGLE                          ; define new structure
        .UpperLeft:    RESB Point.size     ; instance of Point structure
        .LowerRight: RESB Point.size       ; instance of Point structure
ENDSTRUC
```

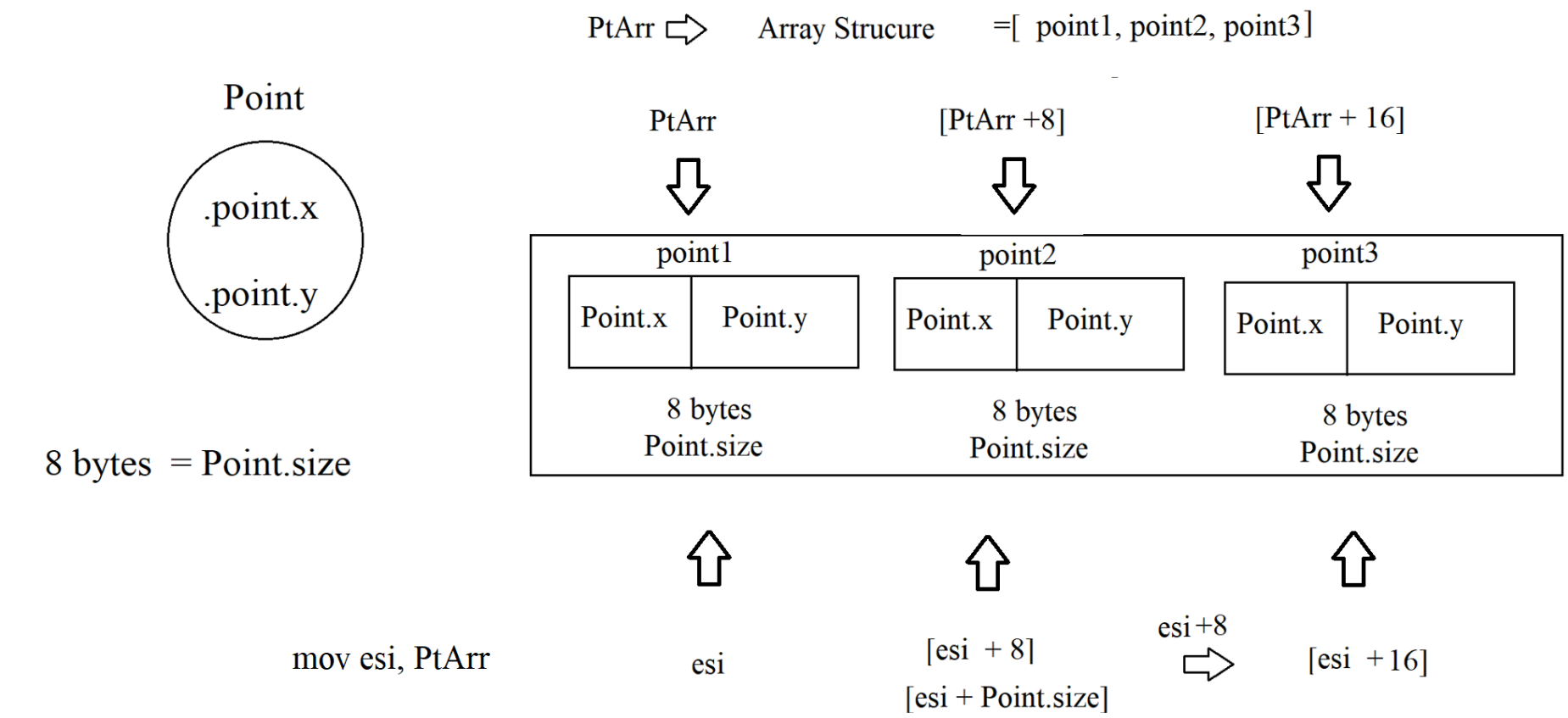
- Initialize the Rectangle structure

```
rect1: ISTRUC RECTANGLE
        AT RECTANGLE.UpperLeft. Point.x,  dd 10      ; initialize the x field
        AT RECTANGLE.UpperLeft. Point.y,  dd 15      ; initialize the x field
        AT RECTANGLE.LowerLeft. Point.x,  dd 20      ; initialize the x field
        AT RECTANGLE.LowerLeft. Point.y,  dd 25      ; initialize the x field
        IEND
```

- Access and reference structure members

```
mov [rect1+RECTANGLE.UpperLeft. Point.x], 35      ; direct access
mov esi,  rect1
mov [esi + RECTANGLE.UpperLeft. Point.x ], 35     ; indirect operand
```

Array of Structures



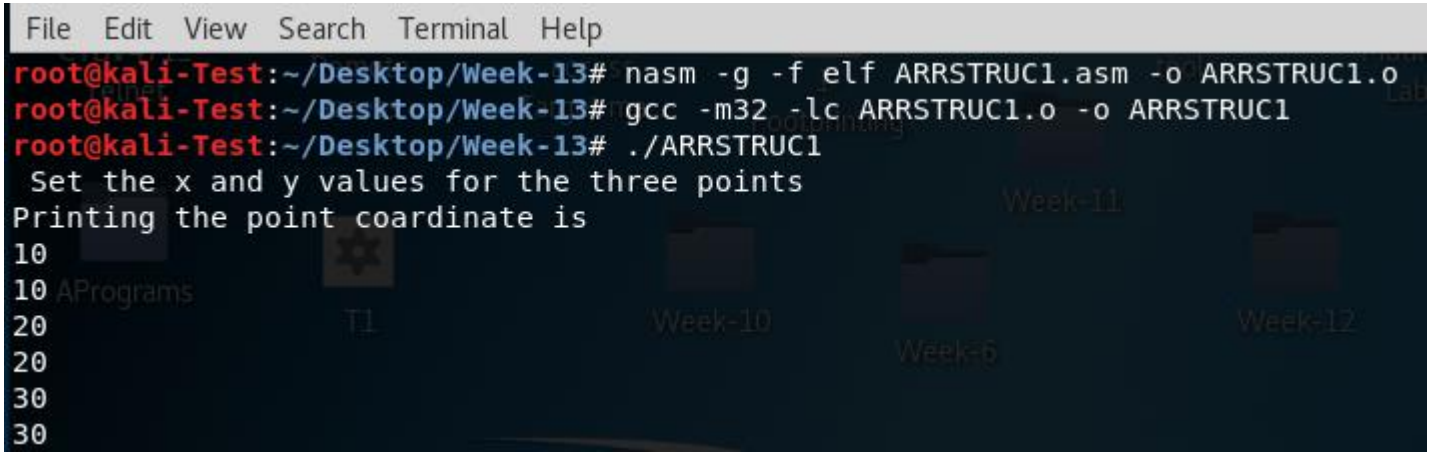
Array of Structures

```
; ARRSTRU1.asm    create an array of structures
 STRUC Point          ; define point structure
    .x:  RESD 1       ; reserve 4 bytes for x coardinate
    .y:  RESD 1       ; reserve 4 bytes for y coardinate
    .size:
 ENDSTRUC
SECTION .data        ; Data section
  msg1:     db " Set the x and y values for the three points ",10, 0
  msgL1:    equ $-msg1

  msg2:     db "Printing the point coardinate is ",10, 0
  msgL2:    equ $-msg2

 ;declare an instance of point structure and intialize its fields
 P: ISTRUC Point
     AT  Point.x,  dd 0          ; initilaize the fields with different values
     AT  Point.y,  dd 0
IEND

SECTION .bss        ; bss section
PtArr:  RESB Point.size*3      ; reserve place for 3 structures.
                               ; PtArr is a pointer points to the start of all arrays
ArrCount:  EQU ($ - PtArr) / Point.size       ; should be 3 structures

SECTION .text
 global main
 main:
       push ebp
       mov ebp, esp
       mov  ecx, ArrCount       ; ecx has the count of array elements
       mov  esi, PtArr          ; esi has the address of first structure in the arraay
       mov ecx,msg1
       mov edx,msgL1
       call PString
```

Array of Structures

```
mov  WORD [esi + Point.x], 10    ; set x value for first point
mov  WORD [esi + Point.y], 10    ; set x value for first point
add  esi, Point.size             ; move to next structure in the array

mov WORD [esi + Point.x], 20         ; set x value for second point
mov WORD [esi + Point.y], 20         ; set x value for second point
add  esi, Point.size                 ; move to next structure in the array

mov WORD [esi + Point.x], 30         ; set x value for second point
mov WORD [esi + Point.y], 30         ; set x value for second point
add  esi, Point.size                 ; move to next structure in the array

; Print the  three point values using a loop

mov ecx,msg2                     ; print coardinate message
mov edx,msgL2
call PString

mov  ecx, ArrCount      ; ecx has the count of array elements
mov  esi, PtArr         ; esi has the address of first structure in the array

L1:
mov eax, [esi + Point.x]   ; indirect access to x value
call printDec
call  println
mov eax, [esi + Point.y]   ; indirect access to y value
call printDec
call  println
add  esi, Point.size                 ; move to next structure in the array
loop L1

; exit the program and cleaning
mov esp, ebp
pop ebp
ret
```

Array of Structures

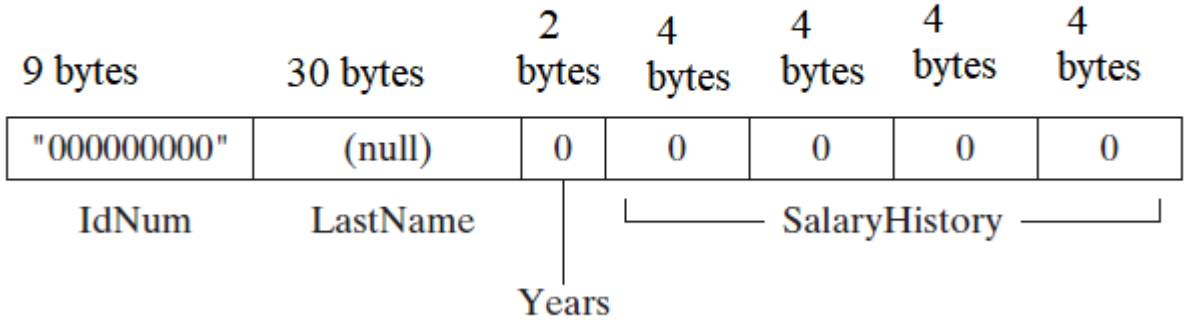


Aligning Structure Fields

- The following **Employee** structure describes employee information such as employee ID number, last name, years of service, and an array of last four years of salary values.

```
STRUC Employee
   IdNum BYTE  "000000000“          ; 9 bytes
   LastName  BYTE 30 DUP(0)         ; 30 bytes
  YearServ   WORD 0                 ; 2 bytes
   SalaryHistory DWORD 0,0,0,0      ; 4-element array, each is 4 bytes (total 16 bytes
ENDStruc
```

Linear representation of the structure's memory layout



Aligning Structure Fields

- Here is the memory layout for the previous structure

```
STRUC Employee
    IdNum BYTE  "000000000“          ; 9 bytes
    LastName  BYTE 30 DUP(0)         ; 30 bytes
   YearServ   WORD 0                 ; 2 bytes
    SalaryHistory DWORD 0,0,0,0      ; 4-element array, each is 4 bytes (total 16 bytes)
```

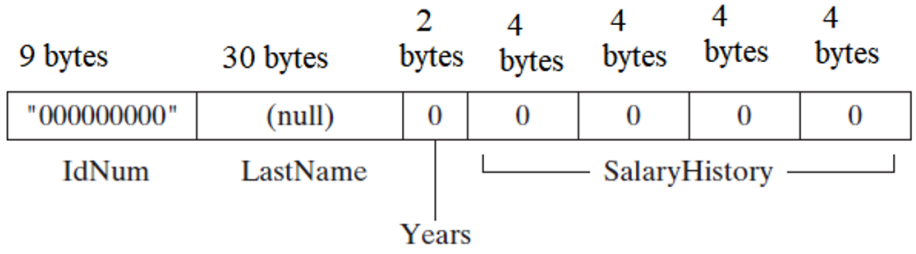
8 bytes layout

IdNum	0	0	0	0	0	0	0	0
LastName	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
Years	0	0	0	0	0	0	0	0
SalaryHistory[0]	0	0	0	0	0	0	0	0
SalaryHistory[2]	0	0	0	0	0	0	0	0
	0							

SalaryHistory[1]

SalaryHistory [3]

Linear representation of the structure's memory layout



ALLGN Instruction

- The ALIGN directive sets the address alignment of the next field or variable

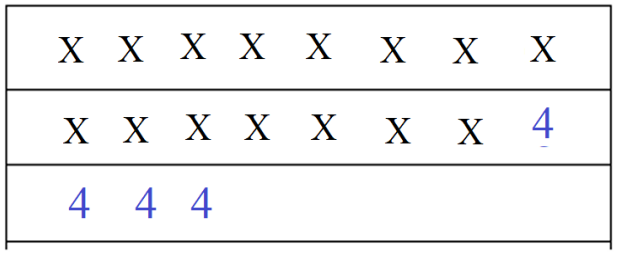
  ALIGN datatype

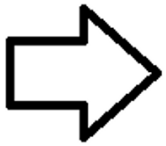
  variable datatype

- The following, for example, aligns **Var1** to a doubleword boundary (4 bytes):

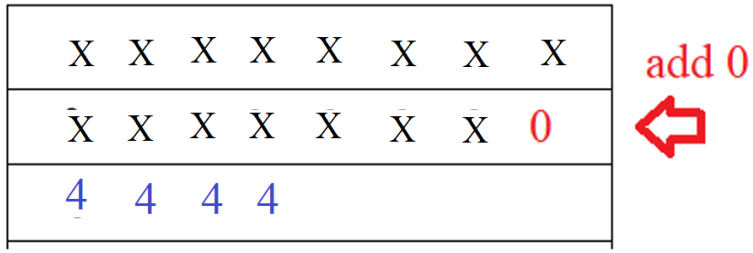
```
section .data
 ALIGN DWORD
 Var1 DWORD 0
```

Var1 DWORD 4444





ALIGN DWORD
Var1 DWORD 4444



Aligning Structure Fields

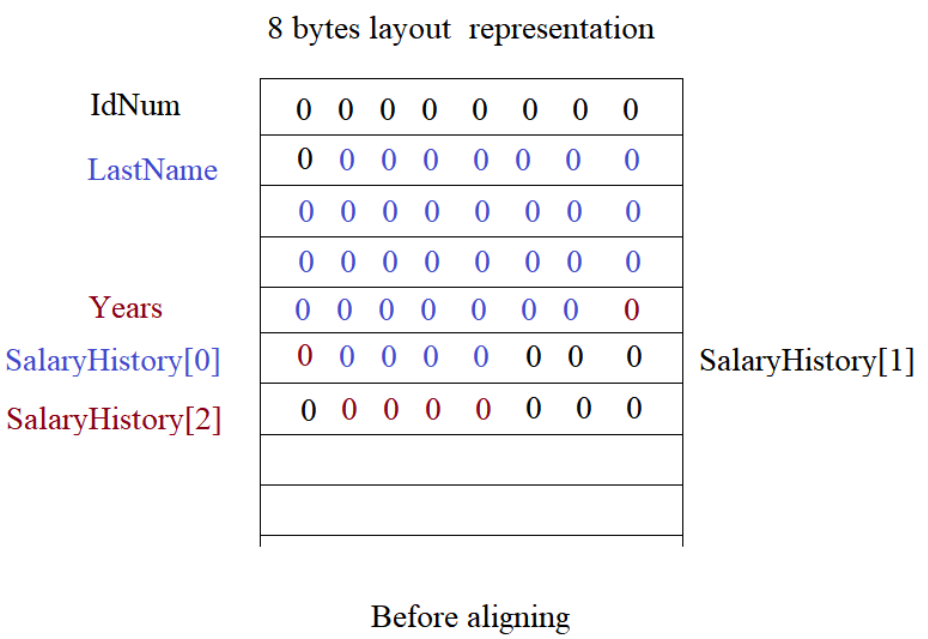
- Before memory aligning

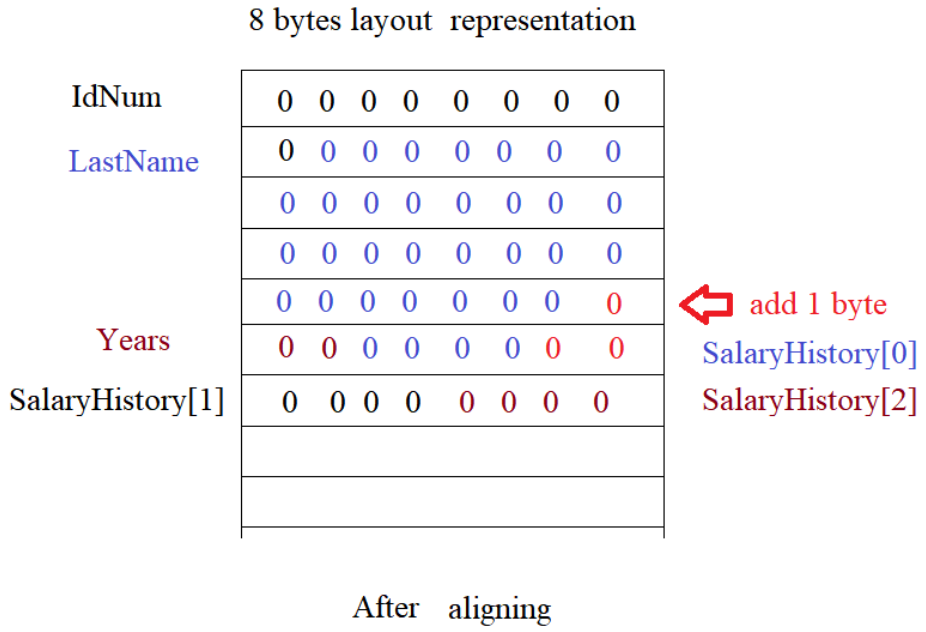
```
STRUC Employee
    IdNum BYTE  "000000000“          ; 9 bytes
    LastName  BYTE 30 DUP(0)         ; 30 bytes
   YearServ   WORD 0                 ; 2 bytes
    SalaryHistory DWORD 0,0,0,0      ; 4-element array, each is 4 bytes (total 16 bytes)
ENDSTRUC                             ;  total memory = 57 bytes
```

After aligning

```
STRUC Employee
     IdNum BYTE  "000000000“          ; 9 bytes
     LastName  BYTE 30 DUP(0)         ; 30 bytes
    ALIGN WORD                        ;  one byte added
    YearServ   WORD 0                 ; 2 bytes
     ALIGN DWORD                      ;  two bytes added
    SalaryHistory DWORD 0,0,0,0       ; 4-element array, each is 4 bytes (total 16 bytes)
ENDSTRUC                              ;  total memory = 60 bytes
```

Aligning Structure Fields





Data Type	Alignment
BYTE	Align on 1 byte boundary
WORD	Align on 2 bytes boundary
DWORD	Align on 4 bytes boundary
QWORD	Align on 8 bytes boundary
REAL4	Align on 4 bytes boundary
REAL8	Align on 8 bytes boundary

Putting It All Together

You should know:

- **Data Structure**
  - What is a data structure?
  - Define Structures
  - Declare Structures
  - Referencing Structure Variables
  - Access Structure's Members
  - Structures Containing Structures
  - Array of Structures
  - Aligning Structure Fields
  - ALLGN Instruction

Questions?

Coming Next Week
Data Structures-2

Week 14 Assignments

- **Learning Materials**
  1- Week 14 Presentation
  2- NASM manual, PP: 68-71

- **Assignment**
  1- Complete “Lab 14” by coming Sunday 11:59 PM.