Computer Org. & Assembly Language Lab

Lab#14: Structures and Macros

Agenda

- Structures
    - Nested Structures
- Unions
- Macros

Structures

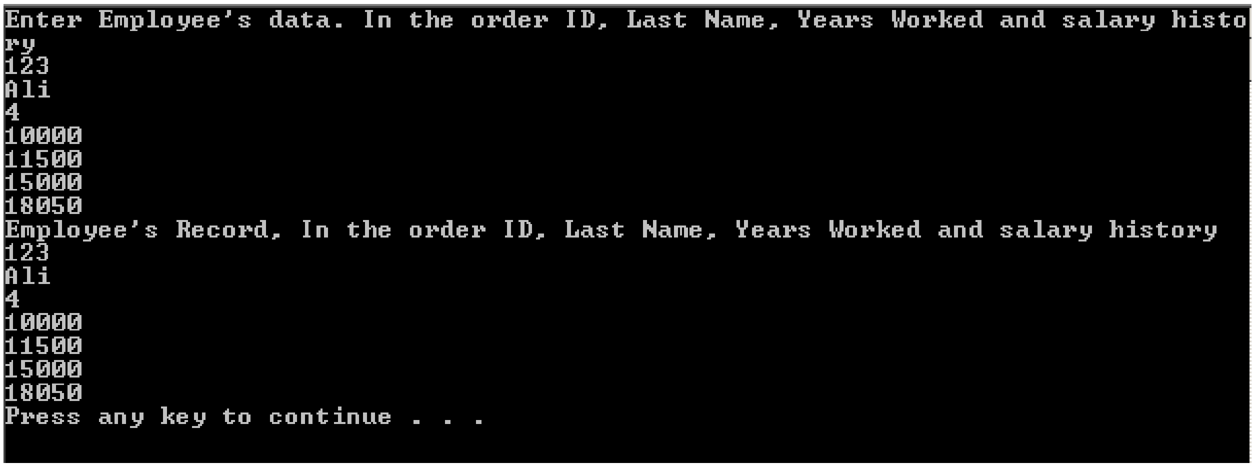
A structure is defined using the STRUCT and ENDS directives. Inside the structure, you define fields using the same syntax as for ordinary variables. The basic syntax is:

```
name STRUCT

      field-declarations

name ENDS
```

Structures can contain virtually any number of fields.

```
include irvine32.inc

Employee STRUCT
      IdNum BYTE "000000000"
      LastName BYTE 30 DUP(0)
      Years WORD 0
      SalaryHistory DWORD 0 ,0 ,0 ,0
Employee ENDS

.data
      message1 byte "Enter Employee's data. In the order ID, Last Name, Years
Worked and salary history",0
      message2 byte "Employee's Record, In the order ID, Last Name, Years
Worked and salary history", 0

      person1 Employee <"555223333">
      person2 Employee {"555223333"}
      person3 Employee <, "Jones ">
      person4 Employee <,,,2 DUP (20000)>
      person5 Employee<,,5>
.code
main PROC
      mov edx, offset message1
      call writestring
      call crlf

      xor eax,eax

      ;Enter ID
      mov edx, offset person1.IdNum
      mov ecx, (sizeof person1.IdNum)
      call readstring

      ;Enter LastName
      mov edx, offset person1.LastName
      mov ecx, (sizeof person1.LastName)
      call readstring

      ;Enter Years Worked
      call readint
      mov person1.Years, ax

      ;Enter Salary History
      mov cx, 4
      mov esi, offset person1.SalaryHistory

      L1:
            call readint
            mov DWORD PTR [esi], eax
            add esi,4
      Loop L1

      mov edx, offset message2
      call writestring
      call crlf

      ;Show Id
      mov edx, offset person1.IdNum
      call writestring
      call crlf

      ;Show LastName
      mov edx, offset person1.LastName
      call writestring
      call crlf

      ;Show Years worked
      mov ax, person1.years
      call writedec
      call crlf

      ;Show Salary history
      mov cx, 4
      mov esi, offset person1.SalaryHistory

      L2:
            mov eax, [esi]
            call writedec
            call crlf
            add esi,4
      Loop L2
exit
main ENDP
end main
```

Sample Output



```
Enter Employee's data. In the order ID, Last Name, Years Worked and salary histo
ry
123
Ali
4
10000
11500
15000
18050
Employee's Record, In the order ID, Last Name, Years Worked and salary history
123
Ali
4
10000
11500
15000
18050
Press any key to continue . . .
```

Indirect Operands

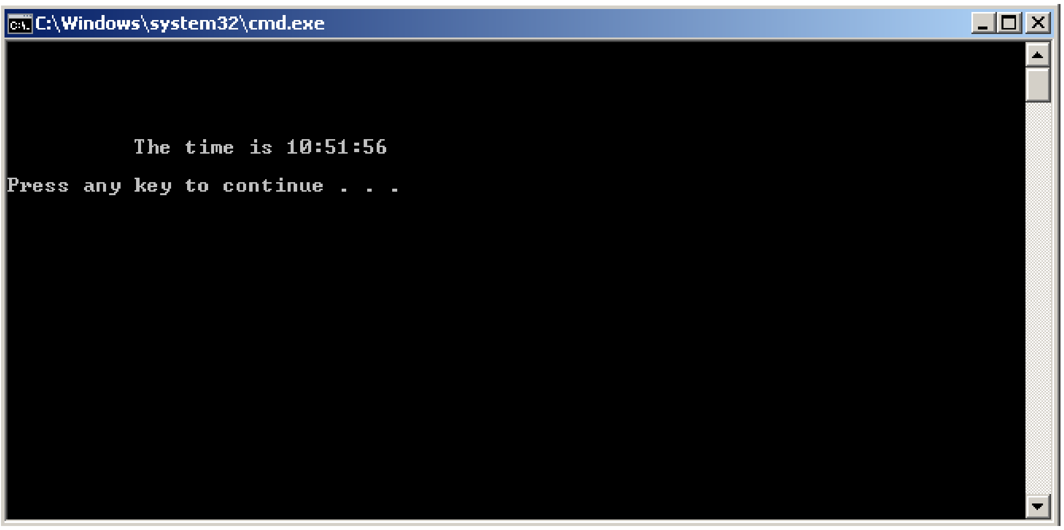
Indirect operands permit the use of a register (such as ESI) to address structure data. Such addressing provides flexibility, particularly when passing a structure's address to a procedure , or when using an array of structures . The PTR operator is required when referencing indirect operands :

```
mov esi ,OFFSET person1
mov ax, (Employee PTR [esi]). Years
```

The following program (ShowTime.asm) retrieves the system time and displays it at a selected screen location

```
TITLE Showing the Time                (ShowTime.ASM)

; This program locates the cursor and displays the
; system time. It two uses MS-Windows structures.

INCLUDE Irvine32.inc

Comment @

Definitions copied from Irvine32.inc (SmallWin.inc):

COORD STRUCT
  X WORD ?
  Y WORD ?
COORD ENDS

SYSTEMTIME STRUCT
    wYear WORD ?
    wMonth WORD ?
    wDayOfWeek WORD ?
    wDay WORD ?
    wHour WORD ?
    wMinute WORD ?
    wSecond WORD ?
    wMilliseconds WORD ?
SYSTEMTIME ENDS
---------------------------------------- @

.data
X = 10
Y = 5
sysTime SYSTEMTIME <>
consoleHandle DWORD ?
colonStr BYTE ":",0
TheTimeIs BYTE "The time is ",0

.code
main PROC
      mov dh, Y
      mov dl, X

      call Gotoxy

      INVOKE GetLocalTime,ADDR sysTime

      mov   edx,OFFSET TheTimeIs          ; "The time is "
      call  WriteString

      ; Display the system time (hh:mm:ss).
      movzx eax,sysTime.wHour             ; hours
      call  WriteDec
      mov   edx,offset colonStr           ; ":"
      call  WriteString
      movzx eax,sysTime.wMinute           ; minutes
      call  WriteDec
      mov   edx,offset colonStr           ; ":"
      call  WriteString
      movzx eax,sysTime.wSecond           ; seconds
      call  WriteDec

      call Crlf
      call Crlf
      exit
main ENDP
END main
```

Output



```
         The time is 10:51:56

Press any key to continue . . .
```

Gotoxy PROC

Locates the cursor at a given row and column in the screen's console buffer. The values passed in DH and DL can range from 0 to X-1 and from 0 to Y-1, where X is the number of columns and Y is the number of rows in the console buffer.The default window size is 80 x 25 but it can be set to a different size.

Use the GetMaxXY procedure to obtain the size of the screen's console buffer.   If the console buffer is set larger than the display window, the window repositions itself automatically to display the cursor position.

```
Call args:  DL = column
            DH = row

Return arg: None

Example: Locate the cursor in the lower right
         corner of an 80 x 25 screen.

      mov  dl,79  ;column
      mov  dh,24  ;row
      call Gotoxy
```

Nested Structures

You can create nested structure definitions, where structures contain other structures. For example, a Rectangle can be defined in terms of its upper-left and lower-right corners, both COORD objects:

```
Rectangle STRUCT
      UpperLef t COORD <>
      LowerRight COORD <>
```

```
Rectangle ENDS
```

```
rect1 Rectangle < >
rect2 Rectangle { }
rect3 Rectangle { {10,10}, {50, 20} }
rect4 Rectangle < <10 ,10> , <50,2 0> >
```

The following is a direct reference to a nested structure field:

```
mov rectl. UpperLeft. X, 10
```

Using an indirect operand, you can access a nested field . In the following example, we move 10 to the Y coordinate of the upper-le ft corner of the structure pointed to by ESI:

```
mov esi,OFFSET rect1
mov (Rectangle PTR [es i] ) .UpperLeft .Y, 10
```

The OFFSET operator can be used to return pointers to individual structure fields , including nested fields:

```
mov edi, OFFSET r ect2.LowerRight
mov (COORD PTR (edi]). X, 50
mov edi, OFFSET rect2.LowerRight. X
mov WORD PTR (edi ], 50
```

Unions

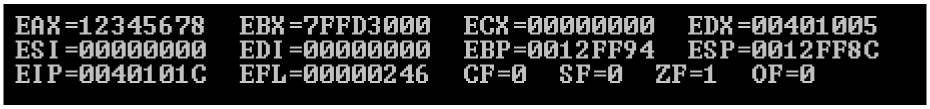
Whereas each field in a structure has an offset relative to the first byte of the structure, all the fields in a union start at the same offset. The storage size of a union is equal to the length of its longest field. When not part of a structure, a union is declared using the UNION and ENDS directives:

```
unionname UNION
      union-fields
unionname ENDS
```

```
INCLUDE Irvine32.inc

Integer UNION
D DWORD 0
W WORD 0
B BYTE 0
Integer ENDS


.data
val1 Integer <12345678h>
va12 Integer <100h >
va13 Integer <>
.code
```

```
main PROC

      Xor eax,eax
      mov eax, val1.D
      call dumpregs

exit
main ENDP
END main
```

Output



```
EAX=12345678  EBX=7FFD3000  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=0040101C  EFL=00000246  CF=0  SF=0  ZF=1  OF=0
```

Macros

A macro procedure is a named block of assembly language statements. Once defined, it can be invoked (called) as many times in a program as you wish. When you invoke a macro procedure, a copy of its statements is inserted directly into the program. It is customary to refer to calling a macro procedure, although technically there is no CALL instruction involved.

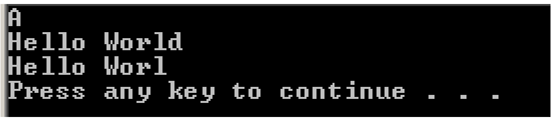
Syntax

```
macroname MACRO [parameter-1, parameter-2 ... ]
      statement-list
ENDM
```

```
include irvine32.inc

mputchar MACRO char
      mov eax, char
      call WriteChar
      call crlf
ENDM

mReadStr MACRO varName, size
      push ecx
      push edx
      mov edx,OFFSET varName
      mov ecx, size
      call ReadString
      pop edx
      pop ecx
ENDM

mWriteStr MACRO string
      mov edx, OFFSET string
      call WriteString
      call crlf
ENDM

.data
msg BYTE ?

.code
main PROC

      mputchar 'A'
      mReadStr msg,10
      mWriteStr msg
```

```
exit
main ENDP
END main
```

Output



```
A
Hello World
Hello Worl
Press any key to continue . . .
```