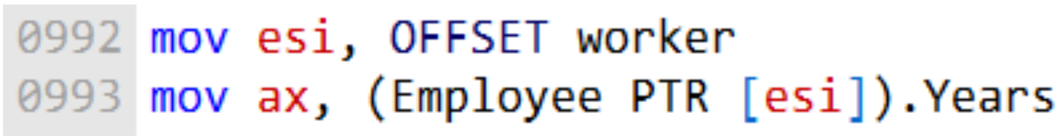
Indirect and Indexed Operands Structs

Indirect Operands

Indirect operands are used to address memory locations through the contents of a register. This can be useful for accessing structure members, since it allows you to use a register to store the address of the structure.

To use an indirect operand, you need to use the **PTR operator.** The PTR operator tells the assembler that the operand is indirect.

For example, the following code uses an indirect operand to access the Years field of the worker structure variable:



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

This code first moves the address of the worker structure variable into the esi register.

Then, it uses the PTR operator to tell the assembler that the [esi] operand is indirect.

This means that the assembler will load the contents of the esi register and use that as the address of the Years field.

Indexed Operands

Indexed operands are used to address memory locations by adding an offset to the contents of a register.

This can be useful for accessing arrays of structures, since it allows you to use a register to store the index of the element in the array.

To use an indexed operand, you need to use the [register + offset] syntax.

The register is the register that contains the index of the element in the array.

The offset is the offset of the structure field from the beginning of the structure.

For example, the following code uses an indexed operand to access the Years field of the employee in index position 1 of the department array:

```
.data
    department Employee 5 DUP(<>)
.code
    mov esi, TYPE Employee
    ; index = 1
    mov department[esi].Years, 4
```

This code first moves the size of the Employee structure into the esi register.

Then, it uses the [esi] syntax to access the element in index position 1 of the department array. Finally, it sets the Years field of that element to 4.

Looping through an Array

Indirect and indexed operands can be used to loop through an array of structures.

The following code shows how to loop through the AllPoints array and assign coordinates to each element:

```
; Loop Through Array
(AllPoints.asm)
INCLUDE Irvine32.inc
NumPoints = 3
.data
    ALIGN WORD
    AllPoints COORD NumPoints DUP(<0,0>)
.code
    main PROC
    mov edi, 0
    ; array index
    mov ecx, NumPoints
    ; loop counter
    mov ax, 1
    ; starting X, Y values
    L1:
    mov (COORD PTR AllPoints[edi]).X, ax
    mov (COORD PTR AllPoints[edi]).Y, ax
    add edi, TYPE COORD
    inc ax
    loop L1
    exit
    main ENDP
    END main
```

This code first moves the value 0 into the edi register. This will be the array index. Then, it moves the value of the NumPoints variable into the ecx register. This will be the loop counter.

Next, the code moves the value 1 into the ax register. This will be the starting X and Y values for the coordinates.

Then, the code enters a loop. Inside the loop, it first uses an indirect operand to access the X field of the element in index position edi of the AllPoints array. It then sets the X field to the value in the ax register.

Next, the code uses an indirect operand to access the Y field of the element in index position edi of the AllPoints array. It then sets the Y field to the value in the ax register.

After that, the code increments the edi register by the size of the COORD structure. This will move the array index to the next element in the array.

Finally, the code increments the ax register by 1. This will increment the X and Y values for the next coordinates.

The loop will continue until the ecx register reaches zero. At that point, the loop will exit and the program will terminate.

Conclusion

Indirect and indexed operands are powerful tools that can be used to access structure members and arrays of structures.