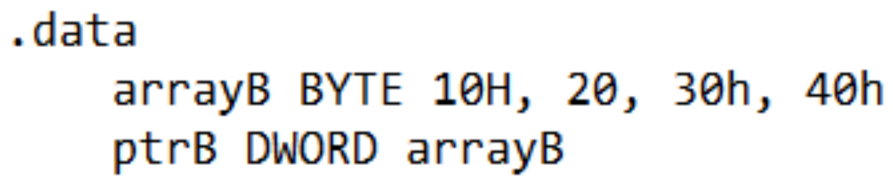
Pointers In ASM

A pointer in assembly language is a variable that stores the memory address of another variable or data structure.

Pointers are a powerful tool for manipulating arrays and data structures because they allow you to access and modify data at runtime by working with memory addresses.

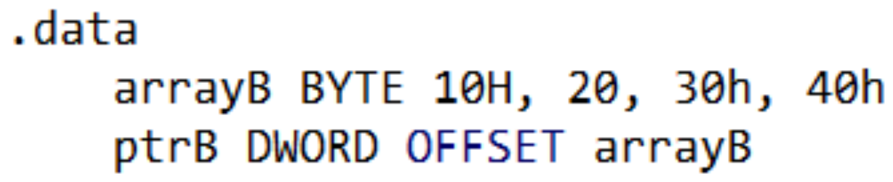
Pointer Size:

The size of a pointer in assembly language depends on the processor's current mode, which can be 32-bit or 64-bit. In the following example, we'll focus on 32-bit code.



```
.data
    arrayB BYTE 10H, 20, 30h, 40h
    ptrB DWORD arrayB
```

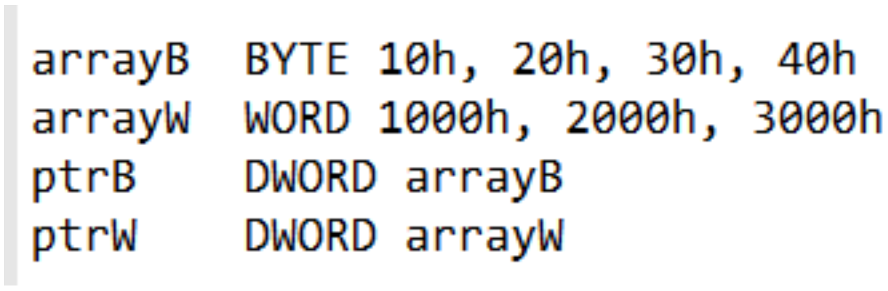
In this code snippet, we have an array called arrayB, which contains four bytes of data. The ptrB variable is declared as a doubleword (32 bits) and stores the memory offset of arrayB. Optionally, you can use the OFFSET operator to make the relationship between the pointer and the array clearer:



```
.data
    arrayB BYTE 10H, 20, 30h, 40h
    ptrB DWORD OFFSET arrayB
```

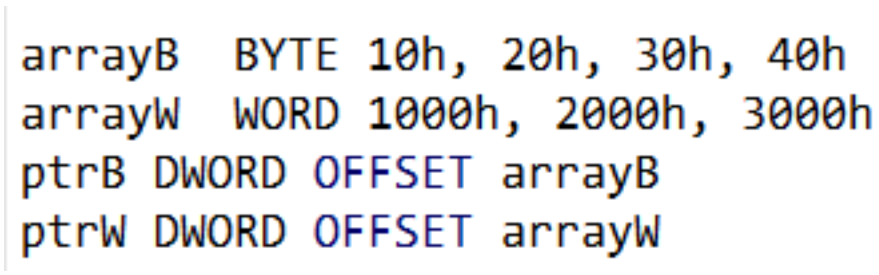
Near Pointers

In 32-bit assembly code, we often use near pointers, which are stored in doubleword variables. For example:



```
arrayB  BYTE 10h, 20h, 30h, 40h
arrayW  WORD 1000h, 2000h, 3000h
ptrB    DWORD arrayB
ptrW    DWORD arrayW
```

Here, we have two arrays: arrayB of bytes and arrayW of words. We declare pointers ptrB and ptrW as doublewords to store the memory offsets of these arrays. Optionally, you can use the OFFSET operator for clarity:



```
arrayB  BYTE 10h, 20h, 30h, 40h
arrayW  WORD 1000h, 2000h, 3000h
ptrB DWORD OFFSET arrayB
ptrW DWORD OFFSET arrayW
```

Understanding Pointers

In high-level languages, the details of pointers are often abstracted because they can vary between different machine architectures.

However, in assembly language, we deal with pointers at a more physical level, allowing us to work directly with memory addresses. This approach helps demystify pointers and provides more control over memory management.

Overall, pointers in assembly language are crucial for dynamic memory manipulation and efficient data access. They allow you to interact with memory locations directly, which is essential for low-level programming tasks

Summary:

The notes you provided are a bit unclear and disorganized, but they can be summarized as follows:

- A **pointer** is a variable that contains the address of another variable.
- The **PTR keyword** tells the assembler that the variable myPointer is a pointer.

• Pointers are useful for manipulating arrays and data structures because the address they hold can be modified at runtime.
• In 32-bit mode, pointers are stored in doubleword variables.
• You can use the OFFSET operator to make the relationship between a pointer and the variable it points to clearer.

Pointers are useful for a variety of reasons, including:

• They allow you to manipulate arrays and data structures in a more efficient way.
• They allow you to dynamically allocate memory at runtime.
• They allow you to pass arguments to functions by reference.
• They allow you to implement function pointers.

```
; Declare a pointer to a byte variable.
myPointer BYTE PTR

; Store the address of the byte variable `myByte` in the pointer `myPointer`.
MOV myPointer, OFFSET myByte

; Load the value of the byte variable pointed to by `myPointer` into the register `EAX`.
MOV EAX, [myPointer]

; Increment the value of the byte variable pointed to by `myPointer`.
INC [myPointer]
```

The first line declares a pointer to a byte variable. The BYTE PTR keyword tells the assembler that the variable myPointer is a pointer to a byte variable.

The second line stores the address of the byte variable myByte in the pointer myPointer. The OFFSET operator returns the address of the variable myByte.

The third line loads the value of the byte variable pointed to by myPointer into the register EAX. To do this, the assembler dereferences the pointer myPointer using the [] operator.

The fourth line increments the value of the byte variable pointed to by myPointer. To do this, the assembler dereferences the pointer myPointer using the [] operator and then increments the value of the byte variable at that address.