Direct Memory Operands

Direct memory operands are used to reference the contents of memory at a specific address. The address of the memory operand is specified explicitly in the instruction.

For example, the following instruction copies the value of the variable var1 into the AL register:

```
mov al, var1
```

The variable var1 is declared in the data segment as follows:

```
.data
var1 BYTE 10h
```

This means that var1 is located at offset 10400h in memory. The mov instruction with a direct memory operand is assembled into the following machine instruction:

```
A0 00010400
```

The first byte in the machine instruction is the operation code (opcode). The remaining part is the 32-bit hexadecimal address of var1.

Direct memory operands can be used in any instruction that allows a memory operand. For example, the following instruction adds the value of the variable var1 to the EAX register:

```
add eax, var1
```

The following instruction stores the value of the EAX register in the

variable var1:

```
mov var1, eax
```

Direct memory operands can also be used in arithmetic expressions. For example, the following instruction adds the value of the variable var1 to the value of the variable var2 and stores the result in the variable var3:

```
mov var3, var1 + var2
```

Alternative Notation
Some programmers prefer to use the following notation with direct operands:

```
mov al, [var1]
```

This notation is equivalent to the previous notation, but the brackets imply a dereference operation. The dereference operation means that the contents of the memory location at the address specified by var1 are loaded into the AL register.

MASM permits this notation, so you can use it in your own programs if you want. However, because so many programs (including those from Microsoft) are printed without the brackets, we will only use them in this book when an arithmetic expression is involved:

```
mov al, [var1 + 5]
```

This is called a direct-offset operand, and it is discussed in more detail later.

---

var1 BYTE 10h what does that mean, it usually confuses me, coz i say it like var1 is a varialbe that holds a byte which is 10h in decimal will be a value like say 16. can var1 be confused with something like var1 is a byte that holds a string inside 10h? or 10h is a hex, and strings can't be in hex?

The declaration var1 BYTE 10h means that var1 is a variable of type BYTE that is initialized to the value 10h. BYTE is a data type that can store a single byte of data. 10h is a hexadecimal value that is equal to 16 in decimal.

Strings cannot be stored in hexadecimal. Strings are typically stored in ASCII or Unicode encoding. ASCII and Unicode are character encodings that assign a unique number to each character in the alphabet. For example, the ASCII code for the letter "A" is 65.
So, to answer your questions:

- Yes, var1 is a variable that holds a byte which is 10h in decimal, which is a value of 16.
- No, var1 cannot be confused with something like a byte that holds a string inside 10h. Strings cannot be stored in hexadecimal.
- Yes, 10h is a hexadecimal value.

So when i see any hex value like var2 byte 10000438h what will that mean? it can mean var2 is holding a decimal or a memory location? or is byte small? Let's say var2 dword 10248132048h?

When you see a hexadecimal value in assembly language, such as 10000438h, it can mean either a decimal value or a memory location. It depends on the context in which the hexadecimal value is used.

For example, the following instruction copies the hexadecimal value 10000438h into the AL register:

```
mov al, 10000438h
```

In this case, the hexadecimal value is being used as a decimal value. The following instruction copies the contents of the memory location at address 10000438h into the EAX register:

```
mov eax, [10000438h]
```

In this case, the hexadecimal value is being used as a memory location.

To determine whether a hexadecimal value is being used as a decimal value or a memory location, you need to look at the context in which the hexadecimal value is used.

If the hexadecimal value is being used as an operand in an arithmetic instruction, then it is most likely being used as a decimal value.

If the hexadecimal value is being used as an operand in a memory instruction, then it is most likely being used as a memory location.

Byte is a data type that can store a single byte of data. Dword is a data type that can store a double word, which is four bytes of data.

The variable declaration var2 dword 10248132048h means that var2 is a variable of type DWORD that is initialized to the hexadecimal value 10248132048h.

This value is equal to 10248132048 in decimal.

So, to answer your questions:

- Yes, a hexadecimal value can mean that a variable is holding a decimal value or a memory location. It depends on the context in which the hexadecimal value is used.
- Yes, a hexadecimal value can be used to represent a memory location.

• The data type BYTE is smaller than the data type DWORD. A BYTE can store a single byte of data, while a DWORD can store four bytes of data.

So, if a hex is inside [] dereferencing its a memory location? Or it can also be just 28323428932h without brackets, but it's still a memory location?

Yes, if a hexadecimal value is inside square brackets ([]), it is a dereferenced memory location.

This means that the contents of the memory location at the address specified by the hexadecimal value are loaded into the register or memory operand that the dereference operator is applied to.

For example, the following instruction loads the contents of the memory location at address 28323428932h into the EAX register:

```
mov eax, [28323428932h]
```

However, if a hexadecimal value is not inside square brackets, it can be either a decimal value or a memory location, depending on the context in which it is used. For example, the following instruction copies the hexadecimal value 28323428932h into the EAX register:

```
mov eax, 28323428932h
```

In this case, the hexadecimal value is being used as a decimal value.

To determine whether a hexadecimal value is being used as a decimal value or a memory location, you need to look at the context in which the hexadecimal value is used.

If the hexadecimal value is being used as an operand in an arithmetic instruction, then it is most likely being used as a decimal value.

If the hexadecimal value is being used as an operand in a memory instruction, then it is most likely being used as a memory location.

Note: It is generally considered **good practice to use square brackets** around hexadecimal values that are being used as memory locations, even if they are not strictly necessary. This helps to make the code more readable and maintainable.

Some programmers prefer to use square brackets [ ] to indicate memory operands, as in mov al,[var1]. This notation explicitly shows that the instruction involves memory access. It's useful when you need to work with memory addresses directly or when performing more complex operations that include offsets, as in mov al,[var1 + 5].

- **Opcode:** It is the operation code that specifies the operation or instruction to be executed by the CPU. It tells the CPU what to do.

- **Operand:** It is the data or value on which the operation specified by the opcode is performed. It tells the CPU what to do the operation on.

```
;This example uses the MOV and ADD opcodes to add two numbers

mov eax, 5        ; Load 5 into the EAX register
add eax, 3        ; Add 3 to the value in EAX

;After execution, EAX will contain 8 (5 + 3)
```

```
;This example uses operands to perform addition

mov ebx, 7        ; Load 7 into the EBX register
mov ecx, 9        ; Load 9 into the ECX register
add ebx, ecx      ; Add the value in ECX to EBX

; After execution, EBX will contain 16 (7 + 9)
```

In this example, ebx and ecx are operands. add is the opcode(operation code), and the operands(data being worked on) are the values in ebx and ecx that are being added together.