Overlapping Values

The provided code example demonstrates how differently sized data can overlap and affect the values stored in a 32-bit register (EAX) in x86 assembly language. Let's break down the code and understand how each instruction affects the register's contents:

```
.data
    oneByte BYTE 78h
    oneWord WORD 1234h
    oneDword DWORD 12345678h
```

In the data section, three variables are declared: oneByte as a single byte, oneWord as a 16-bit word, and oneDword as a 32-bit double word.

Now, let's go through the code:

```
mov eax, 0
```

Sets the entire 32-bit EAX register to zero.

Result: EAX = 00000000h

****

```
mov al, oneByte
```

Moves the value of oneByte (78h) into the low byte of EAX (AL), effectively overwriting the lowest 8 bits of EAX.

Result: EAX = 00000078h

****

```
mov ax, oneWord
```

Moves the value of oneWord (1234h) into the lower 16 bits of EAX (AX), effectively overwriting the lower 16 bits of EAX.

Result: EAX = 00001234h

****

```
mov eax, oneDword
```

Moves the value of oneDword (12345678h) into the entire EAX register, replacing its previous value.

Result: EAX = 12345678h

****

```
mov ax, 0
```

Sets the lower 16 bits of EAX (AX) to zero, leaving the upper 16 bits unchanged.

Result: EAX = 12340000h

****

The key takeaway from this code is that when you move data of different sizes into a larger register, the smaller data gets placed in the lower portion of the larger register while leaving the higher bits unaffected.

In this case, AL represents the lowest byte, AX represents the lowest two bytes, and EAX represents the entire 32-bit register. It's important to be mindful of data size when performing such operations to avoid unintended side effects.