Data Declaration and Direct Addressing in Assembly Language

1. Understanding Data Declaration in Assembly

In assembly language programming, data declaration refers to the process of reserving memory space for variables, which can store data for use during program execution. Instead of constants, memory variables allow programs to store and modify data, making them more flexible and efficient.

For example, the instruction "mov ax, 5" places the constant value 5 in the AX register. However, using constants repeatedly is impractical. To solve this, we declare variables using assembler directives, such as "db" (define byte) and "dw" (define word). A byte is 8 bits, while a word is 16 bits.

Example:

```
db somevalue ; Reserves 1 byte
dw somevalue ; Reserves 2 bytes
```

By associating memory locations with symbols (labels), we can easily refer to variables by their labels instead of addresses. This makes code easier to understand.

Example:

```
num1: dw 5
num2: dw 10
```

In this example, num1 and num2 are memory variables storing 5 and 10, respectively. The label num1 points to the address where the value 5 is stored.

The assembler calculates the addresses for these labels during compilation.

2. Direct Addressing Mode

In direct addressing mode, a memory address is specified explicitly in the instruction. The data at that address is used in the operation.

For instance:

```
mov ax, [num1] ; Load the value at address num1 into register AX
```

In this example, the value 5 (stored at num1) is loaded into the AX register.

Direct addressing allows us to use memory variables instead of constants, making the program more versatile. In addition, data stored in memory can serve as both the source and destination operands in instructions, which isn't possible with constants.

Example Program (Direct Addressing):

```
mov ax, [num1] ; Load first number
mov bx, [num2] ; Load second number
add ax, bx ; Add the two numbers
mov [num3], ax ; Store the result in num3
```

This simple program demonstrates how to add two numbers stored in memory and store the result in another memory location.

Important Notes:

1. The "mov" instruction can only move data between registers and memory, or between registers and constants. Memory-to-memory transfers are not allowed directly.

2. Using direct addressing ensures that the exact memory location of a variable is used for

operations.

In summary, direct addressing makes programs more efficient by allowing memory variables to be manipulated, providing greater flexibility than using constant values.