Operand Types

Let's clear this up first, before we move on:

The data transfer instructions are used to move data between registers, memory, and the stack. The basic addressing modes are direct, immediate, and indirect. Direct addressing is used to access data at a specific memory address. Immediate addressing is used to load a constant value into a register. Indirect addressing is used to access data at a memory address that is stored in a register.

The chapter also introduces the OFFSET, PTR, and LENGTHOF operators. The OFFSET operator is used to calculate the offset of a variable or structure member relative to the beginning of a data structure. The PTR operator is used to cast a value to a pointer type. The LENGTHOF operator is used to calculate the length of a data structure in bytes.

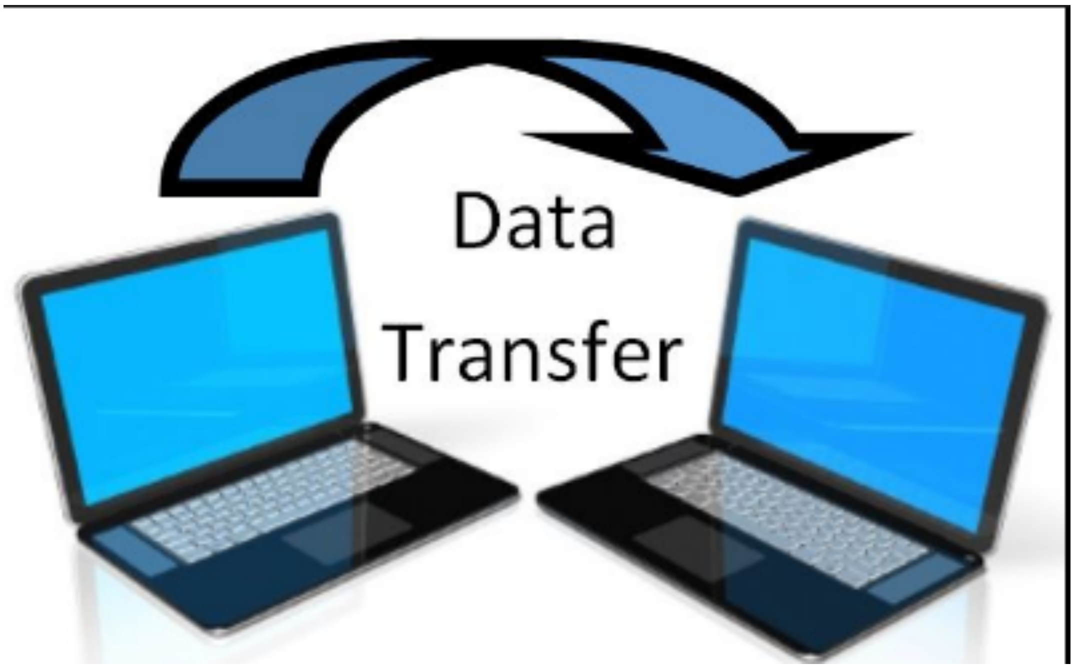
The chapter ends by discussing how to create loops and use some of the basic arithmetic operators.

Here is a summary of the key points in this chapter:

Assembly language is a low-level programming language that gives programmers direct control over the processor.

Compilers perform strict type checking, but assemblers do not. x86 processors have a complex instruction set, which means that there are many different ways to do things.

The basic **data transfer instructions** are MOV, PUSH, and POP.

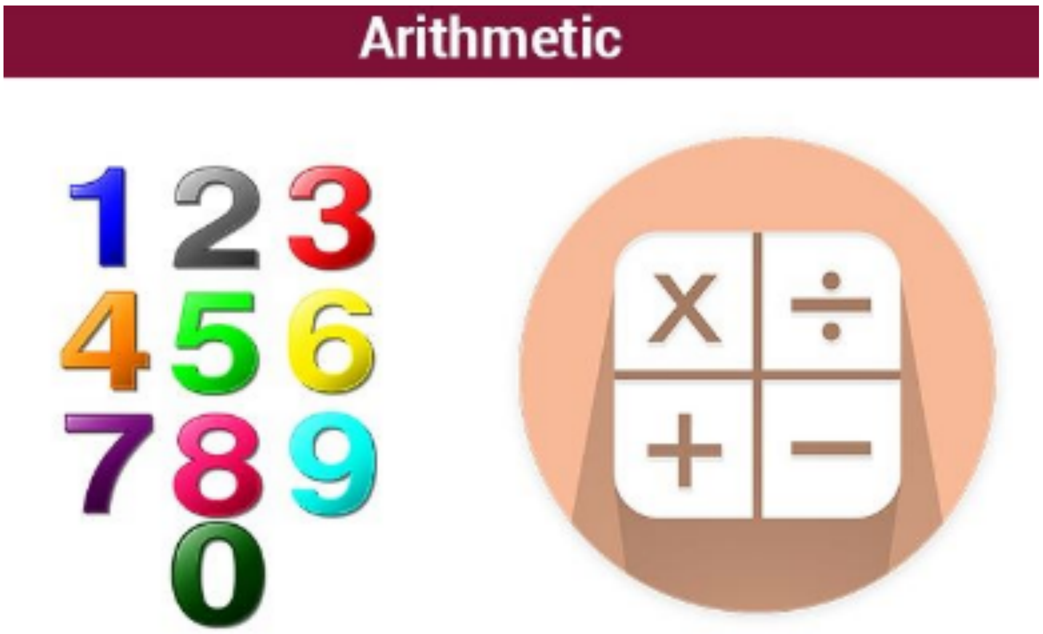


The **basic addressing modes** are direct, immediate, and indirect.

The OFFSET, PTR, and LENGTHOF operators can be used to **manipulate data** in memory.

Loops can be created using the JMP and LOOP instructions.

The **basic arithmetic operators** are ADD, SUB, MUL, and DIV.



Once you have mastered the material in this chapter, you will have a solid foundation in assembly language programming.

Three basic types of operands in x86 assembly language:

• Immediate operands are numeric literal expressions, such as 10 or -255.
• Register operands are named registers in the CPU, such as EAX or EDX.
• Memory operands reference memory locations.

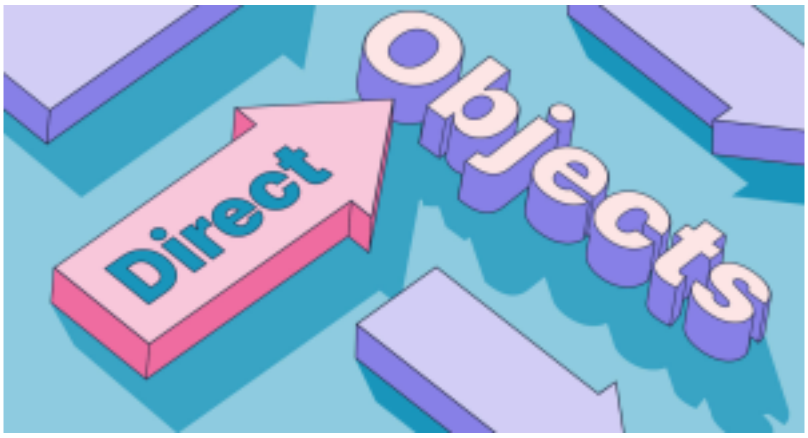
The syntax of x86 instructions depends on the type of operands they use. For example, the MOV instruction can be used to move data between two registers, or to move data between a register and a memory location. The following table shows the syntax of the MOV instruction for different operand types:

```
Register to register      | MOV destination, source
Immediate to register     | mov destination, immediate
Register to memory        | mov [memory location], source
Immediate to memory       | mov [memory location], immediate
```

Memory operands can be specified in a variety of ways, depending on the addressing mode being used.

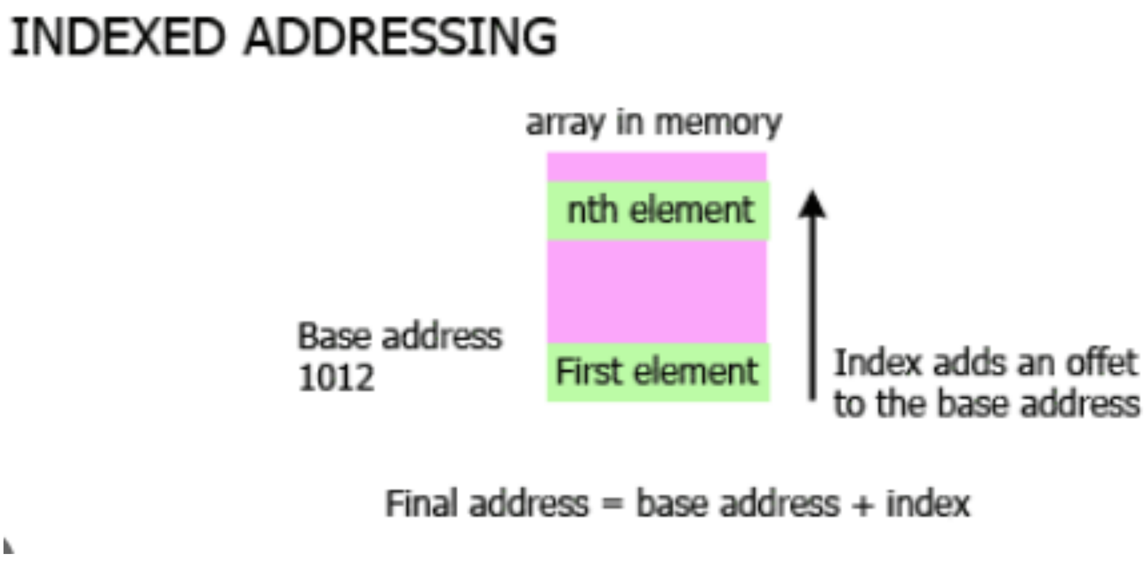
The three most common addressing modes are:

• **Direct addressing** specifies the memory address directly.



• Indirect addressing specifies the memory address indirectly, through a register.

• Indexed addressing specifies the memory address using a register as an index.



The following table shows some examples of memory operands using different addressing modes:

```
Direct addressing   [memory address]
Indirect addressing [register]
Indexed addressing  [register + offset]
```

The **OFFSET operator** can be used to calculate the offset of a variable or structure member relative to the beginning of a data structure.

The **PTR operator** can be used to cast a value to a pointer type.

The **LENGTHOF operator** can be used to calculate the length of a data structure in bytes.

By understanding the different types of operands and addressing modes, you can write x86 assembly language code to perform a wide range of tasks.

Yes, when writing programs for Windows NT (and its successors), you often target a 32-bit flat address space and use a 32-bit instruction set. The use of .MODEL FLAT in assembly language programming and the STDCALL calling convention is related to this.

.MODEL FLAT:

- The .MODEL FLAT directive is used in assembly language programming to specify that you are working in a flat memory model. In a flat memory model, you have a single, continuous address space that can be accessed using 32-bit addressing. This is in contrast to segmented memory models, where memory is divided into segments, and you have to deal with segment registers and offsets.

STDCALL:

- STDCALL is a calling convention used in Windows API functions and many other Windows-related programming contexts.
- It specifies the calling convention for functions, including how parameters are passed to functions and how function results are returned.
- In the STDCALL calling convention, function parameters are typically pushed onto the stack from right to left, and the callee (the called function) is responsible for cleaning up the stack after the function call.
- The STDCALL calling convention ensures a consistent and predictable way for functions to interact, which is crucial for compatibility and interoperability in Windows programming.

So, when you see .MODEL FLAT and STDCALL in assembly code for Windows programming, it's because they are defining the memory model and calling convention that are common and expected in the Windows operating system's environment. This allows for consistent and efficient interaction between your code and the Windows API.

If you're writing 32-bit Windows programs using assembly language or working with WinAPI functions in C programming, you will encounter these conventions frequently to ensure your code can interface with the Windows operating system correctly.