**Construction of 8086 Machine Code**

Table of Contents

[Instruction Templates 3](#_Toc83226944)

[IN Instruction Template 3](#_Toc83226945)

[MOV Instruction Template 4](#_Toc83226946)

When we pass **assembly language** through an **assembler**, we get a series of binary bits as an output, called the **machine code**. To make presentation easier for us, we will be writing any machine code in **hexadecimal** format, but in reality, it is stored as binary bits.

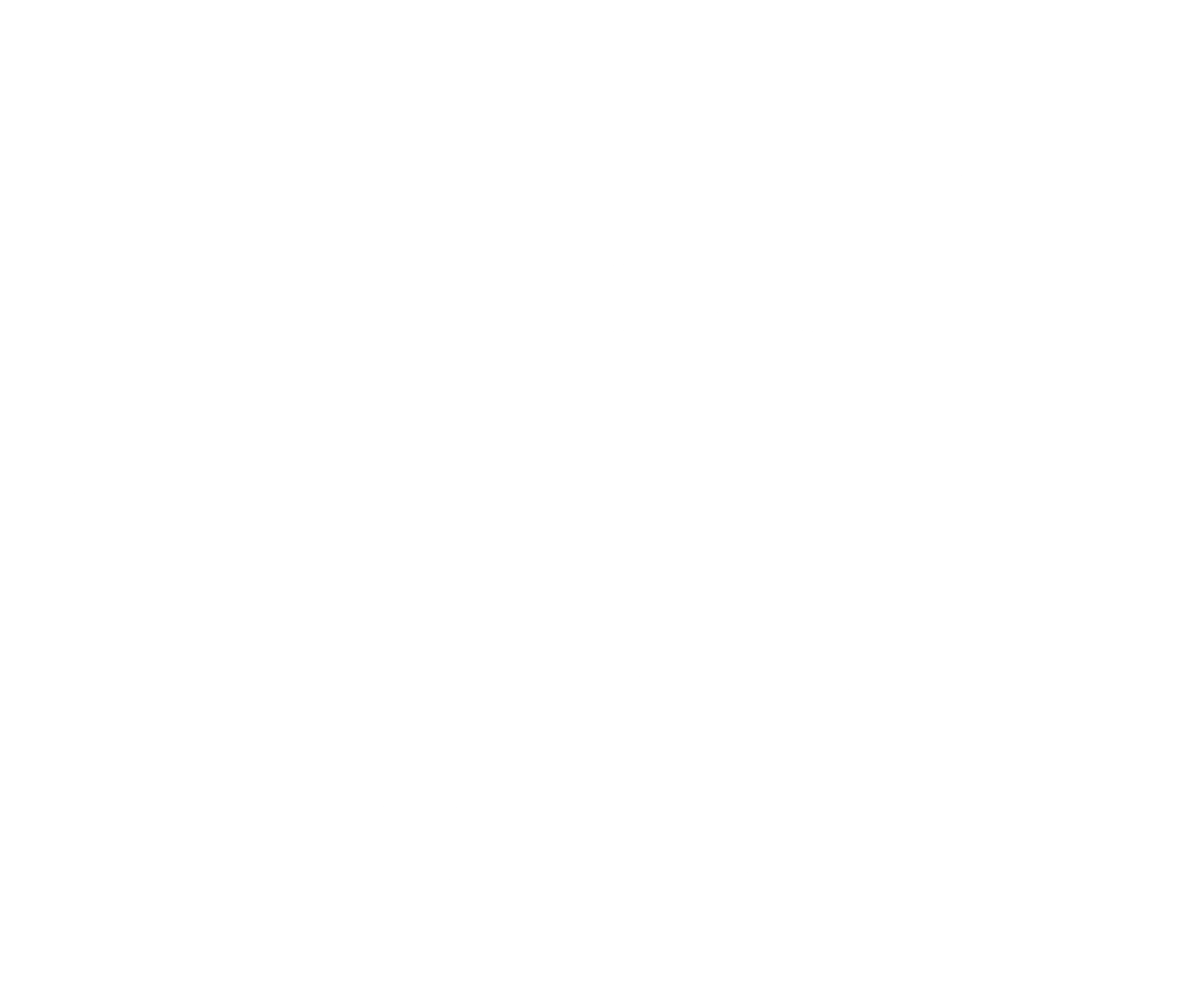
The Intel 8086 has 117 instructions in its instruction set. Each of these instructions is associated with a **binary code** (op-code). Most of the time, it is the **assembler’s** job to convert the instructions into the binary code. The assembler uses some logic, instruction templates and coding formats to do this. For particular instructions, it also needs address modes and register/memory bit patterns. We will be looking into all of this.

## Instruction Templates

### IN Instruction Template

IN AL, 05h

ASSEMBLY



There are **two bytes** in the template. The most significant byte, byte 1, will be stored in the **lower** memory location and the least significant byte, byte 2, will be stored in the **higher** memory location.

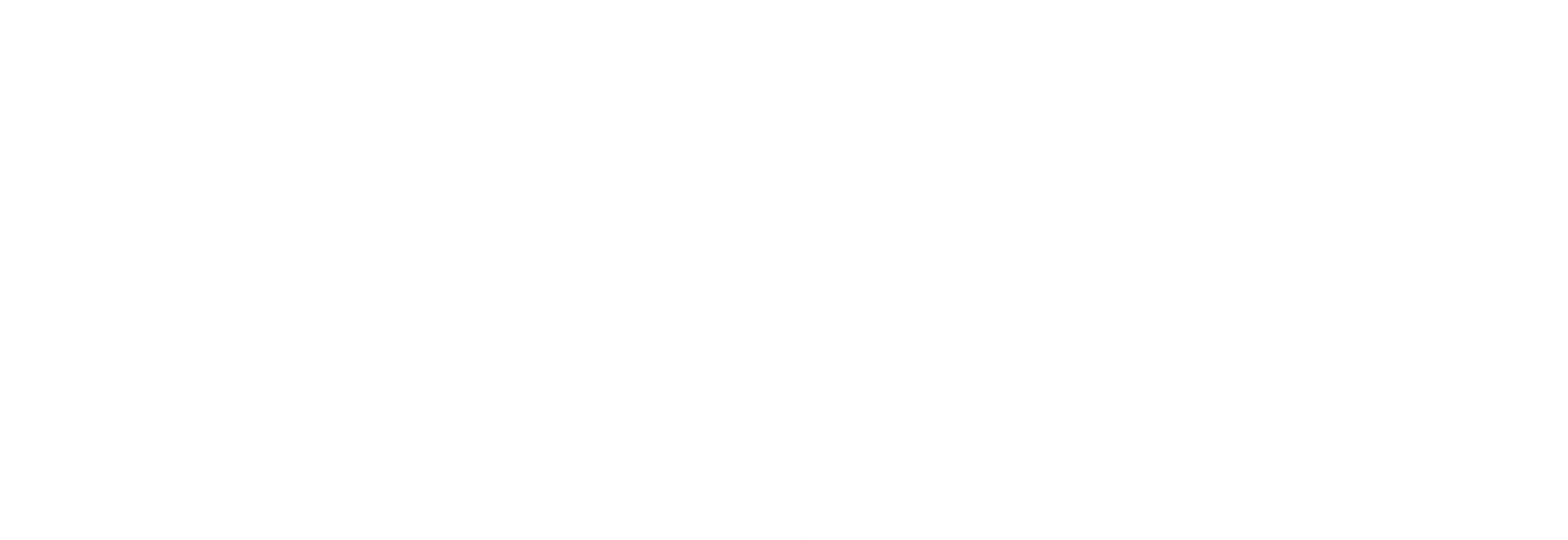
The **binary op-code** for the IN instruction is stored in the **first 7 bits**. The 8th bit stores the **W** value. This is if the IN instruction is meant to take in **one byte** of data. It is if the IN instruction is meant to take **one word** or bits of input. For the instruction above, it is set to , since the input goes to AL. If the input went to AX, then this would be set to . These bits make up the **first byte**.

The **second byte** contains the **port address**.

Thus, the machine code is 11100100 00000101. The two bytes are stored in the addresses 00205h and 00206h respectively.

### MOV Instruction Template

The template for the **MOV** instruction has **four bytes**. The first two bytes are **mandatory**, while the last two are **optional**.



The first **six bits** have the **opcode**, 100010. Next, we have the **direction** bit, D. This is set to 0 if we are moving data **from** a register and set to 1 if we are moving data **to** a register. Finally, we have the **W** bit, which works in the same way that it did in the IN instruction. These 8 bits make up the **first byte**.

Next, we have **two bits** to identify the **addressing mode**. The seven addressing modes we saw can be divided into **four categories**. We will be looking into those soon. Then **three bits** are used to identify one of the eight **registers** to or from which we are moving data. The next **three bits** are for the **R/M value**. These 8 bits make up the **second byte**.

**Bytes three and four** are used if we have **displacement**.

#### REG Field Values

The REG field has a different value depending on which **register** we want to use as well as whether we are performing a **byte** or a **word** operation.

|  |  |  |
| --- | --- | --- |
| REG | W | W |
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

#### MOD and R/M Fields

Using the MOV instruction, we can transfer data:

1. From **register-to-register**
2. From **register-to-memory**
3. From **memory-to-register**

We already provide the address for one **register** in the instruction itself and specify whether we are moving data **from** it or **to** it. The other source or destination is specified using the MOD and R/M fields.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RM \ MOD | 00 | 01 | 10 | 11 | |
| W | W |
| 000 | [BX] + [SI] | [BX] + [SI] + d8 | [BX] + [SI] + d16 | AL | AX |
| 001 | [BX] + [DI] | [BX] + [DI] + d8 | [BX] + [DI] + d16 | CL | CX |
| 010 | [BP] + [SI] | [BP] + [SI] + d8 | [BP] + [SI] + d16 | DL | DX |
| 011 | [BP] + [DI] | [BP] + [DI] + d8 | [BP] + [DI] + d16 | BL | BX |
| 100 | [SI] | [SI] + d8 | [SI] + d16 | AH | SP |
| 101 | [DI] | [DI] + d8 | [DI] + d16 | CH | BP |
| 110 | d16 (direct address) | [BP] + d8 | [BP] + d16 | DH | SI |
| 111 | [BX] | [BX] + d8 | [BX] + d16 | BH | DI |

For **register-to-register**, we always use a MOD value of 11. Notice that the pattern there is the same as that for the REG field.

A MOD value of 00 is for **base plus index** values for memory addresses.

A MOD value of 01 is for **base plus index plus relative** values for memory addresses. Here, the relative addresses are **8 bits**.

A MOD value of 10 is for **base plus index plus relative** values for memory addresses as well. Here, the relative addresses are **16 bits**.

Example 1

Consider the following instruction:

MOV BL, AL

ASSEMBLY

We will be building up the machine code for this step by step, following the template.

First, we have the **opcode**, 100010.

We are moving data **from** the AL register. We will be providing the address for AL in the REG section. Thus, D is set to 0. The opposite could have been done as well and we would need to adjust the value of the REG section accordingly. However, note that an assembler would not take a **memory location** or an **indirect address** as a reference point. If there is one register, it will take that one. If there are two, as there is here, either can be taken.

The AL and BL registers are both **8-bit** registers. Thus, W is set to 0.

The REG value is set to 000, the code for **AL**.

The MOD value is set to 11 and the R/M value is set to 011. This combination is used to identify **BL**.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 100010 | 0 | 0 | 11 | 000 | 011 |
| Op-code | D | W | MOD | REG | R/M |

Example 2

MOV CL, [BX]

ASSEMBLY

Again, the **op code** is 100010.

As mentioned before, the assembler will not consider an indirect address as the reference point. Thus, we are moving data **to** CL, meaning D is set to 1.

We are moving **1 byte**, so W is set to 0.

The REG value is 001 for CL.

The MOD value is set to 00, since we are moving data from a **memory location** with **no displacement**. The R/M value for just [BX] is 111.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 100010 | 1 | 0 | 00 | 001 | 111 |
| Op-code | D | W | MOD | REG | R/M |

Example 3

MOV 43h [SI], DH

ASSEMBLY

The instruction above can also be written as

MOV [SI] + 43h, DH

ASSEMBLY

This should be easier to understand.

Sine we only have one **direct register** reference, we need to move data **from** DH. The D value will thus be 0. The REG value will be 110 and the W value will be 0 as well.

The destination address is stored in SI and we have an **8-bit displacement**. Thus, the MOD value is 01 and the R/M value is 100.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 100010 | 0 | 0 | 01 | 110 | 100 |
| Op-code | D | W | MOD | REG | R/M |

We have mentioned that we have a **displacement**, but we did not say what the displacement is yet. Thus, we store the displacement in the **third byte**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100010 | 0 | 0 | 01 | 110 | 100 | 01000011 |
| Op-code | D | W | MOD | REG | R/M | Displacement |

Example 4

MOV CX, [437Ah]

ASSEMBLY

We will be moving data **to** CX, so D is set to 1, W is set to 1 and REG is set to 001.

The other address is a **direct address**, so MOD is set to 00 and R/M is set to 110.

The direct address is of 16 bits. The **lower 8 bits**, 7Ah, are stored in the **third byte**. The **higher 8 bits**, 43h, are stored in the **fourth byte**. This is **Little Endian**. This order is important, since the data is being stored in this order in consecutive memory locations.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 100010 | 1 | 1 | 00 | 001 | 110 | 01111010 | 01000011 |
| Op-code | D | W | MOD | REG | R/M | Direct Address Lower Byte | Direct Address Higher Byte |

Of course, the provided address is an **offset**. The **segment number** needs to be retrieved from the DS register to obtain the complete address.

Example 5

MOV AX, 0010h

ASSEMBLY

In the above code, we are moving a **direct value** into the AX register.

When we are moving an **immediate value**, the template is different:

We now have a **prefix**, which is 1011. After this, we have the W value, which is 1 in this case, since AX is a 16-bit register. Next, we have the REG value, 000. This makes up the **first byte**.

The **second** and **third bytes** consists of the actual value, 0000 0000 and 0001 0000 in binary. The low part comes first and the high part comes second. If the value was 8 bits, i.e. W is 0, then we would not need the third byte.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1011 | 1 | 000 | 0001 0000 | 0000 0000 |
| Prefix | W | REG | Data | Data |

Example 6

We have previous seen code to copy some data from a register to an indirect address.

MOV [BX], DL

ASSEMBLY

We know that the address stored in the BX register is an **offset** number. We also need the **segment** number, which we can obtain from the **DS** register.

However, it is also possible to use a different segment register, a process called **overriding**. For example, we can choose to use the CS register, which indicates that we are moving the data into the code segment instead of the data segment. Since this is not the default behaviour, we need to specify it explicitly.

MOV CS:[BX], DL

ASSEMBLY

For the regular operation (without the override), we would have the following machine code:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 100010 | 0 | 0 | 00 | 010 | 111 |
| Op-code | D | W | MOD | REG | R/M |

We are moving data from DL, so D is 0. Since the source (DL) is 8-bits, W is 0. Since we are using [BX], the MOD value is 00 and the R/M value is 111. Since the reference register is DL, the REG value is 010.

However, since we are performing an override, there will be an additional prefix, **before** the op-code. This prefix has the format 001xx110. Here, the xx value will depend on the **overriding register**.

|  |  |
| --- | --- |
| Register | Value |
| ES | 00 |
| CS | 01 |
| SS | 10 |
| DS | 11 |

Thus, for the example shown above, the final code will be:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 001**01**110 | 100010 | 0 | 0 | 00 | 010 | 111 |
| Prefix | Op-code | D | W | MOD | REG | R/M |

Thus, the prefix takes the first byte, the op-code, D value and W value take the second byte and the MOD value, the REG value and the R/M value take up the third byte.