**Conditional Control Instructions**

Table of Contents

[Compare 2](#_Toc83228455)

[Labels and Jumps 3](#_Toc83228456)

[Labels 3](#_Toc83228457)

[Jumps 3](#_Toc83228458)

[Unconditional Jumps 5](#_Toc83228459)

[Control Flow 5](#_Toc83228460)

## Compare

The CMP instruction is very useful when used alongside conditional control instructions. The CMP instruction is very similar to the SUB instruction, so let’s take a look at that again.

The SUB instruction takes two parameters, the **destination** and the **source**.

SUB AL, 2

ASSEMBLY

This instruction **subtracts** from AL and stores the result **back into AL**, the destination.

The CMP instruction does exactly the same thing, except it **does not store** the results back into the destination.

CMP AL, BL

ASSEMBLY

If we run the code above, there are three possible results. Either the result is **positive**, meaning AL **is greater than** BL, or the result is **negative**, meaning AL **is less than** BL or the result is **zero**, meaning AL **is equal to** BL.

## Labels and Jumps

If we are not storing the results of a comparison using the CMP instruction, then what are we doing with it? This is where the **jump instructions** come in. However, before we can talk about that, we need to talk about **labels**.

### Labels

A **label** is just an identifier for a ‘section’ of code. It does not do anything else. It is simply a bookmark. We can use labels alongside jump instructions to specify which point of a code to jump to.

ORG 0100H  
MAIN PROC  
 *; some code*some\_label:  
 *; some more code* MAIN ENDP  
END MAIN  
RET

ASSEMBLY

### Jumps

Back to **jumps**. We can place a **jump instruction** directly after a CMP instruction. Depending on the results of the CMP instruction, the jump instruction may or may not be executed. The jump instruction itself will take us to a **location** in our program specified by a **label**.

For example, we can have a jump instruction that will execute if the CMP instruction gives a **positive** result, i.e. the first number is **greater than** the second number.

CMP AL, BL  
JG some\_label *; if AL > Bl, jump to some\_label*

ASSEMBLY

Thus, if AL is greater than BL, only then will the jump instruction, JG, execute.

Similarly, there are several other jump instructions.

CMP AL, BL  
JL some\_label *; executes if AL < BL*

CMP AL, BL  
JB some\_label *; executes if AL < BL, same as JL*

SUB AL, BL  
JZ some\_label *; executes if the Zero flag is set*

CMP AL, BL  
JNZ some\_label *; executes if AL != BL*

CMP AL, BL  
JGE some\_label *; executes if AL >= BL*

CMP AL, BL  
JLE some\_label *; executes if AL <= BL*

ADD AL, BL  
JC some\_label *; jump if the Carry flag is set*

ASSEMBLY

Notice that we had to use a CMP instruction before most of the jump instructions. The ones where we did not depended on specific flag values, as mentioned in the comments.

### Unconditional Jumps

It is also possible to have a jump instruction that executes on its own, without depending on a CMP instruction.

JMP some\_label

ASSEMBLY

### Control Flow

Jump instructions are not anything like functions from high-level languages. We are not calling anything. We are **physically jumping** to the location of the specified label.

This means that the program execution continues **downwards** from the point where we jumped to.

some\_label:  
 *; some code*some\_other\_label:  
 *; some more code* JMP some\_label  
  
*; code that ends the program*

ASSEMBLY

The above code will cause the program to enter an infinite loop. We execute the code under some\_label first, then the code under some\_other\_label. However, the jump instruction takes us back to some\_label, and we start executing that section yet again. We never get to the code that is at the end of the program.

The fact that Assembly language code does not behave like high-level languages and only executes downwards also means that the **first** section of code in our program has to be the ‘main’ section (even though Assembly language does not technically have a specified main section). Basically, we cannot put a bunch of sections of code at the top of our program under different labels as though they were functions, because obviously, they aren’t. The first lines of code have to be direct code. It can’t even be a single jump instruction that takes us to a labelled section.

ORG 0100H  
MAIN PROC  
*; this code won't work*main:  
 *; the 'main' code* MAIN ENDP  
END MAIN  
RET

ASSEMBLY

ORG 0100H  
MAIN PROC  
*; this won't work either* JMP main  
main:  
 *; the 'main' code* MAIN ENDP  
END MAIN  
RET

ASSEMBLY

ORG 0100H  
MAIN PROC  
*; this code will work  
 ; the main code*some\_label:  
 *; some code* MAIN ENDP  
END MAIN  
RET

ASSEMBLY