**PMAS Arid Agriculture University Rawalpindi**

**University Institute of Information Technology**

**LAB MANUAL - VI**

**Class/Program: BS (CS)** **Course: COAL (CS530)**



**Objectives:**

1. **How to check Conditions and perform comparison**
2. **Introduction to CMP statement**
3. **Introduction to jumps**
4. **Use of jumps with CMP statements**
5. **Different types of jumps**
6. **Introduction to Code labels**
7. **Use of Code labels**
8. **Difference between data label and code label**

**CMP statement:**

**To check simple or logical conditions, or to perform comparisons in assembly, CMP command is used. CMP command uses two operands and perform an implied subtraction between the two operands. CMP in conjunction with conditional jumps commands can perform the IF STATEMENT of C/C++. CMP instruction sets different flags depending upon the values of the destination operand. While the conditional jump interrupts the sequential execution of the instructions and branches the program control to given destination. The general syntax is given below.**

**CMP (space) destination, source**

**Conditional Jumps (space) destination**

**The destination and source with the CMP command can be any two registers, one register and one memory or vice versa, one register or one immediate value and one memory and one immediately value. No other combination is possible. While the destination with the conditional jump must be a code label name. Different conditional jumps are given below.**

**Program flow control**

Controlling the program flow is a very important thing, this is where your program can make decisions according to certain conditions.

* **unconditional jumps**  
    
  The basic instruction that transfers control to another point in the program is **JMP**.  
    
  The basic syntax of **JMP** instruction:

JMP label

To declare a *label* in your program, just type its name and add "**:**" to the end, label can be any character combination but it cannot start with a number, for example here are 3 legal label definitions:

label1:  
label2:  
a:

Label can be declared on a separate line or before any other instruction, for example:

x1:  
MOV AX, 1  
x2: MOV AX, 2

here's an example of **JMP** instruction:

org 100h

mov ax, 5 ; set ax to 5.

mov bx, 2 ; set bx to 2.

jmp calc ; go to 'calc'.

back: jmp stop ; go to 'stop'.

calc:

add ax, bx ; add bx to ax.

jmp back ; go 'back'.

stop:

ret ; return to operating system.

Of course there is an easier way to calculate the some of two numbers, but it's still a good example of **JMP** instruction.  
As you can see from this example **JMP** is able to transfer control both forward and backward. It can jump anywhere in current code segment (65,535 bytes).

* **Short Conditional Jumps**  
    
  Unlike **JMP** instruction that does an unconditional jump, there are instructions that do a conditional jumps (jump only when some conditions are in act). These instructions are divided in three groups, first group just test single flag, second compares numbers as signed, and third compares numbers as unsigned.  
    
  **Jump instructions that test single flag**

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Description | Condition | Opposite Instruction |
| JZ , JE | Jump if Zero (Equal). | ZF = 1 | JNZ, JNE |
| JC , JB, JNAE | Jump if Carry (Below, Not Above Equal). | CF = 1 | JNC, JNB, JAE |
| JS | Jump if Sign. | SF = 1 | JNS |
| JO | Jump if Overflow. | OF = 1 | JNO |
| JPE, JP | Jump if Parity Even. | PF = 1 | JPO |
| JNZ , JNE | Jump if Not Zero (Not Equal). | ZF = 0 | JZ, JE |
| JNC , JNB, JAE | Jump if Not Carry (Not Below, Above Equal). | CF = 0 | JC, JB, JNAE |
| JNS | Jump if Not Sign. | SF = 0 | JS |
| JNO | Jump if Not Overflow. | OF = 0 | JO |
| JPO, JNP | Jump if Parity Odd (No Parity). | PF = 0 | JPE, JP |

as you may already notice there are some instructions that do that same thing, that's correct, they even are assembled into the same machine code, so it's good to remember that when you compile **JE** instruction - you will get it disassembled as: **JZ**, **JC** is assembled the same as **JB** etc...  
different names are used to make programs easier to understand, to code and most importantly to remember. very offset dissembler has no clue what the original instruction was look like that's why it uses the most common name.  
  
if you emulate this code you will see that all instructions are assembled into **JNB**, the operational code (opcode) for this instruction is **73h** this instruction has fixed length of two bytes, the second byte is number of bytes to add to the **IP** register if the condition is true. because the instruction has only 1 byte to keep the offset it is limited to pass control to -128 bytes back or 127 bytes forward, this value is always signed.

jnc a

jnb a

jae a

mov ax, 4

a: mov ax, 5

ret

**Jump instructions for signed numbers**

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Description | Condition | Opposite Instruction |
| JE , JZ | Jump if Equal (=). Jump if Zero. | ZF = 1 | JNE, JNZ |
| JNE , JNZ | Jump if Not Equal (<>). Jump if Not Zero. | ZF = 0 | JE, JZ |
| JG , JNLE | Jump if Greater (>). Jump if Not Less or Equal (not <=). | ZF = 0 and SF = OF | JNG, JLE |
| JL , JNGE | Jump if Less (<). Jump if Not Greater or Equal (not >=). | SF <> OF | JNL, JGE |
| JGE , JNL | Jump if Greater or Equal (>=). Jump if Not Less (not <). | SF = OF | JNGE, JL |
| JLE , JNG | Jump if Less or Equal (<=). Jump if Not Greater (not >). | ZF = 1 or SF <> OF | JNLE, JG |

<> - sign means not equal.  
  
  
**Jump instructions for unsigned numbers**

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Description | Condition | Opposite Instruction |
| JE , JZ | Jump if Equal (=). Jump if Zero. | ZF = 1 | JNE, JNZ |
| JNE , JNZ | Jump if Not Equal (<>). Jump if Not Zero. | ZF = 0 | JE, JZ |
| JA , JNBE | Jump if Above (>). Jump if Not Below or Equal (not <=). | CF = 0 and ZF = 0 | JNA, JBE |
| JB , JNAE, JC | Jump if Below (<). Jump if Not Above or Equal (not >=). Jump if Carry. | CF = 1 | JNB, JAE, JNC |
| JAE , JNB, JNC | Jump if Above or Equal (>=). Jump if Not Below (not <). Jump if Not Carry. | CF = 0 | JNAE, JB |
| JBE , JNA | Jump if Below or Equal (<=). Jump if Not Above (not >). | CF = 1 or ZF = 1 | JNBE, JA |

Generally, when it is required to compare numeric values **CMP** instruction is used (it does the same as **SUB** (subtract) instruction, but does not keep the result, just affects the flags).  
  
The logic is very simple, for example:  
it's required to compare 5 and 2,  
5 - 2 = 3  
the result is not zero (Zero Flag is set to 0).  
  
Another example:  
it's required to compare 7 and 7,  
7 - 7 = 0  
the result is zero! (Zero Flag is set to 1 and **JZ** or **JE** will do the jump).

**Introduction to Labels:**

In assembly language labels are used to represent different values and areas of the program. There are two types of labels in assembly

Data Labels: the names of the variables are technically termed as data labels in assembly language. These are present in .DATA.

For example num BYTE 4, where num is a data label

Code Labels: the markers in the program to separate different segments or block of code. These are present in .CODE.

For example

num:

; General program statements