

CSE-231

MICRO PROCESSOR AND ASSEMBLY LANGUAGE LAB

DEPERTMENT of C.S.E

SECTION: A

ID: 161 – 15 -:3

DATE OF SUBMISSION: 12/12/2017

**PROJECT NAME: SIGNED JUMP INSTRUCTION**

SUBMITTED TO:

SUBMITTED BY:

MD. HAFIZUR RAHMAN ARFIN

LECTURER, DEPARTMENT OF C.S.E

MR. MD. RIAZUR RAHMAN

DAFFODIL INTERNATIONAL UNIVERSITY

DAFFODIL INTERNATIONAL UNIVERSITY

Introduction:

The programme control instructions allow the flow of the programme to change. A change in flow often occurs when decisions are made. Programme control flow is affected by two types of instructions, Unconditional and Conditional. These are also called branching instructions. JMP an unconditional jump will take the main program counter to the specified location. Conditional jumps are based on the current status of the different flags in the flag or status register.

Details:   
“Jump if signed” is a conditional jump instruction. It checks the “**Sign Flag(SF)**”. The FLAGS register is the status register that contains the current state of the processor. There are different kinds of flags like Carry Flag (CF), Parity Flag (PF), Zero flag (ZF), Overflow Flag (OF) and Signed Flag(SF).

**Signed Flag:** Changes when operation **produced a negative number**. In a computer processor the negative flag or sign flag is a single bit in a system status (flag) register used to indicate whether the **result of the last mathematical operation** resulted in a value whose **most significant bit was set**. In a **two's complement** interpretation of the result, the negative flag is **set** if the result was negative.

**Jump if Signed:** if the sign bit is set from a previous arithmetic expression. This instruction is executed and Loads EIP with the specified address

JS Location

Executes when SF = 1

|  |  |  |  |
| --- | --- | --- | --- |
| **JS** | Jump if sign |  | SF = 1 |

For example, in an 8-bit signed number system, -37 will be represented as 1101 1011 in binary (the most significant bit is 1), while +37 will be represented as 0010 0101 (the most significant bit is 0).  
As the MSB is 1, the **JS** instruction will executed in case of -37.

*In essence if the last operation produced a negative number the JS instruction executes.*

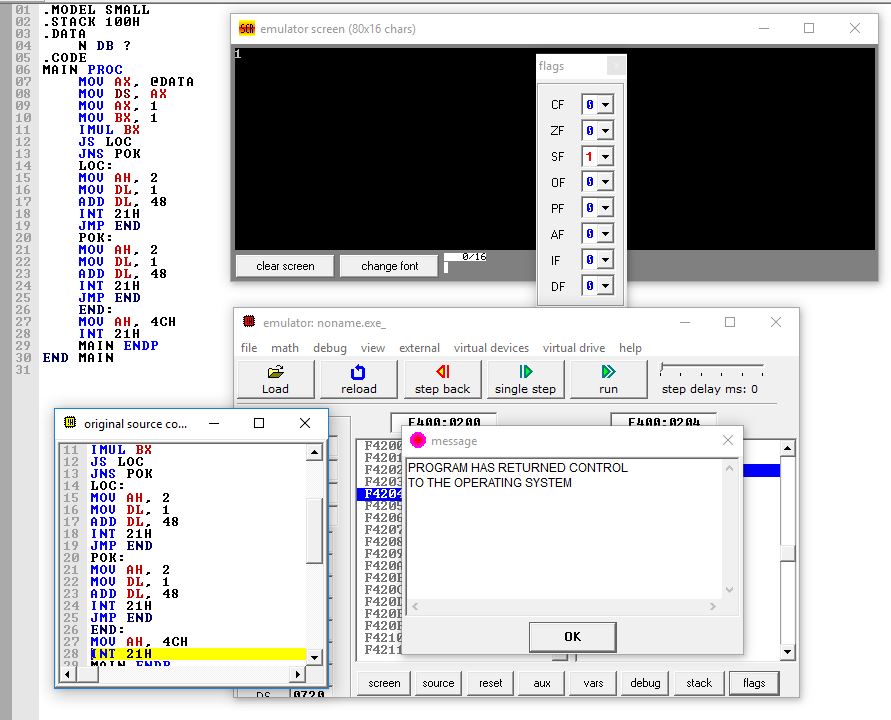
System:

1. A computer (Desktop, with intel core processor)
2. An 8086 emulator (EMU8086)

Live Example:  
.MODEL SMALL  
.STACK 100H  
.DATA  
 N DB ?  
.CODE  
MAIN PROC  
 MOV AX, @DATA  
 MOV DS, AX  
 MOV AX, 1  
 MOV BX, -1  
 IMUL BX

JS LOC  
 JNS POK

LOC:  
 MOV AH, 2  
 MOV DL, 1  
 ADD DL, 48  
 INT 21H  
 JMP END  
 POK:  
 MOV AH, 2  
 MOV DL, 0  
 ADD DL, 48  
 INT 21H  
 JMP END  
 END:  
 MAIN ENDP  
END MAIN



Input and output:

Input: The input was static for the programme, We’ve put 1 in the AX register and -1 in the BX register

|  |  |
| --- | --- |
| AX | 1 |
| BX | -1 |

Output: The output of the programme was based on the SF status, as the status for this example was SF = 1 the programme printed “1” as expected.

|  |  |
| --- | --- |
| JS loc | “1” |

Procedure:

1. Our goal is to check JS (Jump Signed) instruction’s workings, to do that we build a basic structure of assembly code.
2. We’ve used small model and put the stack pointer to 0100h, we took N as a data byte variable.
3. We’ve put **1 into the register AX**.
4. We’ve put **-1 into the register BX**.
5. Then we’ve used the IMUL (INTEGER MULTIPLICATION – SIGNED MULTIPLICATION) operation to multiply the numbers.
6. We know that “**IMUL Source**” will multiply the Source with whatever in the AX register.
7. **IMUL BX** gives us the multiplication result of AX and BX and puts the product in **DX:AX**.
8. AX multiplied by BX will result in **Decimal Product: -1** and the **HEX Product will be: FFFFFFFF**
9. Status of **DX:AX = FFFFFFFF (HEX)**
10. Status of **DX:AX** = **1111111111111111 (BIN)**
11. So the **MSB is 1.**
12. So the **SF = 1.**
13. **JS loc** checks the SF and executes this instruction and **goes to loc.**
14. **Prints “1”** and jumps to the end: (**JMP end**)

Conclusion:   
This project was supposed to clear the concept of **SIGNED JUMP** instruction. While describing this instruction **JS** we’ve seen the use of **SIGNED FLAG** and **IMUL** instruction. The there are numerous amount of jump instruction we can see in x8086 processor’s instruction set. This was one of the most important ones to know. While working with signed operations (add, sub, mul, div) or even shifting operations the SF flag changes according to the negative result of the number and programmer can use this advantage of changing flag by using Signed Jump instruction