**Lab Manual # 06**

**Implementation of Other Arithmetic Instructions like MUL, DIV and Jump**

**MUL**

8-bit:

Command MUL BL

AX=AL\*BL

16-bit: When operand is a byte: AL \* operand.

MUL CX

(DX AX) =AX\*CX

When operand is a word: (DX AX) = AX \* operand.

Note that the product is stored in a register (or group of registers) twice the size of the operands. The operand can be a register or a memory operand.

****

**Example 1:**

Mov al, 5h

Mov bl, 10h

Mul bl ; AX = 0050h, CF = 0

**Example 2:**

mov dx,100h

mov ax,2000h

mul dx ; DX:AX = 00200000h, CF=1

\*The Carry flag indicates whether or not the upper half of the product contains significant digits.

**DIV:**

Div CL

AX is divided by CL

8 bit: AX= dividend, divided by contents of any 8-bit register or memory location

After division:

* AL= Quotient
* AH= Remainder

16 bit: DX-AX, divided by any 16-bit register or memory location

* AX= Quotient
* DX= Remainder

**Example 3:**

mov ax,0024h ; dividend

mov cl,0004h ; divisor

div cl ; Ax= 00 09

**Example 4:**

mov dx,0 ; clear dividend

mov ax,8003h ; dividend

mov cx,100h ; divisor

div cx

**Gates:**

* NOT - Reverse each bit of operand.
* NEG - Make operand negative (two's complement).
* AND - AND operator gives 1 only if both operands are 1.

**Example 5:**

mov dx,0 ; clear dividend

mov ax,8003h ; dividend

mov cx,100h ; divisor

**JUMP instructions: -**

There are two type of JUMP instructions:

1. Unconditional Jumps.

2. Conditional Jumps.

**1. Unconditional Jumps:** The basic instruction that transfers control to another point in the program is JMP.

The basic syntax of JMP instruction is: 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: or label2:

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

x1:

MOV AX, 1

Or

x2: MOV AX, 2

**Example 6:**

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:

**2. Conditional jumps:**

Unlike JMP instruction that does an unconditional jump, there are instructions that do a conditional jump (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
* 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 |

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:

It's required to compare 5 and 2, 5 - 2 = 3

the result is not zero (Zero Flag is set to 0).

Or

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).

here's an example of CMP instruction and conditional jump:

**Example: 7**

mov al, 25h ; set al to 25

mov bl, 10h ; set bl to 10.

cmp al, bl ; compare al - bl

je equal ; jump if al = bl (zf = 1).

Mov CX, BX ; if it gets here, then al <> bl,

jmp stop ; so print 'n', and jump to stop

equal: ; if gets here,

Mov CX, AX ; then al = bl, so print 'y'

stop:

**Lab Tasks**

**Execute the following tasks CLO [1]**

**Task 1:**

What will be the hexadecimal values of DX, AX, and the Carry flag after the following instructions execute?

mov ax,1234h

mov bx,100h

mul bx

**Task 2:**

What will be the hexadecimal values of DX and AX after the following instructions execute?

mov dx,0087h

mov ax,6000h

mov bx,100h

div bx

**Task 3:**

Take two binary numbers such as11111111

11111111

Perform two’s complement of the first number and save it to one of the register. Then, reverse the bits of second binary number and save it to the other register.

1. Take the value 0 in third register and perform the AND operation with both of the results computed above.
2. Take the value 1 in third register and perform the AND operation with both of the results computed above.

**Task 5:**

Division of 8 bit numbers using immediate addressing mode?

Write a code to perform multiplication on 16 bit numbers in consecutive memory locations?

Write a code to add 16 bit numbers and find the average of numbers?

**Task 6:**

Translate the high-level language assignment statement: A=5×A+12×B

Let A and B be word variables, and suppose there is no overflow.

**Task 7:**

Try the example no 7 with different numbers for AL and BL, open flags by clicking on flags button, use single step and see what happens