# *Signed DIV Instructions*

Signed integer division in MASM is similar to unsigned integer division, with one key difference: the dividend must be sign-extended before the division takes place. This is because the IDIV instruction (signed integer division) treats the dividend as a signed integer, and the result of the division is also a signed integer.

To sign-extend a number means to copy the sign bit of the number to all of the higher bits of the number.

This can be done using the:

- **CWD instruction (convert word to doubleword)**
- **CBW instruction (convert byte to word).**

The following MASM code shows how to sign-extend a 16-bit integer and then perform signed integer division:

```
.data
    wordVal SWORD -101      ;009Bh
.code
    mov eax, 0              ;EAX = 00000000h
    mov ax, wordVal         ;EAX = 0000009Bh (+155)
    cwd                     ;EAX = FFFFFF9Bh (-101)
    mov bx, 2               ;EBX is the divisor
    idiv bx                 ;divide EAX by BX
```

In this code, the CWD instruction is used to sign-extend the AX register into the EAX register.

This ensures that the EAX register contains the correct signed value of -101 before the division operation is performed.

The IDIV instruction then divides the EAX register by the EBX register and stores the result in the EAX register.

The following table shows the results of the division operation:

| Dividend | Divisor | Quotient | Remainder |
|---|---|---|---|
| -101 | 2 | -50 | 1 |

The quotient of the division operation is -50 and the remainder is 1.

It is important to note that the **IDIV instruction** can also be used to perform unsigned integer division.

However, in this case, the dividend does not need to be sign-extended.

***==========***

## *Sign Extension Instructions (CBW, CWD, CDQ)*

***==========***

The CBW, CWD, and CDQ instructions are sign extension instructions that are used to extend the sign bit of a smaller integer to a larger integer.

## *CBW*

The CBW instruction (convert byte to word) extends the sign bit of the AL register into the AH register.

This means that if the AL register contains a negative byte value, the AH register will be set to FFh. Otherwise, the AH register will be set to 00h.

The following MASM code shows how to use the CBW instruction:

```
.data
    byteVal SBYTE -101
    ;9Bh
.code
    mov al, byteVal      ;AL = 9Bh
    cbw                  ;AX = FF9Bh
```

In this code, the CBW instruction is used to extend the sign bit of the AL register into the AH register. After the CBW instruction is executed, the AX register will contain the value FF9Bh, which is the signed representation of the number -101.

---

## *CWD*

The CWD instruction (convert word to doubleword) extends the sign bit of the AX register into the DX register.

This means that if the AX register contains a negative word value, the DX register will be set to FFh. Otherwise, the DX register will be set to 00h.

The following MASM code shows how to use the CWD instruction:

```
.data
    wordVal SWORD -101
; FF9Bh
.code
    mov ax, wordVal ; AX = FF9Bh
    cwd ; DX:AX = FFFFFF9Bh
```

In this code, the CWD instruction is used to extend the sign bit of the AX register into the DX register. After the CWD instruction is executed, the DX:AX registers will contain the value FFFFFF9Bh, which is the signed representation of the number -101.

---

## *CDQ*

The CDQ instruction (convert doubleword to quadword) extends the sign bit of the EAX register into the EDX register.

This means that if the EAX register contains a negative doubleword value, the EDX register will be set to FFh. Otherwise, the EDX register will be set to 00h.

The following MASM code shows how to use the CDQ instruction:

```
.data
    dwordVal SDWORD -101
    ;FFFFFF9Bh
.code
    mov eax, dwordVal    ;EAX = FFFFFF9Bh
    cdq                  ;EDX:EAX = FFFFFFFFFFFFFF9Bh
```

---

In this code, the CDQ instruction is used to extend the sign bit of the EAX register into the EDX register. After the CDQ instruction is executed, the EDX:EAX registers will contain the value FFFFFFFFFFFFFF9Bh, which is the signed representation of the number -101.

## *When to use sign extension instructions*

Sign extension instructions are typically used in the following situations:

When performing signed integer arithmetic operations. When converting a signed integer from a smaller type to a larger type.

When passing a signed integer to a function that expects a signed integer parameter.

For example, if you are writing a function that calculates the average of two signed integers, you would need to use a sign extension instruction to ensure that the two integers are converted to the same type before the division operation is performed.

to the same type before the division operation is performed.

## *Conclusion*

Sign extension instructions are a powerful tool that can be used to ensure that signed integers are handled correctly. By understanding how to use these instructions, you can write more efficient and reliable code.