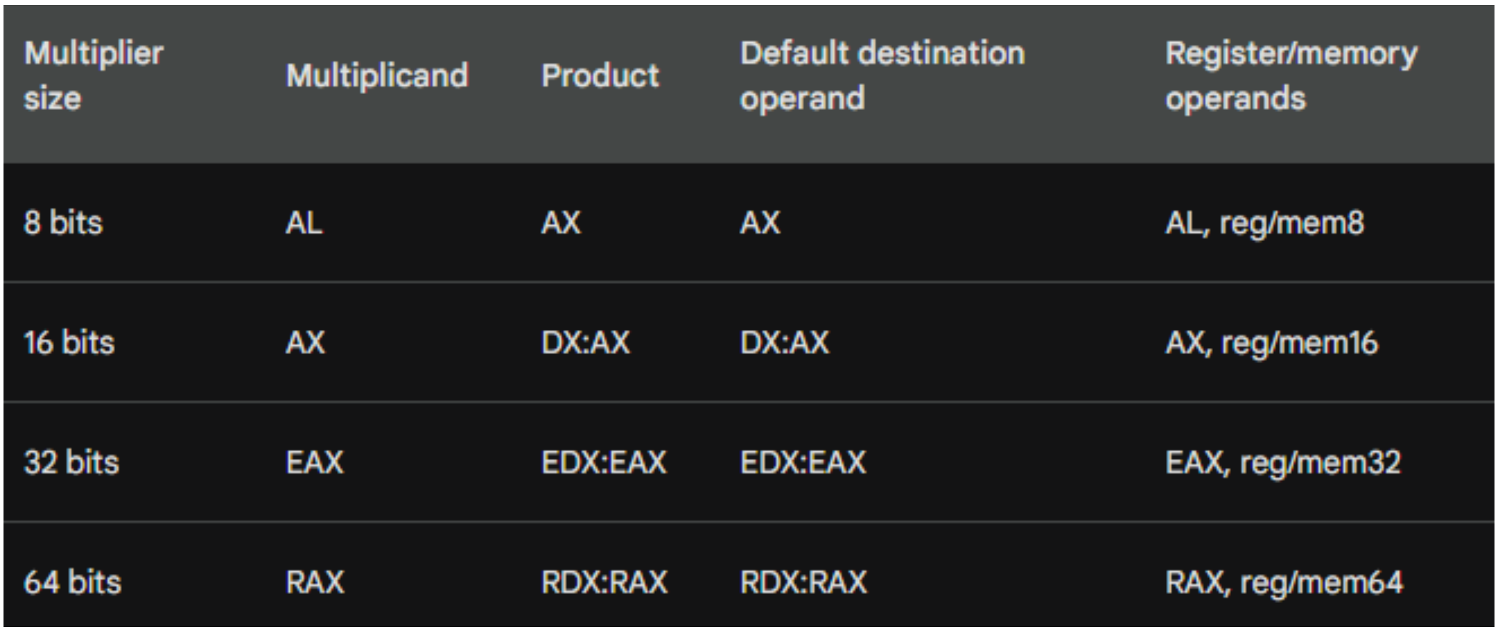
MUL Operator

The MUL instruction performs unsigned integer multiplication.

It has three versions, which multiply an 8-bit operand by AL, a 16-bit operand by AX, or a 32-bit operand by EAX.

The multiplicand and multiplier must always be the same size, and the product is twice their size.

The following table shows the default multiplicand and product, depending on the size of the multiplier:



Multiplier size	Multiplicand	Product	Default destination operand	Register/memory operands
8 bits	AL	AX	AX	AL, reg/mem8
16 bits	AX	DX:AX	DX:AX	AX, reg/mem16
32 bits	EAX	EDX:EAX	EDX:EAX	EAX, reg/mem32
64 bits	RAX	RDX:RAX	RDX:RAX	RAX, reg/mem64

Because the destination operand is twice the size of the multiplicand and multiplier, overflow cannot occur.

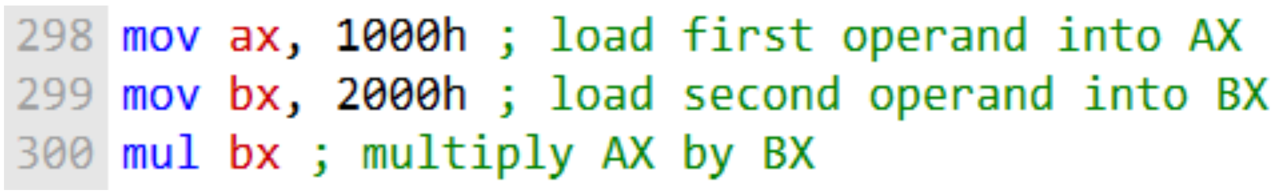
However, the MUL instruction sets the Carry and Overflow flags if the upper half of the product is not equal to zero.

The Carry flag is ordinarily used for unsigned arithmetic, so it can be used to detect overflow in the MUL instruction.

For example, if AX is multiplied by a 16-bit operand, the product is stored in the combined DX and AX registers.

If DX is not equal to zero after the multiplication operation, then the product will not fit into the lower half of the implied destination operand, and the Carry flag will be set.

Here is an example of how to use the MUL instruction to multiply two 16-bit operands:

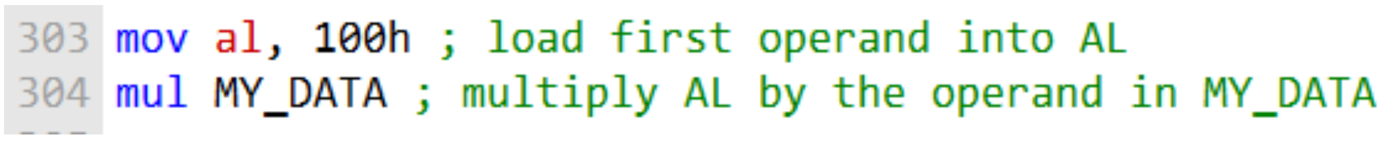


```
mov ax, 1000h ; load first operand into AX
mov bx, 2000h ; load second operand into BX
mul bx ; multiply AX by BX
```

After the multiplication operation, the product will be stored in the combined DX and AX registers. If DX is not equal to zero, then the product will not fit into the lower half of the AX register, and the Carry flag will be set.

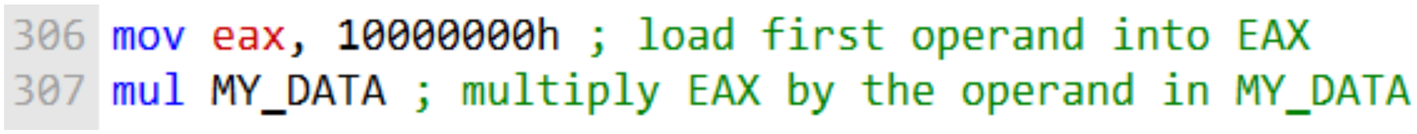
As you can see, the MUL instruction supports register and memory operands for all multiplier sizes. This gives you a lot of flexibility in how you use the instruction.

For example, the following assembly code multiplies the AL register by the 8-bit operand in memory location MY_DATA:



```
mov al, 100h ; load first operand into AL
mul MY_DATA ; multiply AL by the operand in MY_DATA
```

The following assembly code multiplies the EAX register by the 32-bit operand in memory location MY_DATA:



```
mov eax, 10000000h ; load first operand into EAX
mul MY_DATA ; multiply EAX by the operand in MY_DATA
```

The MUL instruction is a powerful tool for performing unsigned integer multiplication on the x86 architecture. It is important to understand the different versions of the instruction and how to use the Carry flag to detect overflow.

A good reason for checking the Carry flag after executing MUL is to know whether the upper half of the product can safely be ignored.

The MUL instruction multiplies two operands and stores the product in two registers. If the product is too large to fit in the destination registers, the Carry flag is set.

For example, if you multiply two 16-bit operands, the product will be 32 bits.

The MUL instruction will store the lower 16 bits of the product in the AX register and the upper 16 bits of the product in the DX register.

If the upper 16 bits of the product are zero, then you can safely ignore them. However, if the upper 16 bits of the product are non-zero, then you will need to use the DX register to store the entire product.

You can check the Carry flag to determine whether the upper half of the product is zero.

If the Carry flag is clear, then the upper half of the product is zero and you can safely ignore it.

However, if the Carry flag is set, then the upper half of the product is non-zero and you will need to use the DX register to store the entire product.

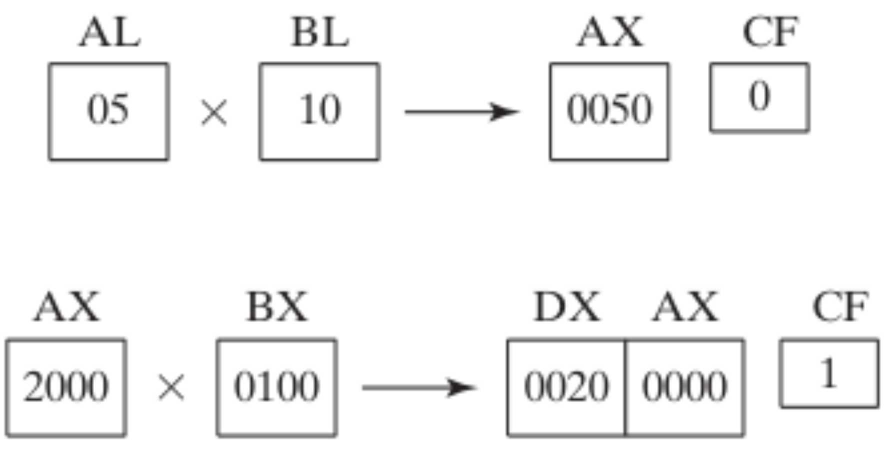
Here is an example of how to check the Carry flag after executing MUL:

```
mov ax, 1000h   ;load first operand into AX
mov bx, 2000h   ;load second operand into BX
mul bx ; multiply AX by BX

;check the Carry flag
jc overflow     ;jump to overflow handler if the Carry flag is set

;the upper half of the product is zero, so we can ignore it
;use the AX register to store the lower half of the product
```

Example 1:

```
; 8-bit multiplication
mov al, 5h
mov bl, 10h
mul bl
; AX = 0050h, CF = 0

; 16-bit multiplication
mov ax, 2000h
mul val2
; DX:AX = 00200000h, CF = 1
```

For this code, this is the data flow:



The following statements multiply 12345h by 1000h, producing a 64-bit product in the combined EDX and EAX registers:

```
mov eax, 12345h
mov ebx, 1000h
mul ebx
; EDX:EAX = 0000000012345000h, CF = 0
```

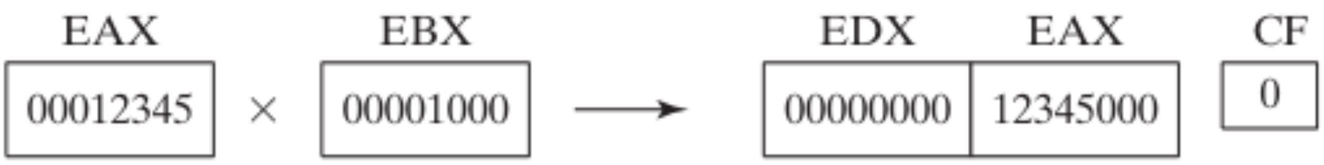
The MUL instruction multiplies two unsigned integers and stores the product in two registers: the low-order half of the product is stored in the EAX register, and the high-order half of the product is stored in the EDX register.

The Carry flag is set if the product is too large to fit in the destination registers.

In this case, the product of 12345h and 1000h is 12345000h, which is a 64-bit value. The product fits in the EDX and EAX registers, so the Carry flag is clear.

The following diagram illustrates the movement between registers:

```
Before:
EAX = 12345h
EBX = 1000h
EDX = 0

After:
EAX = 0000h
EBX = 1000h
EDX = 12345h
```



Here is a summary of the MUL instruction:

• The MUL instruction multiplies two unsigned integers and stores the product in two registers.

• The multiplicand (the first operand) is stored in the AL register (for 8-bit multiplication) or the AX register (for 16-bit multiplication).

• The multiplier (the second operand) is stored in another register or in memory.

• The product is stored in two registers: the low-order half of the product is stored in the AL register (or the AX register for 16-bit multiplication), and the high-order half of the product is stored in the AH register (or the DX register for 16-bit multiplication).

• The Carry flag is set if the product is too large to fit in the destination registers.

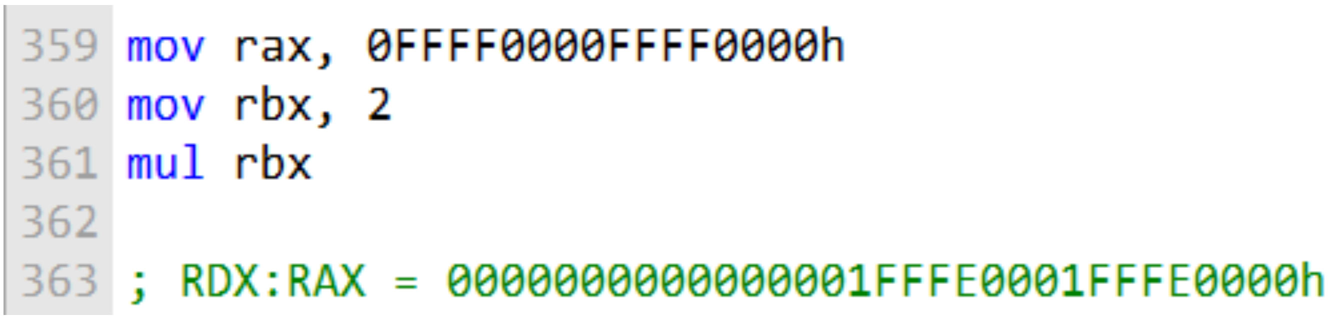
====================

MUL in 64-bit mode

=====================

In 64-bit mode, the MUL instruction can be used to multiply two 64-bit operands. The result is a 128-bit product, which is stored in the RDX:RAX register pair.

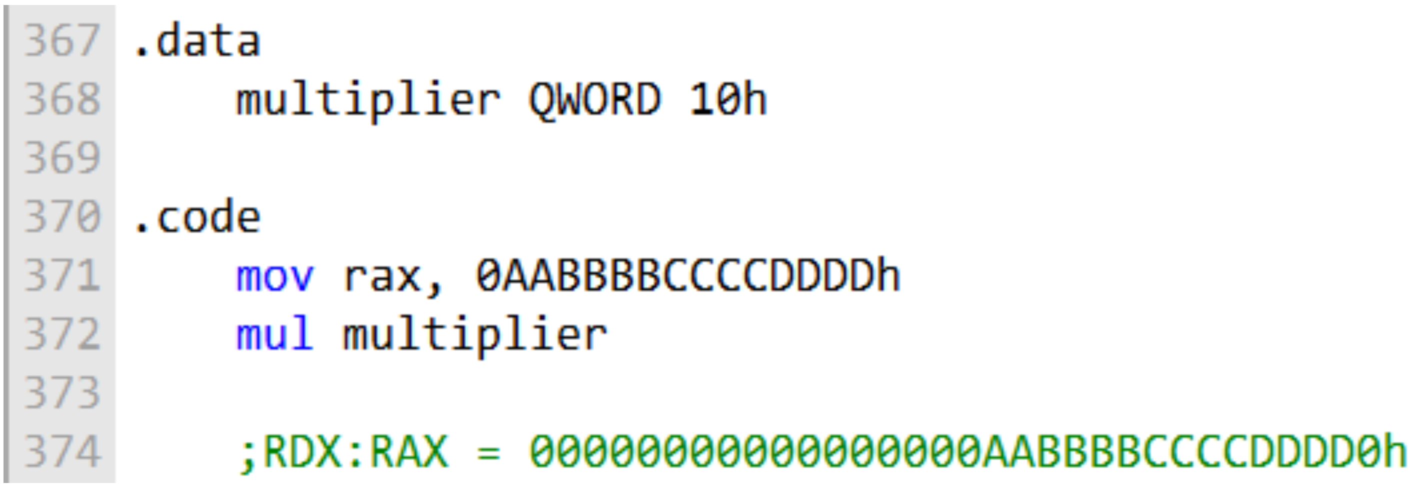
The following example shows how to use the MUL instruction to multiply RAX by 2:



```
mov rax, 0FFFF0000FFFF0000h
mov rbx, 2
mul rbx

; RDX:RAX = 00000000000000001FFFE0001FFFE0000h
```

In this example, the highest bit of RAX spills over into the RDX register because the product is too large to fit in a 64-bit register.

The following example shows how to use the MUL instruction to multiply RAX by a 64-bit memory operand:



```
.data
    multiplier QWORD 10h

.code
    mov rax, 0AABBBBCCCCDDDDh
    mul multiplier

    ;RDX:RAX = 00000000000000000AABBBBCCCCDDDD0h
```

In this example, the product is a **128-bit value**, but both halves of the product fit in RAX and RDX because the product is less than **2^{128}.**

Here is a more in-depth explanation of what happens when the MUL instruction is used in 64-bit mode:

The RAX and RDX registers are multiplied together. The low-order 64 bits of the product are stored in RAX. The high-order 64 bits of the product are stored in RDX. The Carry flag is set if the product is too large to fit in RAX and RDX.