IMUL Operation

The IMUL instruction performs signed integer multiplication.

This means that it multiplies two integers and takes into account their signs. Unlike the MUL instruction, the IMUL instruction preserves the sign of the product.

How does the IMUL instruction work?

The IMUL instruction works by first sign extending the highest bit of the lower half of the product into the upper bits of the product.

This ensures that the sign of the product is the same as the sign of the multiplicand (the number being multiplied).

What are the different formats of the IMUL instruction?

The IMUL instruction has three formats:

```
One-operand format
Two-operand format
Three-operand format
```

The **one-operand format of the IMUL instruction** multiplies the operand by itself and stores the product in the same operand. This is equivalent to squaring the operand.

The **two-operand format of the IMUL instruction** multiplies the two operands and stores the product in the first operand. The second operand can be a register, a memory operand, or an immediate value.

The **three-operand format of the IMUL instruction** multiplies the first and third operands and stores the product in the second operand. The first and third operands can be registers or memory operands, and the second operand must be a register.

When should I use the IMUL instruction?

You should use the IMUL instruction whenever you need to perform signed integer multiplication. This is especially important when you

need to preserve the sign of the product.

Here are some examples of how to use the IMUL instruction:

```
; One-operand format
imul eax
; eax = eax * eax

; Two-operand format
imul eax, ebx
; eax = eax * ebx

; Three-operand format
imul eax, ebx, ecx
; eax = ebx * ecx
```

Important things to keep in mind:

- The IMUL instruction can generate overflow. If the product of the two operands is too large to fit in the destination operand, the Overflow flag is set.
- The IMUL instruction can also generate a carry. If the highest bit of the product is set, the Carry flag is set.
- It is important to check the Overflow and Carry flags after performing an IMUL operation to ensure that the result is correct.

```
;Multiply two 16-bit integers and store the product in AX
mov ax, 1000h
mov bx, 2000h
imul bx

;Multiply two 32-bit integers and store the product in EDX:EAX
mov eax, 10000000h
mov ebx, 20000000h
imul ebx

;Multiply a 32-bit integer by an 8-bit immediate value and store the product in AX
mov ax, 1000h
imul ax, 2

;Multiply a 32-bit integer by a 32-bit immediate value and store the product in EDX:EAX
mov eax, 10000000h
imul eax, 20000000h

;Multiply two 16-bit integers and store the product in memory
mov ax, 1000h
mov bx, 2000h
imul bx
mov [word_variable], ax

;Multiply two 32-bit integers and store the product in memory
mov eax, 10000000h
mov ebx, 20000000h
imul ebx
mov [dword_variable], eax
```

The first two lines of code move the values 1000h and 2000h into the AX and BX registers, respectively. The third line then uses the IMUL instruction to multiply the two values and store the product in the AX register.

The next two lines of code do the same thing, but with 32-bit integers. The IMUL instruction in this case stores the product in the EDX:EAX register pair.

The next two lines of code multiply a 32-bit integer by an 8-bit and 32-bit immediate value, respectively. The IMUL instruction in these cases stores the product in the AX and EDX:EAX register pair, respectively.

The last two lines of code multiply two 16-bit and 32-bit integers and store the product in memory. The IMUL instruction in these cases stores the product in the word and dword variable, respectively.

It is important to note that the IMUL instruction can generate overflow. This means that if the product of the two operands is too large to fit in the destination operand, the Overflow flag is set. It is important to check the Overflow flag after performing an IMUL

operation to ensure that the result is correct.

```
;Multiply two 32-bit integers and store the product in EDX:EAX
mov eax, 80000000h
mov ebx, 80000000h
imul ebx
jc overflow_label     ; Check the Overflow flag
jc carry_label         ; Check the Carry flag
;No overflow or carry occurred, so the result is correct
;Exit the program
int 3
overflow_label:
;Overflow occurred, so the result is incorrect
;Handle the overflow error
;Exit the program
int 3
carry_label:
;Carry occurred, but the result may still be correct
;Check the highest bit of the product
test edx, edx
jz carry_ok
;The highest bit of the product is set, so the result is incorrect
;Handle the carry error
;Exit the program
int 3
carry_ok:
;The highest bit of the product is not set, so the result is correct
;Continue with the program
```

The first two lines of code move the values 1000h and 2000h into the AX and BX registers, respectively. The third line then uses the IMUL instruction to multiply the two values and store the product in the AX register.

The next two lines of code do the same thing, but with 32-bit integers. The IMUL instruction in this case stores the product in the EDX:EAX register pair.

The next two lines of code multiply a 32-bit integer by an 8-bit and 32-bit immediate value, respectively. The IMUL instruction in these cases stores the product in the AX and EDX:EAX register pair, respectively.

The last two lines of code multiply two 16-bit and 32-bit integers and store the product in memory. The IMUL instruction in these cases stores the product in the word and dword variable, respectively.

It is important to note that the IMUL instruction can generate overflow. This means that if the product of the two operands is too large to fit in the destination operand, the Overflow flag is set. It is important to check the Overflow flag after performing an IMUL operation to ensure that the result is correct.

```
;Multiply two 32-bit integers and store the product in EDX:EAX
mov     eax, 80000000h      ;Load the first integer into EAX
mov     ebx, 80000000h      ;Load the second integer into EBX
imul    ebx                 ;Multiply EAX by EBX; result in EDX:EAX
;Check for overflow:
jc      overflow_label      ;Jump if the Overflow flag is set
;Check for carry:
jc      carry_label         ;Jump if the Carry flag is set
;No overflow or carry occurred, so the result is correct
;Exit the program
int     3
overflow_label:
;Overflow occurred, so the result is incorrect
;Handle the overflow error
;Exit the program
int     3
carry_label:
;Carry occurred, but the result may still be correct
;Check the highest bit of the product
test    edx, edx            ;Test the value in EDX
jz      carry_ok            ;Jump if the highest bit of the product is not set
;The highest bit of the product is set, so the result is incorrect
;Handle the carry error
;Exit the program
int     3
carry_ok:
;The highest bit of the product is not set, so the result is correct
;Continue with the program
```

In this example, we multiply two 32-bit integers, 80000000h and 80000000h. The product of these two integers is 6400000000h, which is too large to fit in a 32-bit register. Therefore, the Overflow flag is set.

We then check the Carry flag. The Carry flag is set if the highest

bit of the product is set. In this case, the highest bit of the product is not set, so the Carry flag is not set.

We then check the Overflow flag to see if overflow occurred. If overflow occurred, the result of the multiplication is incorrect. In this case, overflow occurred, so the result is incorrect.

We could handle the overflow error in a number of ways. For example, we could print an error message and exit the program. Or, we could try to scale the result down so that it fits in a 32-bit register.

If overflow did not occur, we need to check the Carry flag. If the Carry flag is set, the highest bit of the product is set. This may or may not indicate that the result is incorrect.

In this example, the highest bit of the product is not set, so the result is correct. We can then continue with the program.

It is important to note that this is just a simple example of how to illustrate the IMUL instruction overflow and carry flag in MASM. There are many other ways to handle overflow and carry errors.

==========

IMUL in 64-Bit Mode

==========

```
488 ;64-bit mode IMUL example
489 mov rax,-4
490 mov rbx,4
491 imul rbx     ;RDX = 0FFFFFFFFFFFFFFFFh, RAX = -16
```

In this example, we multiply the 64-bit register RBX by the 64-bit register RAX. The product of these two integers is -64, which is a 128-bit value. The IMUL instruction in this case stores the product in the RDX:RAX register pair.

The RDX register stores the high-order 64 bits of the product, and the RAX register stores the low-order 64 bits of the product.

In this case, the high-order 64 bits of the product are all ones, so the RDX register is filled with the value 0xFFFFFFFF...FFFFh. The low-order 64 bits of the product are equal to -64, so the RAX register is filled with the value FFFFFFFFFFFFFFC0h.

```
;64-bit mode IMUL example
.data
    multiplicand QWORD -16
.code
    imul rax, multiplicand, 4 ;RAX = FFFFFFFFFFFFFFC0 (-64)
```

In this example, we multiply the 64-bit memory operand multiplicand by the immediate value 4. The product of these two operands is -64, which is a 64-bit value. The IMUL instruction in this case stores the product in the RAX register.

The multiplicand memory operand is defined in the data section of the program. It is a QWORD variable, which means that it is 64 bits wide. The immediate value 4 is a 32-bit value, but it is automatically promoted to 64 bits before the multiplication operation is performed.

The IMUL instruction in this case stores the product in the RAX register, which is a 64-bit register. Therefore, the RAX register will be filled with the value FFFFFFFFFFFFFFC0h after the IMUL instruction is executed.

Unsigned multiplication

The IMUL instruction can also be used to perform unsigned multiplication. However, there is a small disadvantage to doing so: the Carry and Overflow flags will not indicate whether the upper half of the product is equal to zero.

This is because the IMUL instruction always sign-extends the product, even if the operands are unsigned. This means that the Carry and Overflow flags will be set if the high-order bit of the product is set, even if the product is a valid unsigned integer.

The IMUL instruction is a powerful instruction that can be used to perform signed and unsigned multiplication in 64-bit mode. However, it is important to be aware of the fact that the Carry and Overflow flags will not indicate whether the upper half of the product is equal to zero when performing unsigned multiplication.

equal to zero when performing unsigned multiplication.

MUL Examples in Depth

MUL Overflow

The MUL instruction can generate overflow if the product of the two operands is too large to fit in the destination operand. For example, the following code will generate overflow:

```
mov ax, -32000
imul ax, 2
```

The product of -32000 and 2 is -64000, which is too large to fit in a 16-bit register. Therefore, the Overflow flag will be set after the IMUL instruction is executed.

MUL Signed and Unsigned Examples

```
;Multiply 48 by 4 and store the product in AX.
mov al, 48
mov bl, 4
mul bl
;AX = 00C0h, OF = 1

;Multiply -4 by 4 and store the product in AX.
mov al, -4
mov bl, 4
mul bl
;AX = FFF0h, OF = 0

;Multiply 48 by 4 and store the product in DX:AX.
mov ax, 48
mov bx, 4
mul bx
;DX:AX = 000000C0h, OF = 0

;Multiply 4,823,424 by -423 and store the product in EDX:EAX.
mov eax, 4823424
mov ebx, -423
mul ebx
;EDX:EAX = FFFFFFFF86635D80h, OF = 0
```

Two-Operand IMUL Instructions

The two-operand IMUL instruction uses a destination operand that is the same size as the multiplier. Therefore, it is possible for signed overflow to occur. Always check the Overflow flag after executing these types of IMUL instructions. The following code examples demonstrate two-operand IMUL instructions:

```
;Multiply -16 by 2 and store the product in AX.
mov ax, -16
imul ax, 2
; AX = -32

;Multiply -32 by 2 and store the product in AX.
imul ax, 2
;AX = -64

;Multiply -64 by word1 and store the product in BX.
;word1 is a 16-bit variable.
mov bx, word1
imul bx, word1, -16
;BX = word1 * -16

;Multiply -64 by dword1 and store the product in BX.
;dword1 is a 32-bit variable.
mov bx, dword1
imul bx, dword1, -16
;BX = dword1 * -16

;Multiply dword1 by -2000000000 and store the product in BX.
;This instruction will generate overflow because the product is too large to fit in a 32-bit register.
imul bx, dword1, -2000000000
;signed overflow!
```

Three-Operand IMUL Instructions

The three-operand IMUL instruction uses a destination operand that is the same size as the first operand. The first operand is multiplied by the second operand and the product is stored in the third operand. The following code examples demonstrate three-operand IMUL instructions:

```
;Multiply word1 by -16 and store the product in BX.
mov bx, word1
imul bx, word1, -16
;BX = word1 * -16

;Multiply dword1 by -16 and store the product in BX.
mov bx, dword1
imul bx, dword1, -16
;BX = dword1 * -16

;Multiply dword1 by -2000000000 and store the product in BX.
;This instruction will generate overflow because the product is too large to fit in a 32-bit register.
mov bx, dword1
imul bx, dword1, -2000000000
;signed overflow!
```

The MUL and IMUL instructions are powerful instructions that can be used to perform signed and unsigned multiplication.

However, it is important to be aware of the fact that these instructions can generate overflow.

Always check the Overflow flag after executing these types of instructions to ensure that the result is correct.