Shift Left and Shift Right

==========

Shift and Rotate Instructions:

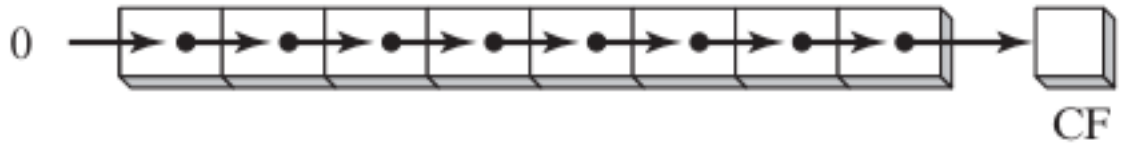
==========

Bit shifting involves moving bits within an operand either to the left or right. The x86 processor architecture offers a wide range of shift and rotate instructions, each with specific purposes and effects on flags like Overflow and Carry. Here are a few common shift and rotate instructions on x86:

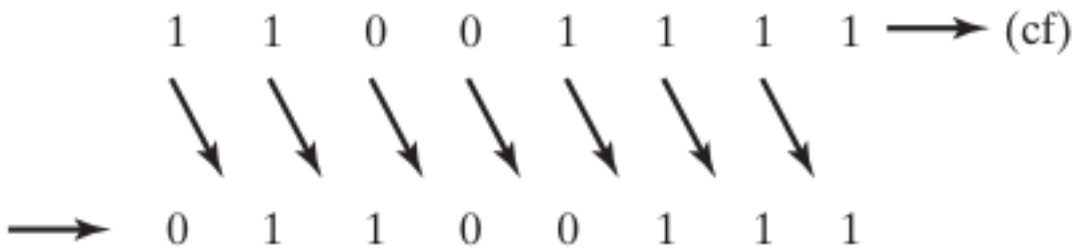
==========

SHL(Shift Left):

These instructions shift bits left (SHL) or right (SHR) within an operand. SHL multiplies the value by 2 for each shift left, while SHR effectively divides the value by 2 for each shift right.

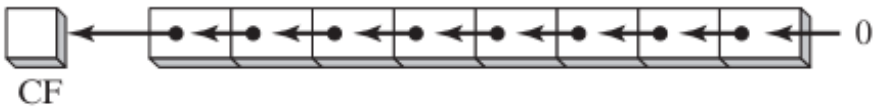


The following illustration shows a single logical right shift on the binary value 11001111, producing 01100111. The lowest bit is shifted into the Carry flag:

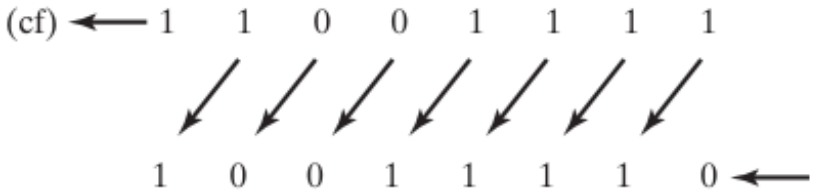


The **SHL (shift left) instruction** performs a logical left shift on the destination operand, filling the lowest bit with 0. The highest bit

is moved to the Carry flag, and the bit that was in the Carry flag is discarded:



If you shift 11001111 left by 1 bit, it becomes 10011110:




```
SHL destination, count
```

```
SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL
```

The SHL instruction can **only be used to shift integers**, not floating-point numbers.

The imm8 operand must be between 0 and 7, inclusive. For example, the following instruction is invalid:

```
shl eax, 8
```

If the imm8 operand is greater than 7, the shift count will be wrapped around to the range 0-7. For example, the following instruction is equivalent to the instruction shl eax, 1:

```
shl eax, 9
```

If the CL register is used as the shift count operand, it must contain a value between 0 and 31, inclusive. For example, the following instruction is invalid:

```
shl eax, cl ; cl = 32
```

If the CL register contains a value greater than 31, the shift count will be wrapped around to the range 0-31. For example, the following instruction is equivalent to the instruction shl eax, 1:

```
shl eax, cl ; cl = 33
```

Here is a more clear explanation of the SHL instruction:

• The SHL instruction shifts the bits of the destination operand to the left by the specified number of bits.

• The highest bit of the destination operand is shifted out and copied into the Carry flag.

• The lowest bit position of the destination operand is assigned zero.

The following table shows the possible operands for the SHL instruction:

Name	Description
reg	A general-purpose register.
mem	A memory location.
imm8	An immediate value between 0 and 7.
CL	The CL register.

The notes you provided are not clear because they do not explicitly state the following:

• The **SHL instruction** can only be used to shift integers, **not** floating-point numbers.

• The **imm8** operand must be **between 0 and 7, inclusive.** For example, the following instruction is invalid:

• **shl eax, 8** If the imm8 operand is greater than 7, the shift count will be wrapped around to the range 0-7.

For example, the following instruction is equivalent to the instruction shl eax, 1:

```
07 mov bl, 8Fh ; BL = 10001111b
08 shl bl, 1
09 ; CF = 1, BL = 00011110b
```

After the SHL instruction is executed, the Carry flag will be set to 1 and the BL register will contain the value 00011110b.

When the SHL instruction is used to shift a value to the left multiple times, the Carry flag will contain the last bit to be shifted out of the most significant bit (MSB).

For example, the following code shows how to shift the value of the AL register to the left by two bits:

```
13 mov al, 10000000b
14 shl al, 2
15 ; CF = 0, AL = 00000000b
```

After the first SHL instruction is executed, the Carry flag will be set to 1 and the AL register will contain the value 01000000b.

After the second SHL instruction is executed, the Carry flag will be set to 0 and the AL register will contain the value 00000000b.

The SHL instruction can be used to perform a variety of operations, such as multiplying a value by two, converting a binary number to a decimal number, and packing and unpacking data.

When a value is shifted rightward multiple times, the **Carry flag contains the last bit to be shifted out of the least significant bit (LSB).**

The image you sent is correct. It shows that shifting the binary number 00001010 (decimal 10) to the left by two bits is the same as multiplying it by 2^2.

This is because shifting a binary number to the left by one bit is the same as multiplying it by 2. Shifting to the left by two bits is the same as multiplying by 2^2, and so on.

```
mov  dl,10                    ; before:  00001010
shl  dl,2                     ; after:   00101000
```

After the SHL instruction is executed, the BL register will contain the value 20.

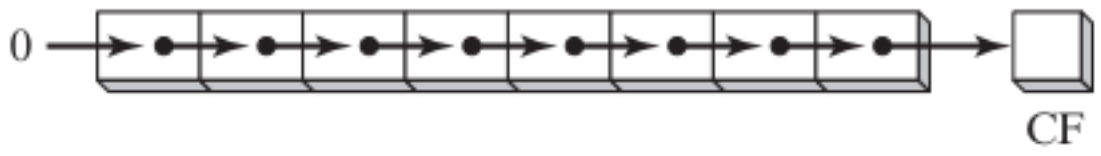
Bitwise multiplication is often used in graphics and signal processing applications. For example, it can be used to scale images, rotate images, and apply filters to images.

==========

Shift Right Instruction

==========

The SHR instruction performs a logical right shift on the destination operand, replacing the highest bit with a 0.



The lowest bit is copied into the Carry flag, and the bit that was previously in the Carry flag is lost.

```
mov  al,0D0h                    ; AL = 11010000b
shr  al,1                       ; AL = 01101000b, CF = 0
```

In a multiple shift operation, the last bit to be shifted out of position 0 (the LSB) ends up in the Carry flag:

```
mov  al,00000010b
shr  al,2                       ; AL = 00000000b, CF = 1
```

Bitwise division is a way of dividing two numbers by shifting the bits of the dividend (the number being divided) to the right by the number of bits in the divisor.

The result of this operation is the quotient of the two numbers, rounded down to the nearest integer.

For example, to divide 32 by 2^1, we would shift the bits of 32 to the right by 1 bit. This would result in the value 16, which is the quotient of 32 divided by 2^1.

Here is a step-by-step explanation of bitwise division:

1. Write the dividend and divisor in binary form.
2. Shift the bits of the dividend to the right by the number of bits in the divisor.
3. The result of the shift is the quotient of the two numbers, rounded down to the nearest integer.

For example, to divide 32 by 2^1, we would do the following:

```
mov  dl,32
shr  dl,1
```

Before: 0 0 1 0 0 0 0 0 = 32

After: 0 0 0 1 0 0 0 0 = 16

In the following example, 64 is divided by 2^3:

```
mov  al,01000000b                ; AL = 64
shr  al,3                        ; divide by 8, AL = 00001000b
```

1. Write the dividend and divisor in binary form:

```
Dividend: 100000
Divisor: 00100
```

2. Shift the bits of the dividend to the right by 1 bit:

```
100000 >> 1 = 010000
```

3. The result of the shift is the quotient of the two numbers, rounded down to the nearest integer:

```
Quotient = 010000 = 16
```

Bitwise division can be used to perform a variety of operations, such as:

- Dividing a number by a power of two.
- Converting a hexadecimal number to a decimal number.
- Calculating the remainder of a division operation.

Division of signed numbers by shifting is accomplished using the SAR instruction because it preserves the number's sign bit.

For example, if the AX register contains the value 0x1234, then the following instruction will shift the value to the right by one bit:

```
SHR AX, 1
```

After the instruction is executed, the AX register will contain the value 0x091A.

The highest bit of the original value (0x1) has been shifted out and lost, and the lowest bit of the original value (0x4) has been copied into the Carry flag.

The SHR instruction can be used to perform a variety of operations, including:

In this statement, what's being explained is how the SHR (Shift Right) instruction works using an example with the AX register containing the value 0x1234 (a 16-bit hexadecimal number).

When you apply the SHR instruction to this value, you are shifting the bits of the number to the right by one bit position. Here's a step-by-step breakdown:

1. Original Value: 0x1234 (binary: 0001001000110100)

2. Right Shift by One Bit: Shifting the bits to the right by one position results in "0000100100011010."

3. The highest bit in the original value (0x1) is the leftmost bit, and it gets shifted out and is "lost" because there's no longer space for it in the 16-bit AX register.

4. The lowest bit in the original value (0x4) is the rightmost bit, and it has been copied into the Carry flag. This means that after the shift, the Carry flag will hold the value 1 (indicating that the bit that was shifted out was a 1).

5. The new value in the AX register after the shift is 0x091A (binary: 0000100100011010).

Dividing a value by two

Converting a decimal number to a binary number Unpacking data.

The following table shows some examples of how to use the SHR

instruction:

Example	Description
SHR AX, 1	Shifts the value of the AX register to the right by one bit.
SHR BX, 2	Shifts the value of the BX register to the right by two bits.
SHR CX, 4	Shifts the value of the CX register to the right by four bits.
SHR DX, 8	Shifts the value of the DX register to the right by eight bits.

The SHR instruction can also be used to test the value of the Carry flag.

For example, the following instruction will test the value of the Carry flag and set the Zero flag if the Carry flag is clear:

```
SHR AX, 1
JZ TEST
```

If the Carry flag is clear, then the JZ instruction will jump to the TEST label. Otherwise, the JZ instruction will be skipped.

==========

Summary b4 we continue

==========

Certainly, you've provided a detailed explanation of shift and rotate instructions, specifically focusing on SHL (Shift Left) and SHR (Shift Right) instructions in the context of x86 assembly language programming. Here's a summary of the information you've provided:

Shift Left (SHL) Instruction:

• SHL shifts bits to the left within an operand.
• It multiplies the value by 2 for each left shift.
• The highest bit is moved to the Carry flag, and the lowest bit is set to 0.
• It can only be used for integers, not floating-point numbers.
• The imm8 operand must be between 0 and 7, inclusive.
• If imm8 is greater than 7, the shift count wraps around to 0-7.
• If the CL register is used, it must contain a value between 0 and 31.
• If CL exceeds 31, the shift count wraps around.
• It is often used for operations like multiplying a value by 2 and bitwise multiplication.

Shift Right (SHR) Instruction:

• SHR performs a logical right shift on the destination operand.
• It replaces the highest bit with 0 and copies the lowest bit to the Carry flag.
• In multiple shifts, the last bit shifted out ends up in the Carry flag.
• It can be used for operations like dividing a value by 2, converting decimal to binary, and unpacking data.
• SAR (Shift Arithmetic Right) is used for signed numbers to preserve the sign bit.

Bitwise Division:

• Bitwise division involves shifting bits of the dividend to the right by the number of bits in the divisor.
• The result is the quotient, rounded down to the nearest integer.
• It's used for operations like dividing by a power of two and converting hexadecimal to decimal.
• SAR is used for signed numbers to maintain the sign bit.

Since bitwise division (right shifting) and bitwise multiplication (left shifting) are essentially inverse operations of each other, you can focus on one of them in your notes and simply mention their relationship. Here's a more concise way to express this:

• **Bitwise Shifting:** This operation involves moving the bits of a

binary number.

- **Left Shifting:** Equivalent to multiplying by 2 to the power of the shift count.
- **Right Shifting:** Equivalent to dividing by 2 to the power of the shift count

Bitwise operations, including left and right shifts, are primarily related to binary data manipulation. They work with binary representations, and you can perform these operations on hexadecimal numbers, but the shifts operate at the binary level.

In contrast, when you perform bitwise operations on other number formats like decimal or hexadecimal, the operations affect their binary representations. Bitwise operations are versatile but fundamentally operate on binary data, while left and right shifts are more broadly applicable, working with various data types beyond binary numbers.