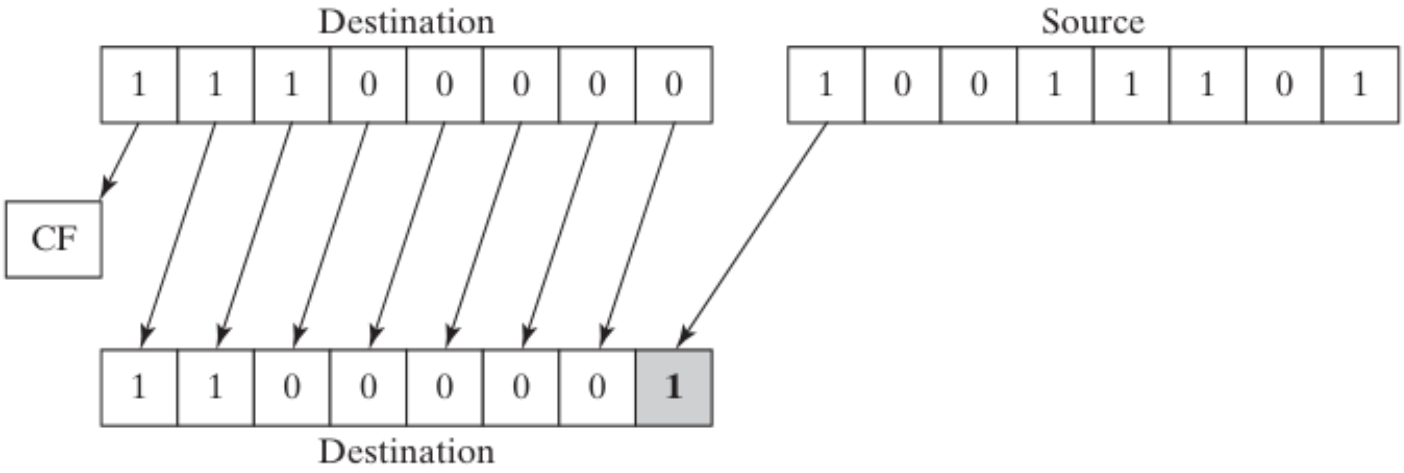
Shift Left Double and Shift Right Double

==========

SHLD (Shift Left Double)

==========

The **SHLD instruction** shifts a destination operand a given number of bits to the left, and fills the opened-up bit positions with the most significant bits of the source operand. The source operand is not affected.



The SHLD instruction has the following syntax:

```
SHLD dest, source, count
```

where:

- dest is the destination operand.
- source is the source operand.
- count is the number of bits to shift.

The count operand must be a value between 0 and 31, inclusive.

If count is 0, the destination operand is not shifted.

If count is 31, the destination operand is shifted all the way to the

left, and the source operand is copied into the destination operand.

The following table shows the effects of the SHLD instruction on the Sign, Zero, Auxiliary, Parity, and Carry flags:

Flag	Before	After
Sign	Sign of the destination operand	Sign of the shifted destination operand
Zero	Zero flag set if the shifted destination operand is 0	Zero flag set if the shifted destination operand is 0
Auxiliary	Auxiliary carry flag set if the carry out of bit 3 is 1	Auxiliary carry flag set if the carry out of bit 3 is 1
Parity	Parity flag set if the shifted destination operand has an even number of 1 bits	Parity flag set if the shifted destination operand has an even number of 1 bits
Carry	Carry flag set if the carry out of bit 31 is 1	Carry flag set if the carry out of bit 31 is 1

Here is an example of how to use the SHLD instruction:

```
21 mov eax, 0x12345678
22 mov ebx, 0xabcdef00
23 SHLD eax, ebx, 1
```

After the SHLD instruction, eax will contain the value 0x23456780.

The SHLD instruction can be used to perform a variety of tasks, such as:

- Shifting a value to the left to multiply it by a power of two.
- Shifting a value to the left to extract the most significant bits.
- Shifting a value to the left to prepare it for a bitwise operation.

==========

SHLD (Shift Right Double)

==========

The SHRD instruction shifts a destination operand a given number of bits to the right, and fills the opened-up bit positions with the least significant bits of the source operand. The source operand is not affected.

The SHRD instruction has the following syntax:

```
SHRD dest, source, count
```

where:

- dest is the destination operand.
- source is the source operand.
- count is the number of bits to shift.

The count operand must be a value between 0 and 31, inclusive.

If count is 0, the destination operand is not shifted.

If count is 31, the destination operand is shifted all the way to the right, and the source operand is copied into the destination operand.

The following table shows the effects of the SHRD instruction on the Sign, Zero, Auxiliary, Parity, and Carry flags:

Flag	Before	After
Sign	Sign of the destination operand	Sign of the shifted destination operand
Zero	Zero flag set if the shifted destination operand is 0	Zero flag set if the shifted destination operand is 0
Auxiliary	Auxiliary carry flag set if the carry out of bit 3 is 1	Auxiliary carry flag set if the carry out of bit 3 is 1
Parity	Parity flag set if the shifted destination operand has an even number of 1 bits	Parity flag set if the shifted destination operand has an even number of 1 bits
Carry	Carry flag set if the carry out of bit 0 is 1	Carry flag set if the carry out of bit 0 is 1

Here is an example of how to use the SHLD instruction:

```
mov eax, 0x12345678
mov ebx, 0xabcdef00
SHRD eax, ebx, 1

;After the SHRD instruction, eax will contain the value 0x092a3c40.
```

The SHRD instruction can be used to perform a variety of tasks, such as:

- Shifting a value to the right to divide it by a power of two.
- Shifting a value to the right to extract the least significant bits.
- Shifting a value to the right to prepare it for a bitwise operation.

The SHRD instruction is a logical instruction that is used to shift a destination operand to the right by a specified number of bits, and then fills the vacated bit positions with the least significant bits of the source operand. The source operand is not affected by the instruction.

The SHRD instruction can be used to perform a variety of tasks, including:

• **Dividing a value by a power of two:** The SHRD instruction can be used to divide a value by a power of two by shifting the value to the right by the number of bits equal to the power of two. For example, to divide a value by 2, the value would be shifted to the right by 1 bit. To divide a value by 4, the value would be shifted to the right by 2 bits, and so on.

• **Extracting the least significant bits of a value:** The SHRD instruction can be used to extract the least significant bits of a value by shifting the value to the right by a number of bits equal to the number of least significant bits that need to be extracted. For example, to extract the least significant 4 bits of a value, the value would be shifted to the right by 4 bits.

• **Preparing a value for a bitwise operation:** The SHRD instruction can be used to prepare a value for a bitwise operation by shifting the value to the right by a number of bits equal to the number of bits that need to be aligned to the right. For example, to align a value to the right by 4 bits, the value would be shifted to the right by 4 bits.

The following instruction formats apply to both SHLD and SHRD. The destination operand
can be a register or memory operand, and the source operand must be a register. The count operand can be the CL register or an 8-bit immediate operand:

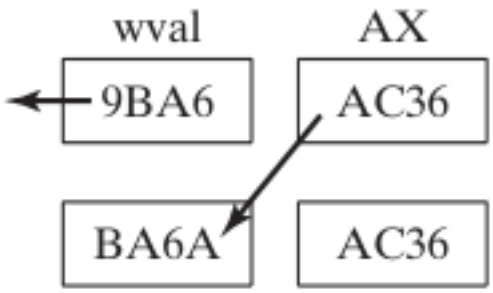
```
48 SHLD  reg16,reg16, CL/imm8
49 SHLD  mem16,reg16, CL/imm8
50 SHLD  reg32,reg32, CL/imm8
51 SHLD  mem32,reg32, CL/imm8
```

Example 1:

The following statements shift **wval** to the left 4 bits and insert the high 4 bits of AX into the low 4 bit positions of wval:

```
.data
    wval WORD 9BA6h
.code
    mov
    ax,0AC36h
    shld
    wval,ax,4        ;wval = BA6Ah
```

The data movement is shown in the following figure:

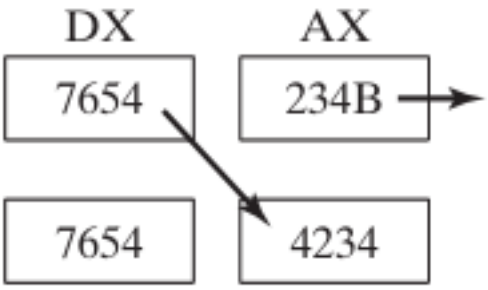


Example 2:

In the following example, AX is shifted to the right 4 bits, and the low 4 bits of DX are shifted into the high 4 positions of AX:

```
mov ax,234Bh
mov dx,7654h
shrd
ax,dx,4
```

The data movement is shown in the following figure:



Example 3:

The code below is an example of using the SHRD instruction to shift an array of doublewords to the right by 4 bits. This is a common operation in low-level programming when dealing with data manipulation, encryption, or even fast multiplication and division with very long integers. Let's break down the code and explain it step by step:

```
.data
    array DWORD 648B2165h,8C943A29h,6DFA4B86h,91F76C04h,8BAF9857h
.code
    mov bl,4
    ; shift count
    mov esi,OFFSET array
    ; offset of the array
    mov ecx,(LENGTHOF array) - 1
    ; number of array elements

L1:
    push ecx
    ; save loop counter
    mov eax,[esi + TYPE DWORD]
    mov cl,bl
    ; shift count
    shrd [esi],eax,cl
    ; shift EAX into high bits of [ESI]
    add esi,TYPE DWORD
    ; point to the next doubleword pair
    pop ecx
    ; restore loop counter
    loop L1
    shr DWORD PTR [esi],COUNT
    ; shift the last doubleword
```

In the .data section, you define an array of doublewords (DWORD). Each doubleword contains a hexadecimal value. This array represents the data that you want to shift.

In the .code section, you set up some initial values:

bl is set to 4, which represents the shift count (you want to shift the data by 4 bits).

esi is loaded with the offset of the array.

ecx is set to the number of elements in the array minus 1 (using LENGTHOF array - 1). This **loop counter** will be used to iterate through the array.

The core of the code is a loop labeled L1. Here's what happens inside the loop:

push ecx saves the loop counter on the stack.

mov eax, [esi + TYPE DWORD] loads the doubleword at the current esi offset into the eax register.

mov cl, bl loads the shift count into the cl register.

shrd [esi], eax, cl performs the shift operation, shifting the value in eax to the right by the number of bits specified in cl.

This result is stored back in the memory location pointed to by esi.

add esi, TYPE DWORD moves esi to the next doubleword in the array.

pop ecx restores the loop counter.

loop L1 decrements ecx (the loop counter) and jumps back to L1 as long as ecx is not zero. This loop processes each doubleword in the array.

Finally, after the loop, you perform a shift on the last doubleword using the shr instruction.

This code effectively shifts the entire array of doublewords to the right by 4 bits.

==========

Questions

==========

Which instruction shifts each bit in an operand to the left and copies the highest bit into both the Carry flag and the lowest bit position?

Answer: The instruction that performs this operation is SHL (Shift Left) or SAL (Shift Arithmetic Left).

Which instruction shifts each bit to the right, copies the lowest bit into the Carry flag, and copies the Carry flag into the highest bit position?

Answer: The instruction that shifts each bit to the right, copies the lowest bit into the Carry flag, and copies the Carry flag into the highest bit position is SHR (Shift Right).

Which instruction performs the following operation (CF = Carry flag)?

Answer: The operation you described is performed by the RCL (Rotate through Carry Left) instruction. It rotates the bits to the left through the Carry flag, as indicated.

What happens to the Carry flag when the SHR AX,1 instruction is executed?

Answer: When the SHR AX,1 instruction is executed, the Carry flag (CF) will receive the value of the least significant bit (LSB) of the AX register before the shift, and the LSB itself will be shifted out of the AX register.

Challenge: Write a series of instructions that shift the lowest bit of AX into the highest bit of BX without using the SHRD instruction. Next, perform the same operation using SHRD.

Answer: Below are the series of instructions to achieve this without SHRD:

```
100 mov cx, 1      ; Set the shift count to 1
101 shl bx, 1      ; Shift left the bits in BX by 1 position
102 rcl bx, 1      ; Rotate through Carry Left (this moves the original LSB of AX to the MSB of BX)
```

To perform the same operation using SHRD:

```
shrd bx, ax, 1  ; Shift right double (moves LSB of AX to the MSB of BX)
```

Challenge: One way to calculate the parity of a 32-bit number in EAX is to use a loop that shifts each bit into the Carry flag and accumulates a count of the number of times the Carry flag was set. Write a code that does this, and set the Parity flag accordingly.

Answer:

```
111 xor ecx, ecx      ; Clear the counter
112 mov ebx, eax      ; Make a copy of EAX
113 parity_loop:
114 shr ebx, 1        ; Shift right by 1 bit
115 adc ecx, 0        ; Add the Carry flag to the counter
116 jnz parity_loop   ; Jump back if there's still a 1 bit in EBX
117 test ecx, 1       ; Test the least significant bit of the counter
118 setp al           ; Set the Parity flag based on the counter
```

This code calculates the parity of a 32-bit number in EAX using a loop that shifts each bit into the Carry flag and accumulates a count of the number of times the Carry flag was set. It sets the Parity flag (PF) accordingly. If the count is even, PF will be set; if it's odd, PF will be cleared.