Bit Manipulation & Integer Arithmetic

Outline

- Shift and Rotate Instructions (7.1.1 to 7.1.8 KI, Chapter 4 BH)
- Applications of Shift and Rotate (7.2.1 and 7.2.2 KI Chapter 4 BH)
  - Multiplication

Shift and Rotate Instructions

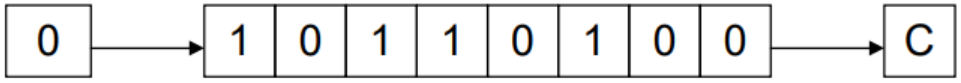
- Bit shifting means to move bits right and left inside an operand.
- Instructions are shown in table
- All these instructions effect the Overflow and Carry flags.
- Syntax
  - <shift operation > <destination>, <count>
  - Where count specified the number of shift/rotations
  - Destination can be reg or mem
  - Count can be immediate value or CL

Table 7-1 Shift and Rotate Instructions.

SHL	Shift left
SHR	Shift right
SAL	Shift arithmetic left
SAR	Shift arithmetic right
ROL	Rotate left
ROR	Rotate right
RCL	Rotate carry left
RCR	Rotate carry right

Shift Logical Right (SHR)

- Inserts a zero from the left and moves every bit one position to the right and copies the rightmost bit in the carry flag.




```
mov al, 10001111b
shr al, 1; CF=1 OF=1 answer was 01000111 1
```

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

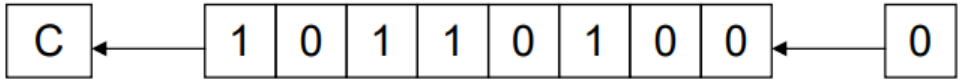
Before: 0 0 1 0 0 0 0 0 = 32

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

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

Shift Logical Left (SHL) / Shift Arithmetic Left (SAL)

- Zero bit is inserted from the right and every bit moves one position to its left with the most significant bit dropping into the carry flag




```
mov al, 10001111b
shl al, 1; CF=1 OF=1 because answer was 1 00011110
```

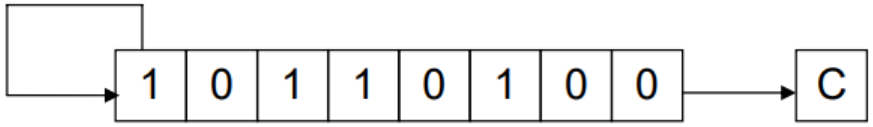
```
mov bl,8Fh                    ; BL = 10001111b
shl bl,1                      ; CF = 1, BL = 00011110b
```

```
mov dl,5        Before: 0 0 0 0 0 1 0 1  = 5
shl dl,1         After: 0 0 0 0 1 0 1 0  = 10
```

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

Shift Arithmetic Right (SAR)

- Use to handle the signed numbers to retain MSB i.e. sign bit
- Shifts every bit one place to the right with a copy of the most significant bit left at the most significant place.
- The bit dropped from the right is caught in the carry basket.

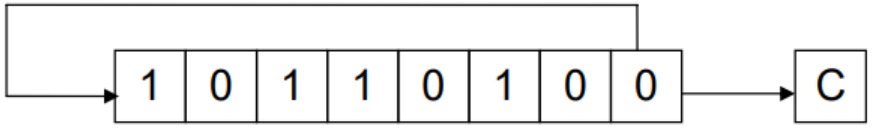



```
mov  dl,-128                ; DL = 10000000b
sar  dl,3                   ; DL = 11110000b
```

```
mov  al,0F0h                ; AL = 11110000b (-16)
sar  al,1                   ; AL = 11111000b (-8), CF = 0
```

Rotate Right (ROR)

- Every bit moves one position to the right and the bit dropped from the right is inserted at the left.
- This bit is also copied into the carry flag.

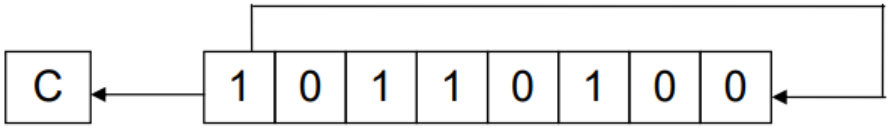



```
mov  al,01h                ; AL = 00000001b
ror  al,1                  ; AL = 10000000b, CF = 1
ror  al,1                  ; AL = 01000000b, CF = 0
```

```
mov  al,00000100b
ror  al,3                  ; AL = 10000000b, CF = 1
```

Rotate Left (ROL)

- Every bit moves one position to the left and the MSB dropped from the right is inserted at the right.
- This bit is also copied into the carry flag.

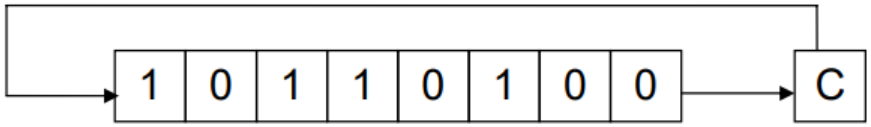



```
mov  al,40h                ; AL = 01000000b
rol  al,1                  ; AL = 10000000b, CF = 0
rol  al,1                  ; AL = 00000001b, CF = 1
rol  al,1                  ; AL = 00000010b, CF = 0
```

```
mov  al,00100000b
rol  al,3                   ; CF = 1, AL = 00000001b
```

Rotate Through Carry Right (RCR)

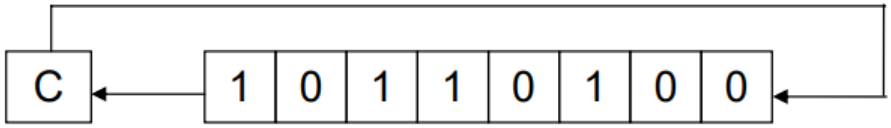
- The carry flag is inserted from the left
- Every bit moves one position to the right.
- The right most bit is dropped in the carry flag.




```
stc                     ; CF = 1
mov  ah,10h             ; AH, CF = 00010000 1
rcr  ah,1               ; AH, CF = 10001000 0
```

Rotate Through Carry Left (RCL)

- The carry flag is inserted from the right
- Every bit moves one position to the left.
- The left most bit is dropped in the carry flag.




```
clc                     ; CF = 0
mov  bl,88h             ; CF,BL = 0 10001000b
rcl  bl,1               ; CF,BL = 1 00010000b
rcl  bl,1               ; CF,BL = 0 00100001b
```

Signed Overflow

- The Overflow flag is set if the act of shifting or rotating a signed integer by **one bit position** generates a value outside the signed integer range of the destination operand.
- To put it another way, the number's sign is reversed.
- Examples
  - a positive integer (+127) stored in an 8-bit register becomes negative (-2) when rotated left:

```
mov  al,+127                ; AL = 01111111b
rol  al,1                   ; OF = 1, AL = 11111110b
```

  - When –128 is shifted one position to the right, the Overflow flag is set. The result in AL (+64) has the opposite sign

```
mov  al,-128                ; AL = 10000000b
shr  al,1                   ; OF = 1, AL = 01000000b
```

- The value of the Overflow flag is undefined when the shift or rotation count is greater than 1

Applications of Bit Manipulation

Leo Villareal's 23,000 Points of Light

- The Renwick Gallery, Washington D.C.

Villareal and his team have developed custom tools and software over the past decade, which ensures that the sequences never repeat, keeping things from getting predictable, even for the artist himself.

https://youtu.be/cfj_E8r-vCk

https://youtu.be/kGxzgO391_Y

https://www.smithsonianmag.com/smithsonian-institution/most-majestic-energy-saving-sculpture-ever-seen-180957105/

Multiplication In Assembly Language

- 4 bit multiplication
  - Shift the multiplier to the right.
  - If CF=1
    - add the multiplicand to the result.
  - Shift the multiplicand to the left.
  - Repeat the algorithm 4 times.
- For 4 bit multiplication, multiplicand and result should be 8 bit.
- Similarly, for n bit multiplication, multiplicand and result should be 2*n bits.

```
  1101 = 13        Accumulated Result
  0101 = 5
  -----                          0 (Initial Value)
  1101 = 13        0 + 13  = 13
 0000x = 0         13 + 0  = 13
1101xx = 52        13 + 52 = 65
0000xxx = 0        65 + 0  = 65 (Answer)
```

```
          01111011      123
    ×     00100100      36
    --------------
          01111011      123 SHL 2
+   01111011            123 SHL 5
    --------------
  0001000101001100      4428
```

Multiplication In Assembly Language

Example 4.1

```
01  ; 4bit multiplication algorithm
02  [org 0x100]
03                  jmp  start
04
05  multiplicand:   db   13                 ; 4bit multiplicand (8bit space)
06  multiplier:     db   5                  ; 4bit multiplier
07  result:         db   0                  ; 8bit result
08
09  start:          mov  cl, 4              ; initialize bit count to four
10                  mov  bl, [multiplicand] ; load multiplicand in bl
11                  mov  dl, [multiplier]   ; load multiplier in dl
12
13  checkbit:       shr  dl, 1              ; move right most bit in carry
14                  jnc  skip               ; skip addition if bit is zero
15
16                  add  [result], bl       ; accumulate result
17
18  skip:           shl  bl, 1              ; shift multiplicand left
19                  dec  cl                 ; decrement bit count
20                  jnz  checkbit           ; repeat if bits left
21
22                  mov  ax, 0x4c00         ; terminate program
23                  int  0x21
```

Question

- Modify the algorithm given in last slide for 8 bit multiplication.
- How will you modify the algorithm given in last slide for 16 or 32 bit multiplication?

Extended Operations

- Working with larger numbers
- For example
  - Adding/Subtracting 32 bit or 64 bit numbers
  - Multiplying 16 or 32 or 64 bit numbers
  - Shifting 32 bit number

Extended Multiplication

- Take an example of multiplying two 16 bit numbers.
- The result will need 32 bits and multiplicand will also need 32 bits.
- This will require SHL 32 bit multiplicand and addition of 32 bit multiplicand and 32 bit result.
- So to perform extended multiplication we should first look at extended addition and extended shifting.
  - *Reminder: one instruction of ADD or SHR only works for 8 or 16 bits*

Extended Shifting Left

- Consider a 32 bit number num1

```
Num1: dd  0000 0000 0000 0000 1111 0000 1111 0000b
```

  - Note spaces are just there for readability.

- How will you Shift it left by 1 such that the result is

```
Num1: dd  0000 0000 0000 0001 1110 0001 1110 0000b
```

Extended Shifting Left

- You will use two instructions

```
Shl word [num1], 1
Rcl word [num1+2], 1
```

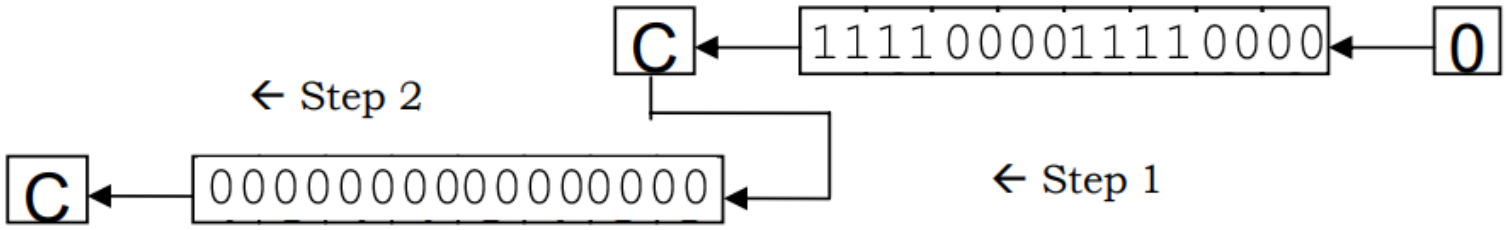
- Effect of 1st instruction:

```
Num1: dd  0000 0000 0000 0000 1110 0001 1110 0000 b
CF=1
```

- Effect of 2nd Instruction

```
Num1: dd  0000 0000 0000 0001 1110 0001 1110 0000 b
CF=0
```

Extended Shifting left



Extended Shifting Right

- Consider a 32 it number num1

```
Num1: dd  0000 0000 0000 0001 0000 0000 0000 0000b
```

  - Note spaces are just there for readability.

- How will you Shift it right by 1 such that the result is

```
Num1: dd  0000 0000 0000 0000 1000 000 0000 0000b
```

Extended Shifting Right

- You will use two instructions

```
Shr word [num1+2], 1
Rcr word [num1], 1
```

- Effect of 1st instruction:

Num1: dd **0000 0000 0000 0000** 0000 0000 0000 0000b

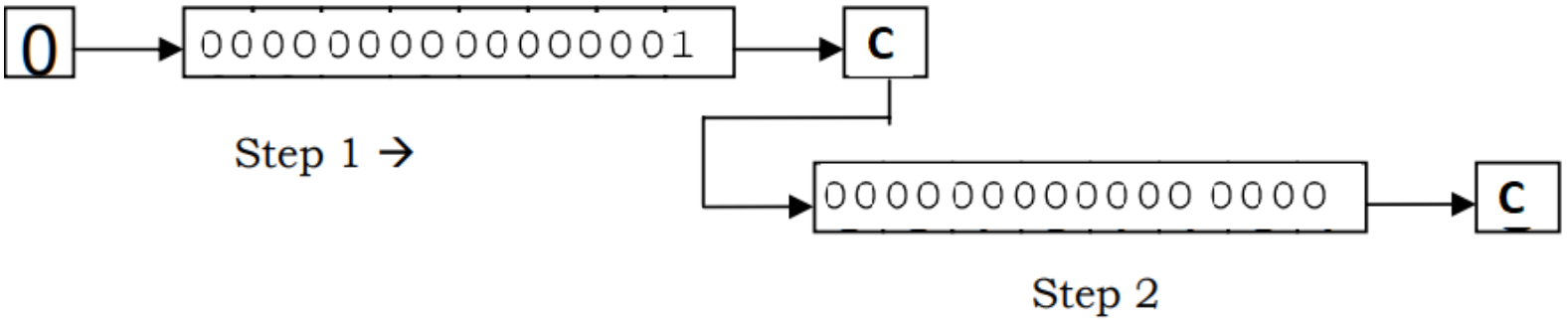
CF=1

- Effect of 2nd Instruction

Num1: dd 0000 0000 0000 0000 **1000 0000 0000 0000** b

CF=0

Extended Shifting Right



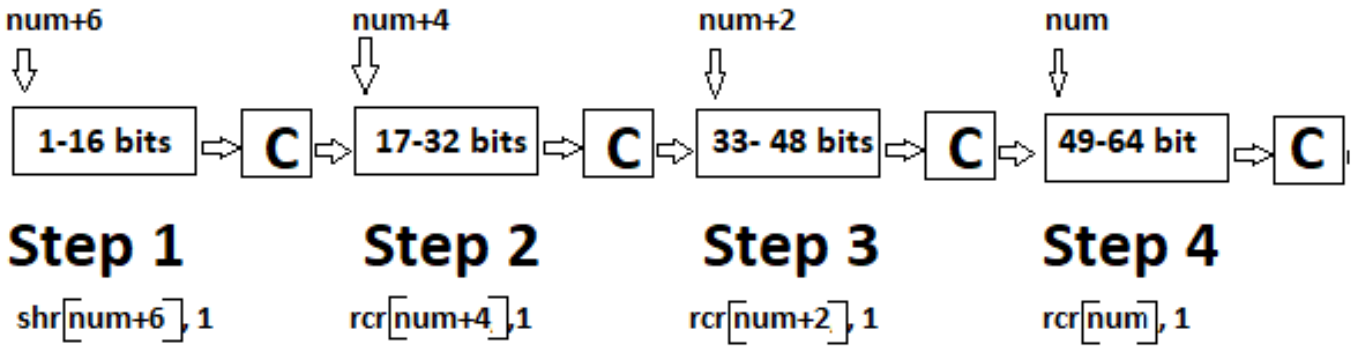
Extended Shift

- How will you shift right a 64 bit number?

Extended Shift

- How will you shift right a 64 bit number?

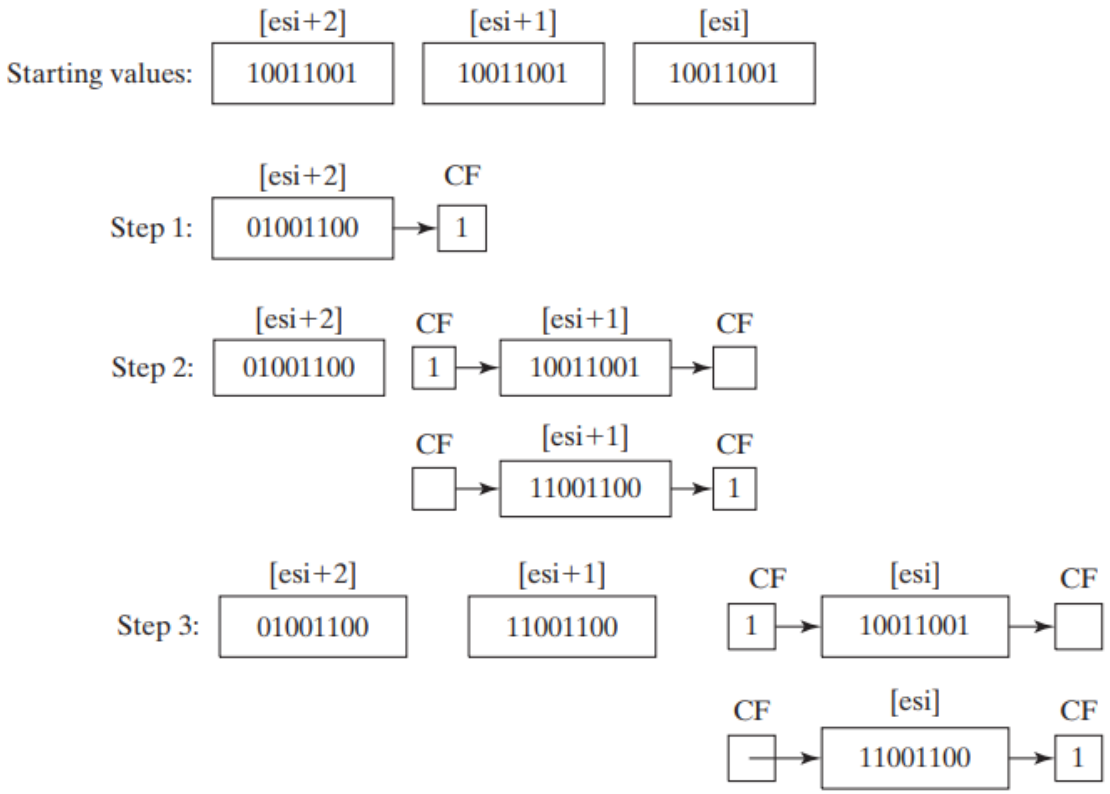
- Answer



Extended Shift

- How will you shift left a 64 bit number?

Another Example



After Step 3 is complete, all bits have been shifted 1 position to the right: