University of Central Punjab
(Incorporated by Ordinance No. XXIV of 2002 promulgated by Government of the Punjab)
FACULTY OF INFORMATION TECHNOLOGY

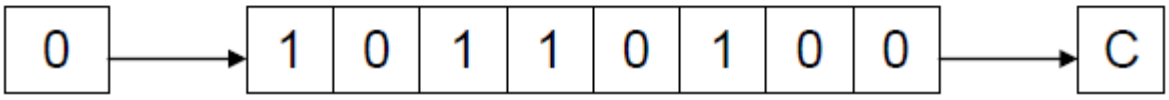
Computer Organization and Assembly Language

Lab 06	
Topic	▪ Arithmetic & Logical instructions ▪ Selective bit setting/clearing/complimenting ▪ Shifting and Rotations variations ▪ Extended addition and subtraction

Part 1

Shift Logical Right (SHR)

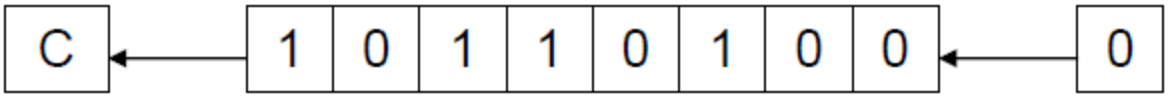
The shift logical right operation inserts a zero from the left and moves every bit one position to the right and copies the rightmost bit in the carry flag. Imagine that there is a pipe filled to capacity with eight balls. The pipe is open from both ends and there is a basket at the right end to hold anything dropping from there. The operation of shift logical right is to force a white ball from the left end. The operation is depicted in the following illustration.



White balls represent zero bits while black balls represent one bits. Sixteen bit shifting is done the same way with a pipe of double capacity.

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

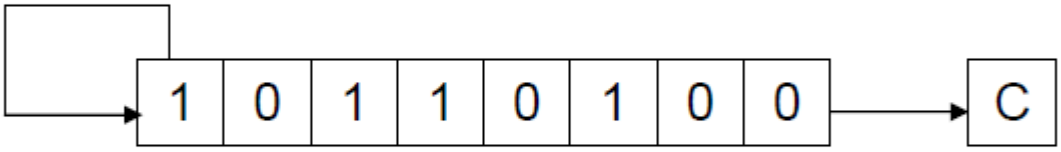
The shift logical left operation is the exact opposite of shift logical right. In this operation the zero bit is inserted from the right and every bit moves one position to its left with the most significant bit dropping into the carry flag. Shift arithmetic left is just another name for shift logical left. The operation is again exemplified with the following illustration of ball and pipes.



Shift Arithmetic Right (SAR)

A signed number holds the sign in its most significant bit. If this bit was one a logical right shifting will change the sign of this number because of insertion of a zero from the left. The sign of a signed number should not change because of shifting.

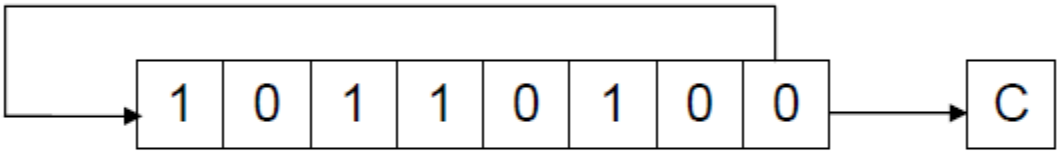
The operation of shift arithmetic right is therefore to shift every bit one place to the right with a copy of the most significant bit left at the most significant place. The bit dropped from the right is caught in the carry basket. The sign bit is retained in this operation. The operation is further illustrated below.



The left shifting operation is basically multiplication by 2 while the right shifting operation is division by two. However for signed numbers division by two can be accomplished by using shift arithmetic right and not shift logical right. The left shift operation is equivalent to multiplication except when an important bit is dropped from the left. The overflow flag will signal this condition if it occurs and can be checked with JO. For division by 2 of a signed number logical right shifting will give a wrong answer for a negative number as the zero inserted from the left will change its sign. To retain the sign flag and still effectively divide by two the shift arithmetic right instruction must be used on signed numbers.

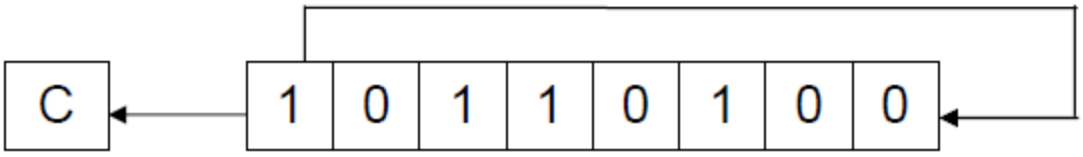
Rotate Right (ROR)

In the rotate right operation 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. The operation can be understood by imagining that the pipe used for shifting has been molded such that both ends coincide. Now when the first ball is forced to move forward, every ball moves one step forward with the last ball entering the pipe from its other end occupying the first ball's old position. The carry basket takes a snapshot of this ball leaving one end of the pipe and entering from the other.



Rotate Left (ROL)

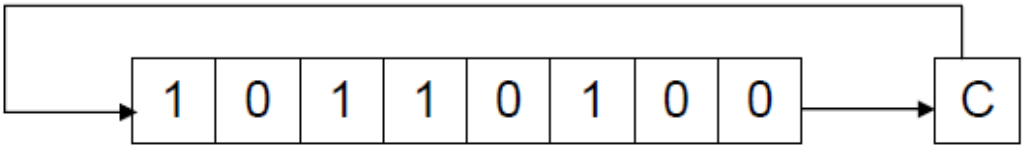
In the operation of rotate left instruction, the most significant bit is copied to the carry flag and is inserted from the right, causing every bit to move one position to the left. It is the reverse of the rotate right instruction. Rotation can be of eight or sixteen bits. The following illustration will make the concept clear using the same pipe and balls example.



Rotate Through Carry Right (RCR)

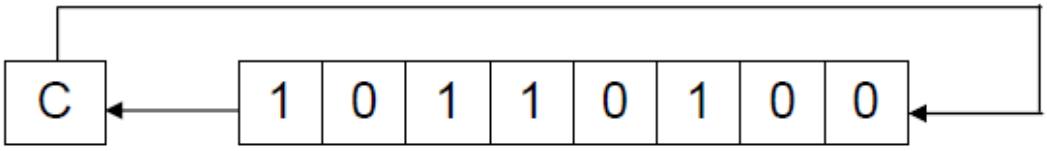
In the rotate through carry right instruction, the carry flag is inserted from the left, every bit moves one position to the right, and the right most bit is dropped in the carry flag. Effectively this is a nine bit or a seventeen bit rotation instead of the eight or sixteen bit rotation as in the case of simple rotations.

Imagine the circular molded pipe as used in the simple rotations but this time the carry position is part of the circle between the two ends of the pipe. Pushing the carry ball from the left causes every ball to move one step to its right and the right most bit occupying the carry place. The idea is further illustrated below.



Rotate Through Carry Left (RCL)

The exact opposite of rotate through carry right instruction is the rotate through carry left instruction. In its operation the carry flag is inserted from the right causing every bit to move one location to its left and the most significant bit occupying the carry flag. The concept is illustrated below in the same manner as in the last example.



Example 1:

Multiply the number by 4 using shift operator.

Let the number is 5.

```
mov al,5
shl al,2
```

Example 2:

Rotate right 3 times the value in register bx.

Let BX=0xEFCD

```
mov bx,0xEFCD
ror bx,3
```

Example 3(Extended addition)

```
MOV AX, [Num1] ;loads two bytes into AX register, AX=FFFF
MOV BX, [Num1+2] ;loads Next two bytes into BX register, BX=0001
ADD AX, [Num2] ; adds into AX; AX=AX+0002;
ADC BX, [Num2+2]; Add with carry instruction
MOV [SUM],AX ; Move the lower bits into Sum variable
MOV [SUM+2],BX  ; Move the higher bits into Sum variable higher bits
mov ax,0x4c00
int 21h
Num1: dd 0x0001FFFF
Num2: dd 0x00010002
SUM: dd 0
```

Example 4(Extended subtraction)

```
MOV AX, [Num2] ;loads two bytes into AX register, AX=0002

MOV BX, [Num2+2] ;loads Next two bytes into AX register, AX=0001

SUB AX, [Num1] ; sub into AX; AX=AX-FFFF;

SBB BX, [Num1+2]; Subtraction with borrow.

MOV [ans],AX ; Move the lower bits into ans variable

MOV [ans+2],BX  ; Move the higher bits into ans variable higher bits

mov ax,0x4c00
int 21h

Num1: dd 0x0001FFFF

 Num2: dd 0x00010002

 ans: dd 0
```

```
a = 10, b = 20, c = 5 , sum=0;

        if (a < =b)
        {
          // L1
                if (a < =c)
                {
                   // L2
                        if (b>c)
                        {
                           // L3
                                sum = a + b + c;
                        }
                        else
                        {
                           // L4
                                sum = a - b - c;
                        }
                }
                else
                {
                   // L5
                        sum = a + b - c;
                }
        }
        else
        {
          // L6
                sum = a - b + c;
        }
```

```
[org 0x100]
mov al,[a]
mov bl,[b]
mov cl,[c]

cmp al,bl
jng l1
jg  l6
l1:
   cmp al,cl
   jng l2
   jg  l5
l2:
   cmp bl,cl
   jg  l3
   jng l4
l3:
   mov [sum],al
   add [sum],bl
   add [sum],cl
   jmp exit
l4:
   mov [sum],al
   sub [sum],bl
   sub [sum],cl
   jmp exit
l5:
   mov [sum],al
   add [sum],bl
   sub [sum],cl
   jmp exit
l6:
   mov [sum],al
   sub [sum],bl
   add [sum],cl
exit:
mov ax,0x4c00
int 21h

a: db 10
b: db 20
c: db 5
sum: db 0
```