Name:

Dawood Safraz

Roll no:

20p-0153

Section:

3B

Batch:

Fall-2020

COAL (Lab) Report:

 # 06

Submitted to:

Dr. Usman Abbasi

FAST NUCES Peshawar Camp.

SHIFTING AND ROTATIONS:

a) ***Shift Logical Right (SHR)***
b) ***Shift Logical Left (SHL) / Shift Arithmetic Left (SAL)***
c) ***Shift Arithmetic Right (SAR)***
d) ***Rotate Right (ROR)***
e) ***Rotate Left (ROL)***
f) ***Rotate Through Carry Right (RCR)***
g) ***Rotate Through Carry Left (RCL)***

Details of Shifting & Rotations:

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.

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.

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

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 bits rotation instead of the eight or sixteen bits 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.

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.

Code for Multiplication:

```
[org 0x0100]
jmp start

multiplicand: db 1      ;we 4-bit number, save space of 8-bits
multiplier:   db 5       ; 4-bit number

result:       db 0       ; 8-bit result
```

```
start:


    mov  cl, 4            ; number of iterations to run the loop
    mov  bl, [multiplicand] ; mov value to register
    mov  dl, [multiplier] ; mov value to register


    checkbit:
        shr  dl, 1        ; do the rotation so that right bit is
thrown in Carry Flag (CF)
        jnc  skip
            add [result], bl     ; only add if Carry Flag (CF)
is SET


        skip:
        shl  bl, 1         ; always shift the multiplicand


    dec  cl
    jnz  checkbit


    mov  ax, 0x4c00
    int  0x21
```

List File:

```
1                       [org 0x0100]
2
3 00000000 E90300               jmp start
4
5 00000003 0D                   multiplicand: db 13     ;we 4-bit number, save space of 8-bits
6 00000004 05                   multiplier:   db 5      ; 4-bit number
7
8 00000005 00                   result:       db 0      ; 8-bit result 13 = 1101 , 05 = 0101 ,1101 x
0101 =
9
10                      start:
11
12 00000006 B104                    mov  cl, 4          ; how many times we need to run the loop
13 00000008 8A1E[0300]                mov  bl, [multiplicand]
14 0000000C 8A16[0400]                mov  dl, [multiplier]
15
16
17                          checkbit:
18 00000010 D0EA                     shr  dl, 1   ; do the rotation so that right bit is thrown in CF
19 00000012 7304                    jnc  skip
20 00000014 001E[0500]                    add [result], bl     ; only add if CF IS SET
21
22
23                              skip:
24 00000018 D0E3                     shl  bl, 1        ; always shift the multiplicand
25
26 0000001A FEC9                    dec  cl
```

27 0000001C 75F2 jnz checkbit

28

29

30 0000001E B8004C mov ax, 0x4c00

31 00000021 CD21 int 0x21

```
[org 0x0100]
jmp start

multiplicand: db 1      ;we 4-bit number, save space of 8-bits
multiplier:   db 5       ; 4-bit number

result:       db 0       ; 8-bit result

start:

    mov  cl, 4            ; number of iterations to run the loop
    mov  bl, [multiplicand] ; mov value to register
    mov  dl, [multiplier] ; mov value to register

    checkbit:
        shr  dl, 1        ; do the rotation so that right bit is thrown in Carry Flag (CF)
        jnc  skip
            add [result], bl     ; only add if Carry Flag (CF) is SET

        skip:
        shl  bl, 1         ; always shift the multiplicand

    dec  cl
    jnz  checkbit

    mov  ax, 0x4c00
    int  0x21
```

```
 1                                  [org 0x0100]
 2
 3 00000000 E90300                  jmp start
 4
 5 00000003 0D                      multiplicand: db 13      ;we 4-bit number, save space of 8-bits
 6 00000004 05                      multiplier:   db 5       ; 4-bit number
 7
 8 00000005 00                      result:       db 0       ; 8-bit result 13 = 1101 , 05 = 0101 ,1101 x 0101 =
 9
10                                  start:
11
12 00000006 B104                        mov  cl, 4            ; how many times we need to run the loop
13 00000008 8A1E[0300]                  mov  bl, [multiplicand]
14 0000000C 8A16[0400]                  mov  dl, [multiplier]
15
16
17                                      checkbit:
18 00000010 D0EA                            shr  dl, 1        ; do the rotation so that right bit is thrown in CF
19 00000012 7304                            jnc  skip
20 00000014 001E[0500]                          add [result], bl     ; only add if CF IS SET
21
22
23                                          skip:
24 00000018 D0E3                            shl  bl, 1         ; always shift the multiplicand
25
26 0000001A FEC9                        dec  cl
27 0000001C 75F2                        jnz  checkbit
28
29
30 0000001E B8004C                      mov  ax, 0x4c00
31 00000021 CD21                        int  0x21
```

DONE