EC-310 Microprocessor & Microcontroller based Design

Dr. Ahsan Shahzad

Department of Computer and Software Engineering (DC&SE)

College of Electrical and Mechanical Engineering (CEME)

National University of Science and Technology (NUST)

Week # 3

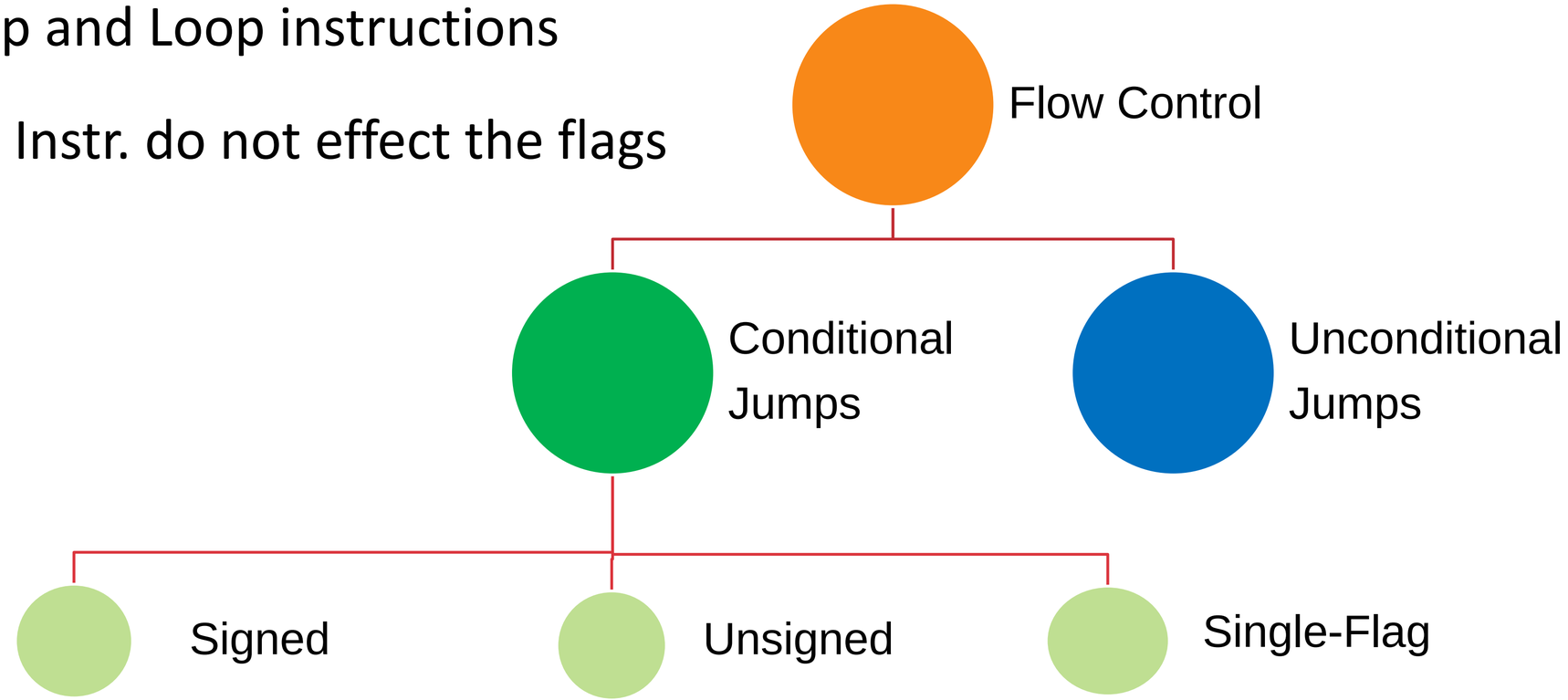
Flow Control Instructions

Unconditional Jump

JMP Destination

Destination label should be in the same segment

- To make Decisions and to Repeat sections of code
- Examples: Jump and Loop instructions
- Jump and loop Instr. do not effect the flags



CMP destination, Source

Similar to SUB instr. except destination will not change

Conditional Jumps

- **Range of conditional jumps:** destination label must precedes jump instruction by 126 bytes or follow it by 127 bytes

Signed Jumps

Symbol	Description	Condition for Jump
JG / JNLE	Jump if greater / Jump If not less than or equal to	ZF = 0 **AND** SF = OF
JGE / JNL	Jump if greater than or equal to / Jump if not less than	SF = OF
JL / JNGE	Jump if less than / Jump if not greater than or equal to	SF <> OF
JLE / JNG	Jump if less than or equal / Jump if not greater	ZF =1 **OR** SF <> OF

Conditional Jumps

Unsigned Jumps

- CMP AX, BX
- JA Label

CMP AX,BX

JG Label

Where, AX = 7FFFh, BX = 8000h

Symbol	Description	Condition for Jump
JA / JNBE	Jump if above / Jump If not below or equal to	CF = 0 **AND** ZF = 0
JAE / JNB	Jump if above or equal to / Jump if not below	CF = 0
JB / JNAE	Jump if below / Jump if not above or equal to	CF = 1
JBE / JNA	Jump if below or equal / Jump if not above	CF = 1 **OR** ZF = 1

Conditional Jumps

Single-Flag Jumps

- Jumps based on individual flags

Symbol	Description	Condition for Jump
JE / JZ	Jump if equal / equal to zero	ZF = 1
JNE / JNZ	Jump if not equal / not zero	ZF = 0
JC (JNC)	Jump if carry (Jump if no carry)	CF = 1 (CF = 0)
JO (JNO)	Jump if overflow (Jump if no overflow)	OF = 1 (OF = 0)
JS (JNS)	Jump if sign negative (if sign +ve)	SF =1 (SF = 0)
JP / JPE	Jump if parity even	PF = 1
JNP / JPO	Jump if parity odd	PF = 0

Extending Range of Conditional Jumps

Using JMP instruction

- Destination label can be anywhere in the same segment INSTEAD of 126 bytes before or 127 bytes after the conditional jump instructions

```
TOP:

; body of the loop:
          DEC CX
          JNZ TOP

EXIT:
          MOV AH,4CH
          INT 21H
```

```
TOP:

; body of the loop:
          DEC CX
          JNZ BOTTOM
          JMP EXIT

BOTTOM:
          JMP TOP
EXIT:
          MOV AH,4CH
          INT 21H
```

Branching Structures - Examples

IF-THEN-ELSE

- **EXAMPLE:** Suppose AL, BL contain ASCII characters. Display the one that comes first in the character sequence

```
        MOV AH, 02 ; prepare to display

        CMP AL, BL ; IF AL <= BL
        JNBE ELSE_
;then
        MOV DL, AL
        JMP DISPLAY
ELSE_:
        MOV DL,BL
DISPLAY:
        INT21H
;END_IF
```

Branching Structures - Examples

CASE (Multiple branch structure)

- **EXAMPLE:** if AX contains a negative number, put -1 in BX; if AX contains 0, put 0 in BX; if AX contains a positive number, put 1 in BX

```
; Case AX
        CMP AX, 0 ; test AX
        JL NEGATIVE ; AX < 0
        JE ZERO     ; AX = 0
        JG POSITIVE ; AX > 0

NEGATIVE:
        MOV BX,-1
        JMP END_CASE ; and exit
ZERO:
        MOV BX,0
        JMP END_CASE ; and exit
POSITIVE:
        MOV BX,1

END_CASE:
```

Looping Structures

FOR LOOP

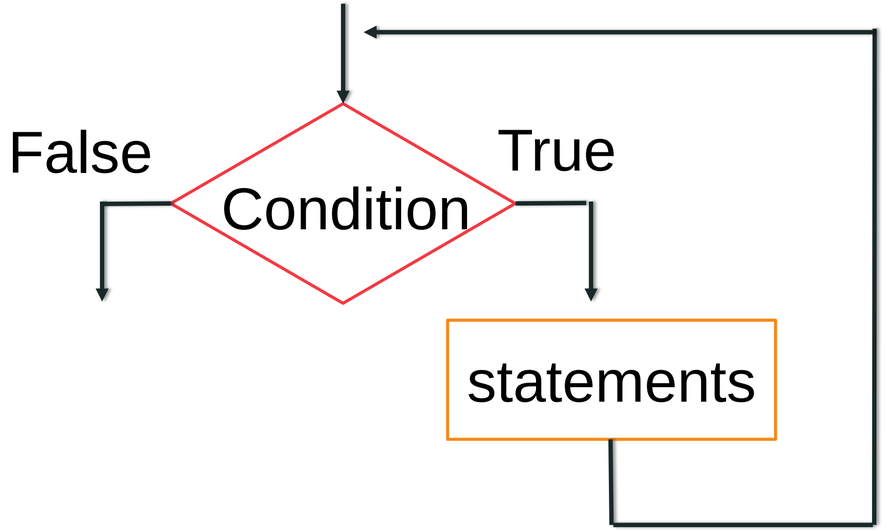
- LOOP Destination_Label
- **CX register** holds the counter value
- LOOP instr. **Auto decrements** the CX register
- Dest._Label must precedes the LOOP instr. by no more than **126 bytes.**
- FOR loop using LOOP instruction is executed at least once.
- If CX initialized with 0, What Happens??

```
; initialize CX to loop_count

TOP:
        ; body of the loop
        LOOP TOP
```

```
; initialize CX to loop_count
        JCXZ SKIP
TOP:
        ; body of the loop
        LOOP TOP
SKIP:
```

Looping Structures



WHILE LOOP

WHILE condition do

statements

END_WHILE

- **Example:** Count the no. of characters in an input line

```
        MOV DX, 0
        MOV AH, 1
        INT 21H
WHILE_:
        CMP AL, 0DH     ; CR?
        JE END_WHILE
        INC DX          ; not CR, increment count
        INT 21H         ; read a character
        JMP WHILE_      ; LOOP BACK
END_WHILE:
```

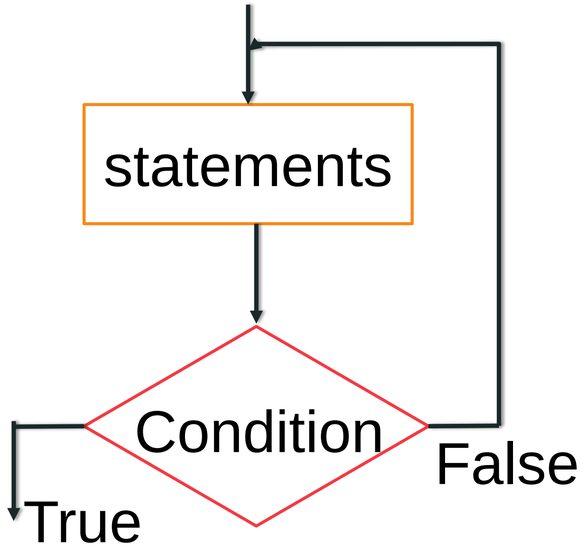
Looping Structures

REPEAT LOOP

REPEAT

statements

UNTIL condition



- **Example:** to read characters until a blank is read

```
        MOV AH, 1

REPEAT:
        INT 21H         ; read a character
;until
        CMP AL, ' '       ; a blank?
        JNE REPEAT        ;if no, keep reading
```

- WHILE Vs. REPEAT LOOP
  - Condition check in the start Vs. at the end (runs at least once)
  - Long code (two jumps) Vs. short code (one jump)

Conditional Count-Controlled Loop

Other Loop Instructions (CX auto decrement)

- **LOOPE label** ; loop while equal (ZF = 1)
- **LOOPZ label** ; loop while zero (ZF = 1)

  ➢ CX = CX-1

  ➢ If ZF = 1 **AND** CX <> 0, then jump to label

- **LOOPNE label** ; loop while not equal (ZF = 0)
- **LOOPNZ label** ;loop while not zero (ZF = 0)

  ➢ CX = CX-1

  ➢ If ZF = 0 **AND** CX <> 0, then jump to label

- **Example:** to read characters until a blank is typed OR 80 characters have been typed
  - How to modify this code??

```
        MOV AH, 1

REPEAT:
        INT 21H         ; read a character
;until
        CMP AL, ' '       ; a blank?
        JNE REPEAT        ;if no, keep reading
```

Logic, Shift and Rotate Instructions

- Used to change the bit pattern of a memory location
- Ability to manipulate bits is generally absent in high level languages (except C)

LOGIC Instructions

- **AND, OR, XOR** Instructions -> Format: **AND destination, source**
- **Effect on Flags:**
  - SF, ZF, PF reflects the result
  - AF is undefined
  - CF, OF = 0
- MASK ?
- Conclusion: To Set, Clear, Complement bits?

b AND 1 = b	b OR 0 = b	b XOR 0 = b
b AND 0 = 0	b OR 1 = 1	b XOR 1 = ~b (Complement)

Logic Instructions (ctd.)

LOGIC Instructions

- **Clear** bit: **AND** with 0 mask bit (1 to preserve)
- **Set** bit: **OR** with 1 mask bit (0 to preserve)
- **Toggle** or Complement: **XOR** with 1 (0 to preserve)

- **Example:** Clear only sign bit of AL Register ?
- **Example:** Change the sign bit of DX Register ?

- **Solution:** AND AL, 7Fh
- **Solution:** XOR DX, 8000h

b AND 1 = b	b OR 0 = b	b XOR 0 = b
b AND 0 = 0	b OR 1 = 1	b XOR 1 = ~b (Complement)

Logic Instructions (ctd.) - Examples

- **Converting an ASCII Digit to a Number**
  - SUB AL, 30h **OR** **AND AL, 0Fh**
- **Clearing a Register**
  - MOV AX, 0 **OR** **SUB AX,AX** **OR** **XOR AX,AX**
- **Testing a Register for Zero**
  - CMP CX,0 **OR** **OR CX,CX**

Logic Instructions (ctd.)

NOT Instruction

- Invert the bits (1's complement) Format: **NOT AX**
- No effect on status flags

TEST Instruction

- Similar to **AND** instr. except the result will not be updated in destination
- **TEST destination, source**
- **Effect on Flags:**
  - **SF, ZF, PF** reflects the result
  - AF is undefined while CF, OF = 0
- **Can be used to examine individual bits**

Example: Jump if AL has even number?

HINT: Even nos. have 0 at bit 0

Solution: TEST AL, 1

JZ label

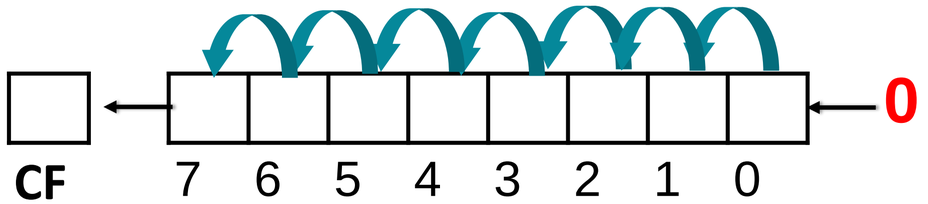
Shift Instructions (SHL, SHR, SAL, SAR)

- Bits in the destination operand are shifted either left or right direction (Bits lost)
- Two Possible formats:
  - Opcode destination,**1** Opcode destination, **CL** (CL contains N shifts)

Shift Left Instruction (SHL)

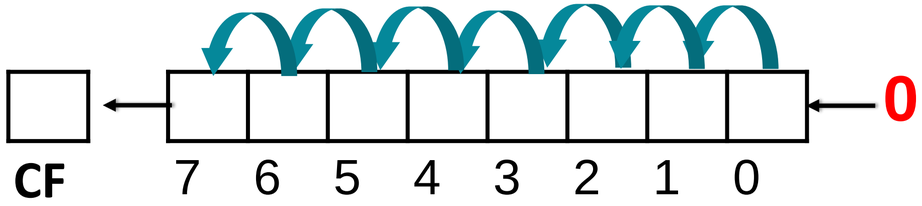
- Shift operand1 Left. The number of shifts is set by operand2.
- Shift all bits left, the bit that goes off is set to CF.
- Zero bit is inserted to the right-most position.
- . **Effect on Flags:**
  - SF, ZF, PF reflects the result (AF undefined)
  - CF = last bit shifted out and OF = 1 if results **changes sign** on **last shift**

Example:
MOV AL, 11100000b
SHL AL, 1 ; AL = 11000000b, CF=1



Shift Instructions (SHL, SHR, SAL, SAR)

Shift Arithmetic Left Instruction (SAL)



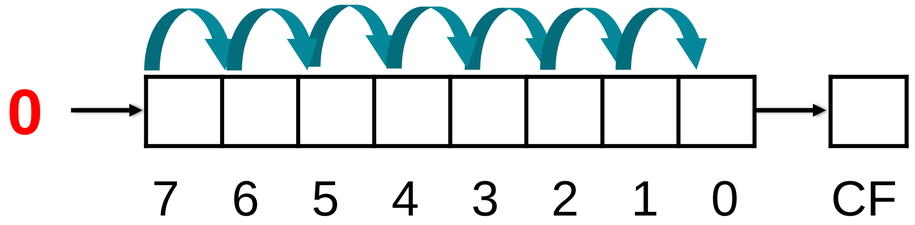
- Exactly same as SHL even machine code.
- Left shift on a binary number -> multiplies it by 2
- To emphasize this multiplication operation, SAL is used when multiplication by multiples of 2 is desired
- Negative numbers can also be multiplied by powers of 2.
- **Effect on Flags:**
  - SF, ZF, PF reflects the result (AF undefined)
  - Overflow possible due to multiplication operation
  - CF and OF accurately indicate Unsigned and Signed overflow respectively for a single shift only.

Example: xply by 4
MOV AL, 1000 0000b ; 80h
MOV CL, 2
SAL AL, CL ; AL = 00000000b,
CF = OF = 0 (Wrong)
Both Overflows exists

Shift Instructions (SHL, SHR, SAL, SAR)

Shift Right Instruction (SHR)

- Shift operand1 right. The number of shifts is set by operand2.
- Shift all bits right, the bit that goes off is set to CF.
- Zero bit is inserted to the left-most position.
- **Effect on Flags:**
  - SF, ZF, PF reflects the result (AF undefined)
  - CF = last bit shifted out and OF = 1 if results **changes sign** on **last shift**



➢ Same as SHL instruction only direction of shift is changed

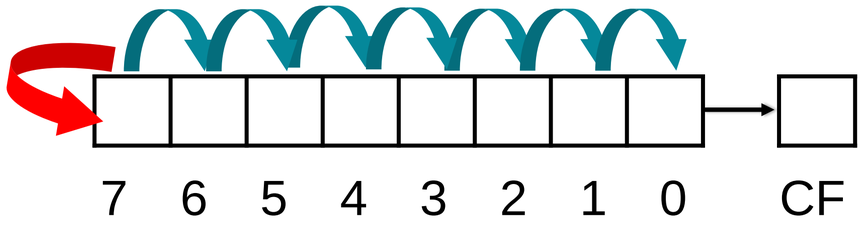
Shift Instructions (SHL, SHR, SAL, SAR)

Shift Arithmetic Right Instruction (SAR)

- Main **difference with SHR**: **MSB bit re-inserted** and hence **sign preserves**
- Division by right shift. (Right shift halves the number and rounds down)
- SHR for unsigned and SAR for signed representation

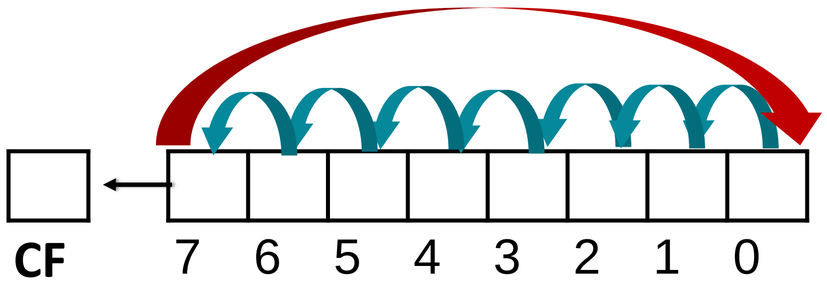
Example:
MOV AL, 0000 0101b ;5
SAR AL, 1 ; AL = 0000 0010b, 2

Example:
MOV AL, 1111 0001b ;-15
SAR AL, 1 ; AL = 1111 1000b, -8

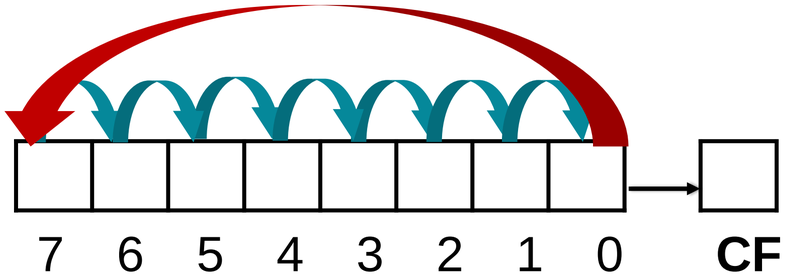


Rotate Instructions (ROL, ROR, RCL, RCR)

- Bits rotated either left or right direction **(Bits will not lost)**
- Two Possible formats:
  - Opcode destination, **1** Opcode destination, **CL** (CL contains N rotations)
- Effects the flags in a similar manner as shift instructions
- In ROL and ROR, CF reflects the bit that is rotated out



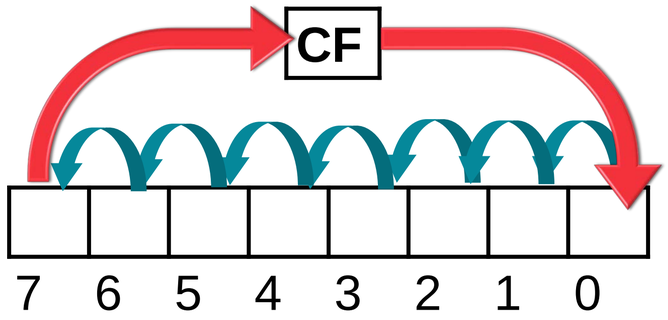
Rotate Left Instruction (ROL)



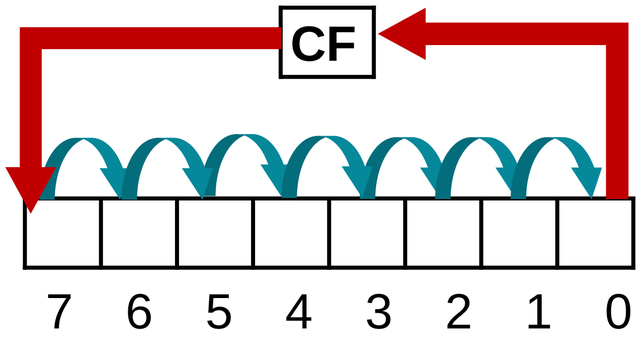
Rotate Right Instruction (ROR)

Rotate Instructions (ROL, ROR, RCL, RCR)

- **RCL:** MSB shifted into CF and previous value of CF is inserted into the right most bit.
- **RCR:** LSB shifted into CF and previous value of CF is inserted into the left most bit.
- **Applications of Shift & Rotate Instr:**
  - Reversing a bit pattern
  - Binary and Hex I/O



Rotate Carry Left (RCL)



Rotate Carry Right (RCR)