Seminar 3

Rechnerarchitektur

Bitverarbeitung

- Bitweise logische Befehle:
  - AND
  - TEST
  - OR
  - XOR
  - NOT

AND – das logische UND

AND	0	1
0	**0**	**0**
1	**0**	**1**

- Syntax: *AND op1, op2*
- verknüpft zwei Operanden bitweise entsprechend dem logischen UND
- Die Operanden können 8, 16 oder 32 Bit haben und Register-, Speicher- oder Direktoperanden sein. Da das Ergebnis im ersten Operanden abgelegt wird kann dieser kein Direktoperand sein.
- Z.B. MOV AL, 0C3h ; AL = 1100 0011b
  AND AL, 66h ; AL AND 0110 0110b
  ; Ergebnis AL = 0100 0010b = 42h

TEST

- Arbeitet genau wie AND mit dem Unterschied, dass er das Ergebnis nicht in den ersten Operanden zurückschreibt.
- Setzt die Flags wie bei AND

OR – das logische ODER

OR	0	1
0	**0**	**1**
1	**1**	**1**

- Ein bitweise logisches ODER: Die Ergebnisbits sind nur dann gleich Null, wenn beide Operandenbits gleich Null sind, sonst Eins.
- Für die Operanden gilt das gleiche wie bei AND

XOR – das exclusive ODER

XOR	0	1
0	**0**	**1**
1	**1**	**0**

- Ein bitweise logisches exclusives ODER: Ein Ergebnisbit ist gleich Eins, wenn die Operandenbits ungleich sind, sonst gleich Null.
- Beispiel:

```
MOV AL, 0C3h      ;          AL = 11000011b
XOR AL, 033h      ;          XOR 00110011b
                  ; Ergebnis AL = 11110000b
```

NOT – bitweise Invertierung

NOT	
0	**1**
1	**0**

- Der NOT-Befehl invertiert alle Bits eines Operanden und daher braucht er nur einen Operanden
- Beispiel:

```
MOV AL, 0E5h      ;                AL = 11100101b
NOT AL            ; Ergebnis: AL = 00011010b = 1Ah
```

Wofür sind die bitweisen logischen Befehle nützlich?

- Der **AND**-Befehl ist nützlich um **ausgewählte Bits** eines Operanden **zu löschen** (auf Null zu setzen) und **bestimmte Bits zu isolieren**
  - z.B. AND AX, 1111 1111 1111 0111b - welches Bit wird hier gelöscht?
  - z.B. Setze Bits 0, 2 und 3 auf Null: AND a, 11110010b
- Man benutzt TEST anstatt AND wenn man den Operanden unverändert behalten will.
  - z.B. TEST AL, 01h - man kann überprüfen ob AL eine gerade oder ungerade Zahl ist (ohne den Wert aus AL zu überschreiben)
- **OR** ist geeignet, um **ausgewählte Bits** eines Operanden **gleich eins zu setzen**.
  - z.B.

```
MOV AL, 0CCh        ;   AL = 1100 1100b
OR AL, 2h           ; AL OR 0000 0010b
                    ; Ergebnis AL = 1100 1110b = CEh
```

Wofür sind die bitweisen logischen Befehle nützlich?

- Der **XOR-Befehl** kann benutzt werden um gezielt **einzelne Bits zu invertieren**.
  - z.B. XOR AX,02h invertiert Bit 1 und lässt alle anderen Bits unverändert
- **Schnelles Null-setzen**:
  - XOR AX, AX

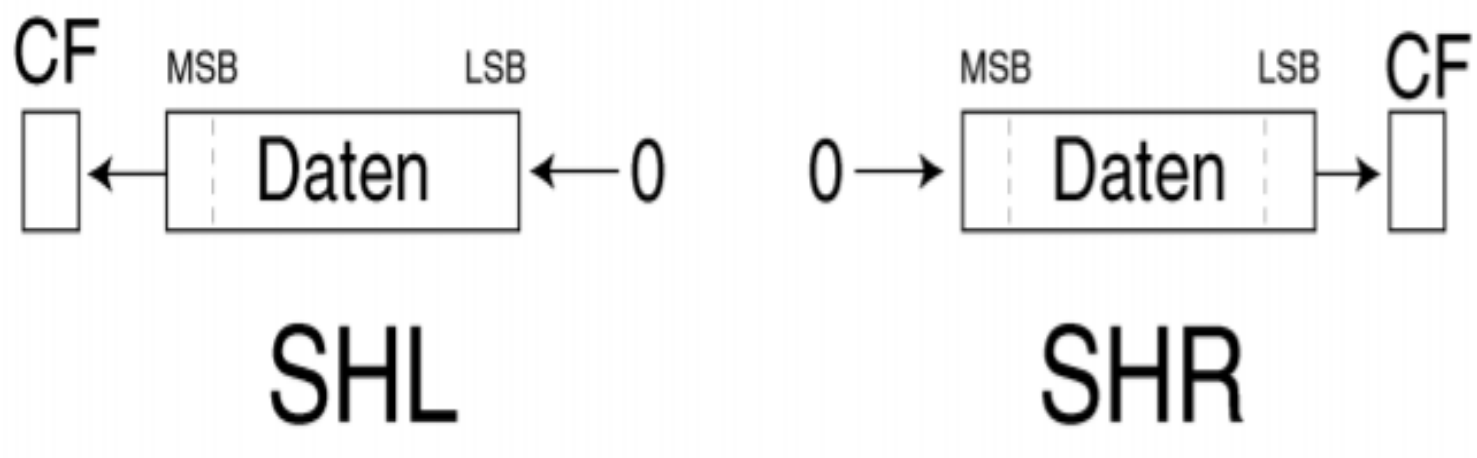
Schiebe- und Rotationsbefehle

- Erlauben es, ein komplettes Bitmuster nach links oder rechts zu schieben
- Wenn das Bit, dass an einem Ende herausfällt, am anderen Ende der Datenstruktur wieder eingesetzt wird spricht man von Rotation, sonst von Schieben (Shift)
- Das **letzte herausgefallene Bit** wird ins **Carryflag** geschrieben.
- Das bearbeitete Bitmuster kann in einem Register oder im Hauptspeicher liegen und 8, 16 oder 32 Bit umfassen.

Schiebe- und Rotationsbefehle

- Syntax: *Schiebe-/Rotationsbefehl Reg/Mem, Konstante/CL*
- Umfasst immer zwei Operanden:
  - das zu bearbeitende Bitmuster und
  - die Anzahl Bits die geschoben oder rotiert werden soll
- Die Bitzahl kann eine Konstante sein oder in CL stehen.
- Für folgende Beispiele seien die Anfangswerte:
  - **AL = abcdefgh**, wobei a-h Bits sind, und
  - **CF=k**

SHL (Shift Left), SHR (Shift Right)



- Einfaches Schieben nach links oder rechts, die frei werdenden Bitstellen werden mit einer Null aufgefüllt.

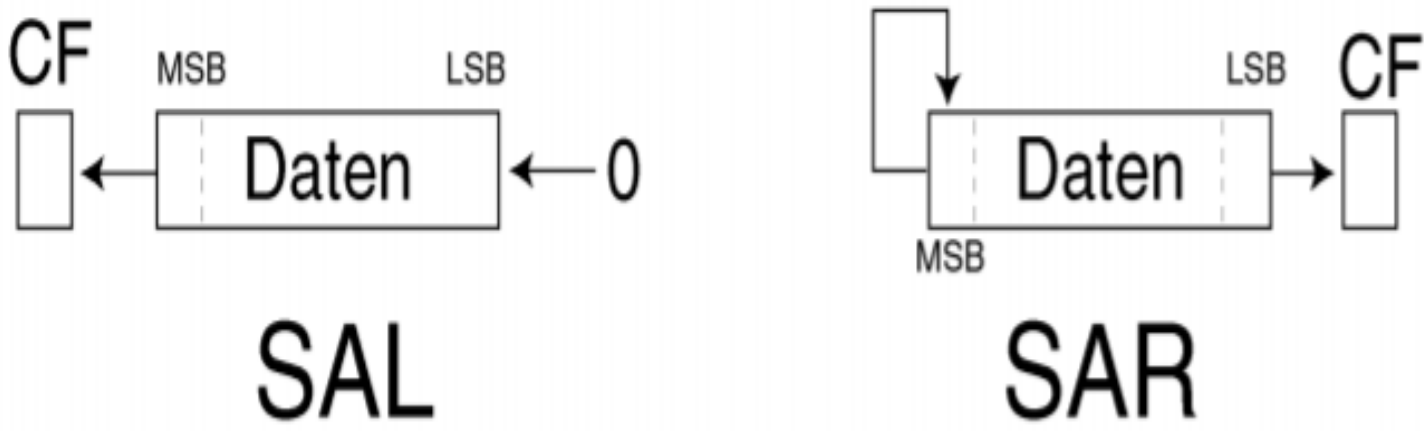
SHL (Shift Left), SHR (Shift Right)

- Beispiel: mov cl, 2
  shl al, cl ; AL=cdefgh00, CF=b

- Beispiel: mov cl, 1
  shr al, cl ; AL=0abcdefg, CF=h

- Bem. Für **vorzeichenlose** Binärzahlen entspricht das einfache Schieben um ein Bit **nach links einer Multiplikation mit zwei**, und **nach rechts** einer **Division durch zwei** (funktioniert aber nur bei vorzeichenlosen Zahlen!).

SAL (Shift Arithmetic Left), SAR (Shift Arithmetic Right)



- SAL arbeitet exakt wie SHL.
- SAR funktioniert folgendermaßen: beim Schieben nach rechts wird auf das frei werdende Bit das Most Significant Bit reproduziert.

SAL (Shift Arithmetic Left), SAR (Shift Arithmetic Right)

- Beispiel: mov cl, 2

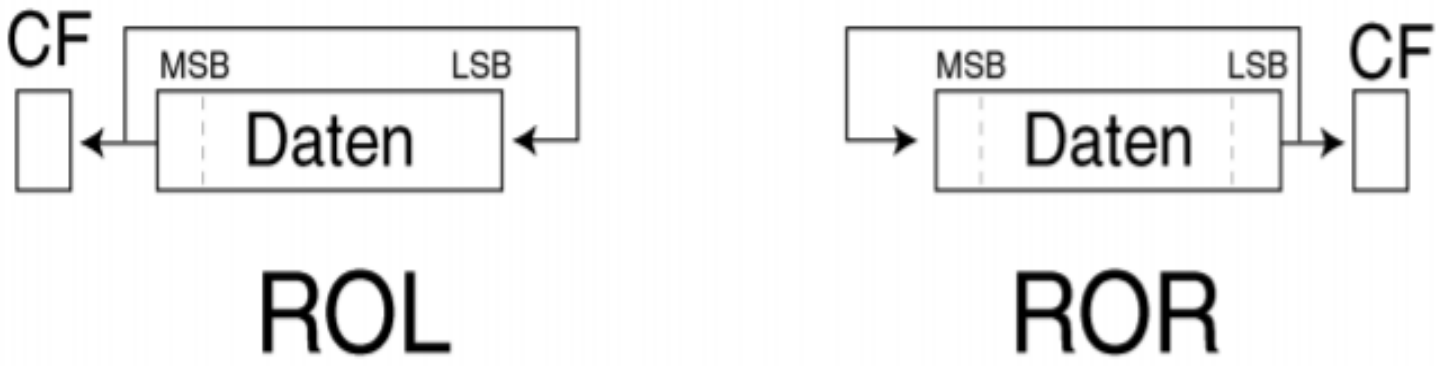
  sar al, cl ; AL=aaabcdef, CF=g

- Bem. Leisten Division durch zwei oder Multiplikation mit zwei auch bei **vorzeichenbehafteten Zahlen**!
- Beispiele:

```
MOV AL, -1   ; AL = 1111 1111b = -1
SAL AL, 1    ; AL = 1111 1110b = -2
SAL AL, 1    ; AL = 1111 1100b = -4
```

```
MOV AL, -16   ; AL = 1111 0000b = -16
SAR AL, 1     ; AL = 1111 1000b = -8
SAR AL, 1     ; AL = 1111 1100b = -4
```

ROL (Rotate Left), ROR (Rotate Right)



- Einfache Rotationen nach links oder rechts, d.h. das herausgefallene Bit kommt auf die freiwerdende Bitstelle und ins Carryflag.

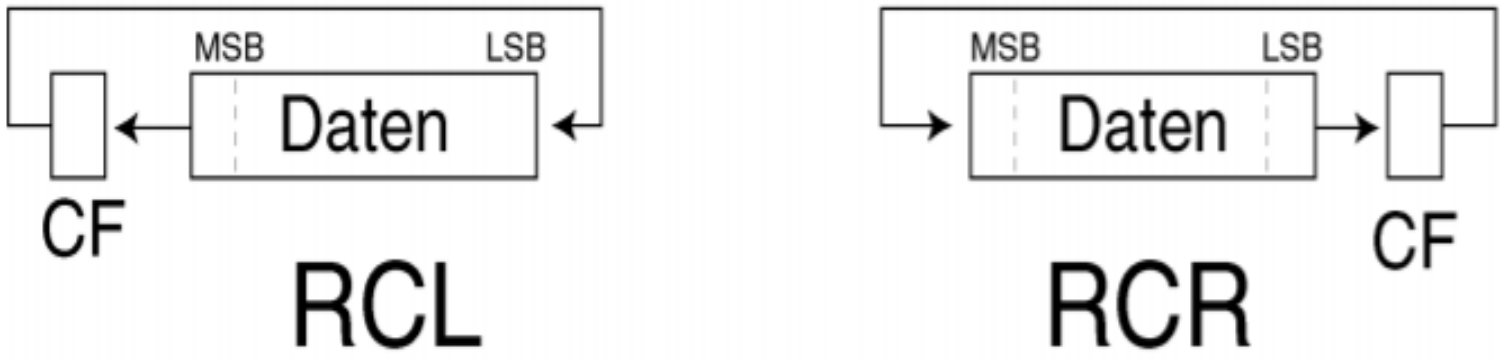
ROL (Rotate Left), ROR (Rotate Right)

- Beispiele:

```
mov cl, 2
rol al, cl   ;  AL=cdefghab, CF=b

mov cl, 2
ror al, cl   ;  AL=ghabcdef, CF=g
```

RCL (Rotate Carry Left), RCR (Rotate Carry Right)



- Ähnlich ROL und ROR, mit dem Unterschied, dass hier **das Carryflag als Bit auf die freiwerdende Stelle gelangt** und das herausgefallene Bit ins Carryflag kommt.

RCL (Rotate Carry Left), RCR (Rotate Carry Right)

- Beispiele:

```
mov cl, 2
rcl al, cl   ;  AL=cdefghka, CF=b

mov cl, 2
rcr al, cl   ;  AL=hkabcdef, CF=g
```

STC (Set Carryflag), CLC (Clear Carryflag)

- Um den Wert des Carryflags zu bestimmen benutz man die zwei Befehle: STC und CLC.
- STC setzt den Wert CF = 1
- CLC setzt den Wert CF = 0.

Aufgabe

Gegeben seien die Wörter A und B. Berechne das Wort C, wobei:

- **Die Bits 0-4 aus C sind gleich mit den Bits 5-9 aus A**
- **Die Bits 5-8 aus C sind gleich mit den Bits 2-5 aus B**
- **Die Bits 9-10 aus C haben den Wert 1**
- **Die Bits 11-12 aus C haben den Wert 0**
- **Die Bits 13-15 aus C sind komplementär zu den Bits 1-3 aus A**

Hinweise

- Man muss **Sequenzen von Bits isolieren** (unverändert behalten, wobei alle anderen Bits Null sind). Dafür benutzt man **AND mit einer Maske** von 1 auf den gewünschten Positionen.
- Die isolierten Bits verschiebt man auf den richtigen Positionen mit Rotationsoperationen.
- Man bildet das Endergebnis mit einer **OR-Operation zwischen den Zwischenergebnisse**.

Hausaufgabe

- **Gegeben sei das Wort A. Berechne die ganze Zahl *n* dargestellt auf den Bits 11-14 aus A. Dann erhalte das Wort B indem man A mit *n* Positionen nach rechts rotiert.**