Welches Register wird für was verwendet ?

AX – Register:
Arithmetische Funktionen

BX – Register:
Arbeitsspeicher adressieren (lea ; mov bx, offset), kann als Zeiger im Datensegment verwendet werden (siehe Direkter Speicherzugriff S.5)

CX – Register:
Für Zählschleifen (mit LOOP)

DX – Register:
Multiplikation, Division, Adressierung von Ports (I/O-Kanäle)

Was darf man nicht ?

Segmentregister (DS, CS, ES, SS) dürfen nicht direkt geladen werden !!!
Nur so:

```
mov bx, Segmentadresse   ; Segmentadresse in BX laden
mov ds,bx                ; BX in DS schreiben
```

Der Instruction-Pointer kann nicht geladen werden !!!

Daten können nicht von Speicherzelle zu Speicherzelle kopiert werden !!!

Anpassung der Ladevorgänge an Variablenlängen:

Beispiele:

```
ByteTabelle     db    00h , 01h , 02h , 03h
WordTabelle     dw    0FFFFh , 0AAAAh, 1234h

mov bl, ByteTabelle                 ; Ohne Anpassung da gleich Inhaltslänge (Byte)

mov bx, WORD PTR ByteTabelle        ; Mit Anpassung durch WORD PTR
                ; da Ziel (BX) länger ist als ein Datenelement von ByteTabelle

mov bx, WordTabelle                 ; Ohne Anpassung da gleich Inhaltslänge (Word)

mov bl, BYTE PTR WordTabelle        ; Mit Anpassung durch  BYTE PTR
                ; da Ziel (BL) kleiner ist als ein Datenelement von WordTabelle
```

Ersatzcode für LEA:

```
lea bx, daten      entspricht      mov bx, OFFSET daten
```

Verwendung von TEST und CMP:

TEST BX, 01000100b
führt eine UND-Verknüpfung zwischen BX und dem Wert durch. BX bleibt unverändert, aber die FLAGS werden gesetzt.
⇒ **Eignet sich hervorragend zum prüfen von gesetzten Bits**

CMP BX, AX
fürt eine Subtraktion durch (BX – AX), verändert aber die Werte in BX und AX nicht. Aber die Flags werden entsprechend dem Ergebnis gesetzt.
⇒ **Eignet sich für die Überprüfung von Zahlen**

Unbedingte Sprungbefehle JMP:

JMP SHORT Label
springt bis zu 128 Bytes (80h) rückwärts und 127 Bytes (79h) vorwärts vom aktuellen Offset aus.
⇒ Nur für sehr kurze Sprünge geeignet

JMP Label (= JMP NEAR Label)
springt bis zu 32768 Bytes (8000h) rückwärts und 32767 Bytes (7FFFh) vorwärts.
⇒ Kann im aktuellen Codesegment jedes Ziel erreichen

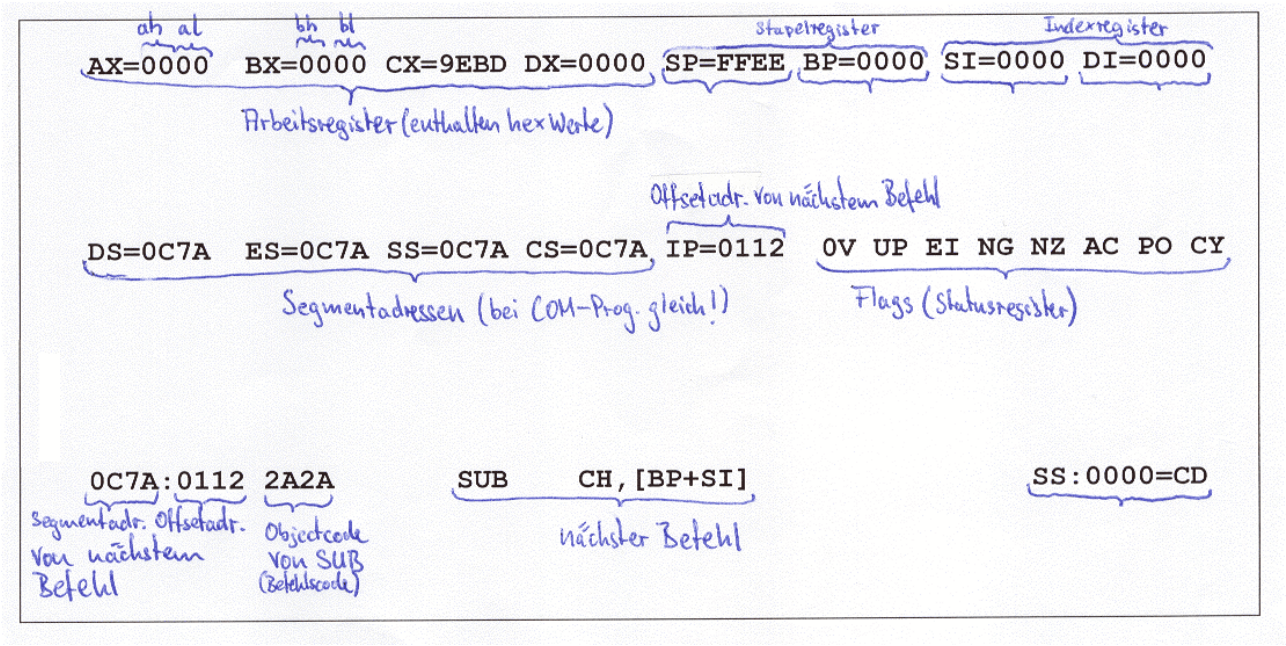
JMP FAR Label (**nicht in COM-Programmen !!!**)
springt im 32-Bit-Adressbereich vorwärts und rückwärts

Debugger-Befehle:

Angaben in [ ] sind optional, d.h. können weggelassen werden

Funktion	Befehl
assemblieren	A [Adresse]
vergleichen	C Bereich Adresse
Bereich anzeigen	**D [Bereich]**
eingeben	E Adresse [Liste]
füllen	F Bereich Liste
starten	G [=Adresse] [Adressen]
hex rechnen	H Wert1 Wert2
einlesen	I E/A-Port (I/O port)
laden	L [Adresse] [Laufwerk] [ErsterSektor] [Anzahl]
verschieben	M Bereich Adresse
benennen	N [Pfadname] [Argumentliste]
ausgeben	O E/A-Port Byte
ganzes UP ausführen	**P [=Adresse] [Anzahl]**
beenden	Q
Register anzeigen	**R [Register]**
suchen	S Bereich Liste
Prog schrittweise abarbeiten	**T [=Adresse] [Wert]**
disassemblieren	**U [Bereich]**
schreiben	W [Adresse] [Laufwerk] [ErsterSektor] [Anzahl]
Expansionsspeicher zuordnen	XA [#Seiten]
Expansionsspeicher freigeben	XD [Zugriffsnummer]
Expansionsspeicher abbilden	XM [LSeite] [PSeite] [Zugriffsnummer]
Expansionsspeicherstatus anzeigen	XS

Debugger-Ausgabe des Befehls' t ':



Stackbelegung bei CALL-Aufrufen: (gilt auch bei CALL NEAR-Aufrufen)

Beim CALL oder CALL NEAR-Aufruf wird erst der aktuelle Inhalt des **IP-Register** (Größe=2 Byte) auf den Stack geschoben und dann die Adresse des Unterprogramms in das IP-Register geladen.

Nun folgen im UP eventuelle PUSH-Befehle um Registerinhalte auf den Stack zu sichern.

Am Ende des UP folgen eventuelle POP-Befehle um Registerinhalte aus dem Stach zurückzuholen.

Beim RET-Befehl wird das **IP-Register** (2 Byte) aus dem Stack wiederhergestellt.
Das IP-Register enthält damit die Adresse des auf den CALL-Befehl folgenden Befehls.

Stackbelegung bei CALL FAR-Aufrufen:

Beim CALL FAR -Aufruf wird erst der aktuelle Inhalt des **CS-Registers** (2 Byte) auf den Stack geschoben, der aktuelle Inhalt des **IP-Register** (2 Byte) auf den Stack geschoben und dann die Adresse des Unterprogramms in das CS und IP-Register geladen.

Nun folgen im UP eventuelle PUSH-Befehle um Registerinhalte auf den Stack zu sichern.

Am Ende des UP folgen eventuelle POP-Befehle um Registerinhalte aus dem Stach zurückzuholen.

Beim RET-Befehl wird **erst** das **IP-Register** (2 Byte) aus dem Stack wiederhergestellt und dann das **CS-Register** aus dem Stack wiederhergestellt.
Das Registerpaar CS:IP enthält damit die Adresse des auf den CALL FAR-Befehl folgenden Befehls.

Stackbelegung bei Interupt-Abarbeitung:

Nach einer Interuptanforderung durch Hardware oder Software, und freigegebenen Interupts (nicht maskiert) läuft folgendes ab.
Als erstes wird der aktuelle Inhalt des **FLAGS-Registers** (2 Byte) auf den Stack geschoben, dann der aktuelle Inhalt des **CS-Registers** (2 Byte) auf den Stack geschoben, dann der aktuelle Inhalt des **IP-Register** (2 Byte) auf den Stack geschoben und dann die Adresse des Unterprogramms in das CS und IP-Register geladen.

Nun folgen n der ISR (Interupt Service Routine) eventuelle PUSH-Befehle um Registerinhalte auf den Stack zu sichern.

Am Ende der ISR folgen eventuelle POP-Befehle um Registerinhalte aus dem Stach zurückzuholen.

Beim IRET-Befehl wird **erst** das **IP-Register** (2 Byte) aus dem Stack wiederhergestellt, dann das **CS-Register** aus dem Stack wiederhergestellt und dann das **FLAG-Register** aus dem Stack wiederhergestellt.

Direkter Speicherzugriff:

Am Beispiel des Beschreibens des Bildschirmspeichers.
Im Bildschirmspeicher hat jedes angezeigte Zeichen 2 Byte Speicherplatz.
im 1. Byte steht der Wert des ASCII-Zeichens. (Siehe ASCII-Tabelle)
Im 2. Byte steht das Anzeigeattribut des Zeichens es setzt sich wie folgt zusammen:

Bit	7	6	5	4	3	2	1	0
Parameter	BL	Hinter			Vorder			

Bit 0 bis 3:
4 – bit Vordergrundfarbe des Zeichens
Werte 0 – 15

Bit 4 bis 6:
3 – bit Hintergrundfarbe des Zeichens
Werte 0 – 7

Bit 7:
1 = Blinken ein, 0 = Blinken aus

Farbe	**Bits**	**Wert**	**Hinter**	**Vorder**
BLACK	0000	0 = 0H	Ja	Ja
BLUE	0001	1 = 1H	Ja	Ja
GREEN	0010	2 = 2H	Ja	Ja
CYAN	0011	3 = 3H	Ja	Ja
RED	0100	4 = 4H	Ja	Ja
MAGENTA	0101	5 = 5H	Ja	Ja
BROWN	0110	6 = 6H	Ja	Ja
LIGHTGRAY	0111	7 = 7H	Ja	Ja
DARKGRAY	1000	8 = 8H	Nein	Ja
LIGHTBLUE	1001	9 = 9H	Nein	Ja
LIGHTGREEN	1010	10 = 0AH	Nein	Ja
LIGHTCYAN	1011	11 = 0BH	Nein	Ja
LIGHTRED	1100	12 = 0CH	Nein	Ja
LIGHTMAGENTA	1101	13 = 0DH	Nein	Ja
YELLOW	1110	14 = 0EH	Nein	Ja
WHITE	1111	15 = 0FH	Nein	Ja

Berechnung der Offset-Adresse der Speicherzelle aufgrund von Zeile und Spalte:

Offset_ASCII = ((zeile – 1) * 160) + ((spalte – 1) * 2)
Offset_Attribut = Offset_ASCII + 1

Beispielprogramm für den Zugriff auf den Bilschirmspeicher:

```
        push ds                 ; DS-Register sichern (Datensegment)

        mov bx,0B800h           ; BX mit Segmentadresse von Bildschirm laden
        mov ds,bx               ; BX in DS-Register schreiben (Datensegment)

; **** Entweder Beschreiben der einzelnen Speicheradressen ****
        mov bx,160              ; BX mit Offset-Adresse für ASCII-Wert laden
        mov al,01h              ; AL mit ASCII-Wert laden (hier Smiley-Zeichen)
        mov [bx],al             ; AL in Adresse DS:BX schreiben

        mov bx,161              ; BX mit Offset-Adresse für Attribut-Wert laden
        mov al,8Ch              ; AL mit Attribut-Wert laden
                                ; (Hintergrund Black, Vordergrund LightRed, Blinken)
        mov [bx],al             ;AL in Adresse DS:BX schreiben

; **** Oder Beschreiben beider Speicheradressen ***
        mov bx,166              ; BX mit Offset-Adresse für ASCII-Zeichen laden
        mov al,06h              ; AL mit ASCII-Wert laden (hier Herz-Zeichen)
        mov ah,0Fh              ; AH mit Attribut-Wert laden
                                ; (Hintergrund Black, Vordergrund White, Blinken)
        mov [bx],ax             ; AX in Adresse DS:BX schreiben

        pop ds                  ; DS-Register wiederherstellen
```

Direkte Portzugriffe mit Assembler: (am Beispiel der Seriellen Schnittstelle)

Basisadressen der Seriellen Schnittstellen:

COM1	Basisadresse 03F8h	IRQ 4
COM2	Basisadresse 02F8h	IRQ 3
COM3	Basisadresse 03E8h	IRQ 4
COM4	Basisadresse 02E8h	IRQ 3

Die Portadresse der Register der Seriellen Schnittstelle setzt sich aus der Basisadresse der Schnittstelle und des Offset des Registers zusammen.

Registeradresse = Basis-Adresse + Offset des Registers

Beispielprogramm zum auslesen des Leitungsstatus-Registers (Offset 05h):

```
push dx                 ; DX auf Stack sichern
push ax                 ; AX auf Stack sichern

mov dx, 02E8h + 05h     ; DX mit Basisadresse der Schnittstelle + Offset laden
in al,dx                ; Wert aus Adresse in DX in AL einlesen

; Hier können Befehle zum Weiterverarbeiten des Wertes in AL stehen

pop ax                  ; AX aus Stack wiederherstellen
pop dx                  ; DX aus Stack wiederherstellen
```

Zum Einstellen der Baudrate bei der seriellen Schnittstelle muss beachtet werden, dass vor dem Schreiben des Teilers für die Baudrate, das DLAB-Bit gesetzt werden muss. Nach dem Schreiben des Teilers sollte das DLAB-Bit wieder gelöscht werden.

Beispielprogramm siehe nächste Seite !!!!

Beispielprogramm zum Einstellen der Baudrate:

```
        push dx                 ; DX auf Stack sichern
        push ax                 ; AX auf Stack sichern

        mov dx, 02E8h + 03h     ; DX mit Basisadresse der Schnittstelle + Offset des
                                ; Datenformat-Registers (=03h) laden

; *** Folgender Teil ist optional ***
        mov ax,0                ; AX auf 0 setzen
        in al,dx                ; Wert aus Adresse in DX in AL einlesen
        push ax                 ; AX und damit alten Inhalt von Datenformat-Register
                                ; auf Stack sichern
; *** Ende Option ***

        ; ** DLAB-Bit setzen
        mov al,1000000b         ; AL mit Muster für gesetztes DLAB-Bit laden
        out dx,al               ; AL in Datenformat-Register schreiben

        ; ** Baudrate einstellen
        mov dx, 02E8h + 00h     ; DX mit Basisadresse der Schnittstelle + Offset des
                                ; Teilerlatch-Registers (=00h) laden
        mov al, 00h             ; AL mit Lower-Byte des Teilers laden
        mov ah, 01h             ; AH mit Higher-Byte des Teilers laden
        out dx,ax               ; AX in Teilerlatch-Register schreiben

        ;** DLAB-Bit löschen
        mov dx, 02E8h + 03h     ; DX mit Basisadresse der Schnittstelle + Offset des
                                ; Datenformat-Registers (=03h) laden
        mov al,00000000b        ; AL mit Muster für gelöschtes DLAB-Bit laden
        out dx,al               ; AL in Datenformat-Register schreiben

; *** Folgender Teil gehört noch zum obigen optionalen Teil ***
        pop ax                  ; Alten Inhalt von Datenformat-Register von Stack holen
                                ; und in AX speichern
        out dx,al               ; AL in Datenformat-Register schreiben
; *** Ende Option ***

        pop ax                  ; AX aus Stack wiederherstellen
        pop dx                  ; DX aus Stack wiederherstellen
```