Seminar 2

Einführung: Register, Flags, Variablen, Konstanten

Für die Verarbeitung der Daten verfügt der Computer über einige **interne Speicherplätze**, die sog. **Register**.

Bei den vier Allzweckregistern **EAX, EBX, ECX und EDX** (E kommt von extended) lassen sich die unteren 16 Bit als AX, BX, CX und DX ansprechen, und diese zusätzlich auch byteweise als AL und AH, BL und BH, CL und CH, DL und DH.

EAX - Hauptrechenregister (bei einigen Rechenbefehlen zwingend benutzt werden muss und bei anderen günstig ist)

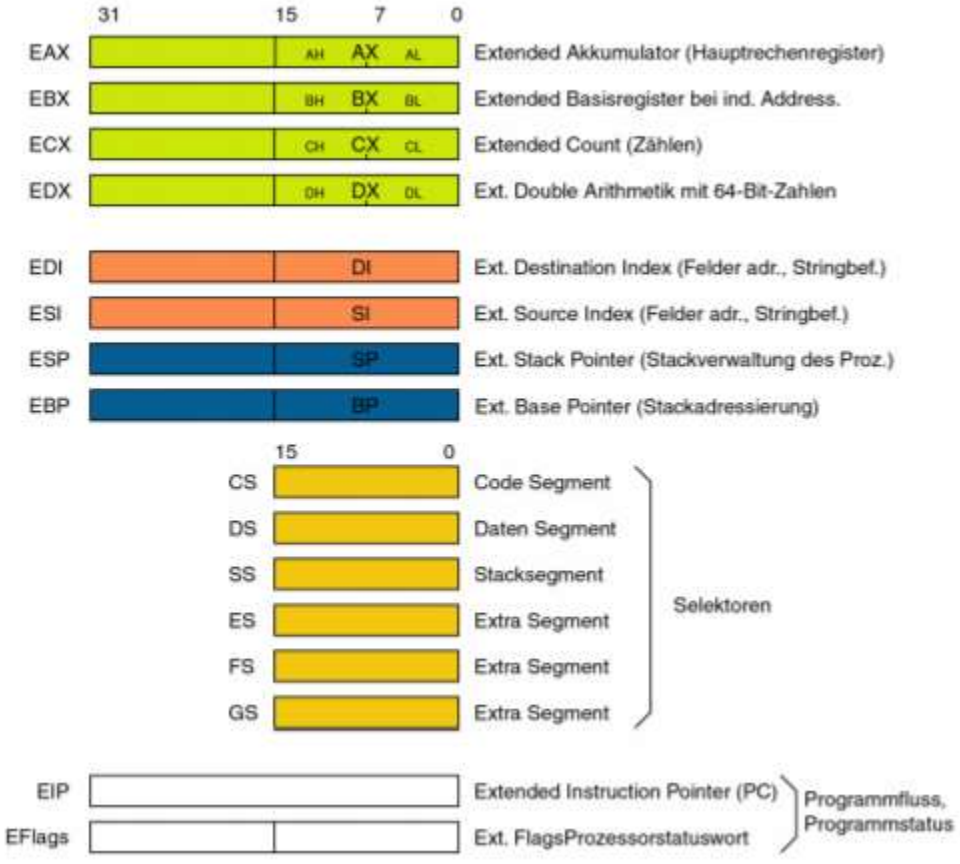
ECX – Zählerregister (wird bei Schleifen und Stringbefehlen zwingend als Zähler eingesetzt)

ESI (Extended Source Index) ist der Zeiger auf den Speicherplatz, der als Datenquelle dient

EDI (Extended Destination index) ist das Ziel

EBP und ESP dienen zur Adressierung des Stacks, eines besonderen Speicherbereiches, der als Last-in-First-out-Speicher organisiert ist.
In ESP (Extended Stack Pointer) ist der Zeiger auf die aktuelle Spitze des Stacks gespeichert, d.h. das zuletzt auf den Stack gebrachte Wort.

Die Register **CS, DS, SS, ES, FS und GS** sind sogenannte **Segmentregister.**



Flags (1 Bit) haben unterschiedliche Bedeutungen.
Ein **Flag ist gesetzt** falls Flag = 1 und ein **Flag ist gelöscht** falls Flag = 0.

Variable Declaration mit Anfangswert
a DB 0A2h
c DD 1230023h

Variable Declaration ohne Anfangswert
a RESB 2 ; 2 Bytes sind reserviert
b RESW 1 ; 1 Wort ist reserviert

Definition von Konstanten
zehn EQU 10

- **Carryflag (CF)**

Das Carryflag, CF, **Übertragsflag,** ist gesetzt, wenn bei der letzten Operation der vorzeichenlose Wertebereich überschritten wird. Anders ausgedrückt: wenn die Anzahl der vorhandenen Bits für das Ergebnis nicht ausreicht

```
mov al,250
add al,10
```

- **Overflowflag (OF)**

Das Overflowflag, OF, **Überlaufsflag,** ist gesetzt, wenn bei der letzten Operation der vorzeichenbehaftete Wertebereich überschritten wird. Im Gegensatz zum Carryflag betrifft das Overflowflag das Rechnen mit vorzeichenbehafteten Zahlen, also Zahlen die positiv und negativ sein können.

```
mov al,120
add al,10          ;Ergebnis 130 überschreitet den Wertebereich
```

Befehle für arithmetische Operationen

In der Assemblersprache wird jeder Maschinenbefehl durch eine einprägsame Abkürzung mit typischerweise 3 Buchstaben dargestellt, das sog. **Mnemonic**.

- **Syntax**: **MOV ziel, quelle** ; ziel = quelle
  Ziel: reg8/16/32/mem8/16/32
  Quelle: reg8/16/32/mem8/16/32|Direktoperand

Der Befehl MOV kopiert den Quelloperanden in den Zieloperanden; der Quelloperand bleibt unverändert.

Ziel und Quelle können Registernamen, Konstante oder Variablen sein.

Einschränkungen:

- Beide Operanden müssen gleiche Bitbreite haben.
- Es können nicht beide Operanden Speicheroperanden sein. (z.B. **NICHT: mov [a], [b]**)
- Es können nicht beide Operanden Segmentregister sein (z.B. **NICHT: mov ES,DS**)

Bsp.

```
MOV AX, 20
MOV BX, AX
MOV [a], AX
```

- **Syntax: ADD operand1, operand2** ; operand1 = operand1+operand2

Operand1: reg8/16/32/mem8/16/32
Operand2: reg8/16/32/mem8/16/32|Direktoperand

Operand1 wird zu Operand2 addiert und das Ergebnis wird in Operand1 abgelegt.
Der Befehl arbeitet für vorzeichenbehaftete und vorzeichenlose Zahlen korrekt.
Bsp. ADD AX, 3
ADD word [a], 5 - warum muss man hier den Datentyp bestimmen?

- **Syntax: ADC operand1, operand2** ; operand1 = operand1+operand2
  Operand1: reg8/16/32/mem8/16/32
  Operand2: reg8/16/32/mem8/16/32|Direktoperand

Operand1 und der Inhalt des Carryflags wird zu Operand2 addiert und das Ergebnis wird in Operand1 abgelegt.

- **Syntax: SUB | SBB Operand1, Operand2**
  Operand1: reg8/16/32/mem8/16/32
  Operand2: reg8/16/32/mem8/16/32|Direktoperand

SUB: Operand2 wird von Operand1 subtrahiert, das Ergebnis wird in Operand1 abgelegt.
SBB: Operand2 und der Inhalt des Carryflags werden von Operand1 subtrahiert, das Ergebnis wird in Operand1 abgelegt
Bsp. MOV AX, [a]
SUB AX, [b] ; AX = a - b

Bem. Die Befehle ADD, ADC, SUB, SBB arbeiten für vorzeichenbehaftete und vorzeichenlose Zahlen korrekt!

- **Syntax: MUL Multiplikator**
  Multiplikator: reg8/16/32/mem8/16/32

MUL führt eine Multiplikation vorzeichenloser Zahlen durch. Im Befehl ist als Operand explizit nur der Multiplikator genannt, der Multiplikand ist immer AL, AX, bzw. EAX.
Je nach Bitbreite des Multiplikators wird eine Byte–, Wort– oder Doppelwort–Multiplikation durchgeführt.
Bei der Byte–Multiplikation ist der **Multiplikand AL** und das **Ergebnis wird in AX abgelegt**.
Bei der Wort–Multiplikation ist **AX der Operand** und das **Ergebnis wird in DX-AX abgelegt**, wobei AX das niederwertige Wort enthält.
Bei der Doppelwort-Multiplikation ist **EAX der Multiplikand** und das **Ergebnis wird in EDX-EAX abgelegt**. Da ein **Overflow** nicht möglich ist, werden **die Flags CF und OF dann gesetzt**, wenn das Ergebnis die Bitbreite der Quelloperanden überschreitet.
MUL führt eine Multiplikation **vorzeichenloser Zahlen** durch!

```
Bsp.  MUL DH            ; AX = AL * DH
      MUL DX            ; DX:AX = AX * DX
      MUL EBX           ; EDA:EAX = EAX * EBX
      MUL BYTE [mem8]   ;AX = AL * BYTE [mem8]
      MUL WORD [mem16]  ; DX:AX = AX * WORD [mem8]
```

- **Syntax: IMUL Multiplikator**
  IMUL führt eine Multiplikation vorzeichenbehafteter Zahlen durch und arbeitet ansonsten wie MUL.
  Aufpassen! Bei der Multiplikation multipliziert man zwei Zahlen mit derselben Darstellungsgröße, aber man speichert das Ergebnis auf eine doppelte Darstellungsgröße!

- **Syntax: DIV | IDIV Divisor**
  DIV führt eine Division vorzeichenloser und IDIV eine Division vorzeichenbehafteter Zahlen durch.

Im Befehl wird **nur der Divisor explizit als Operand aufgeführt**, der Dividend ist immer AX, DX:AX, bzw. EDX:EAX.
Je nach Bitbreite des Divisors wird eine Byte– oder eine Wort–Division durchgeführt.
Dabei werden jeweils **der ganzzahlige Quotient und der Rest separat abgelegt**.
Bei der Byte–Division ist der **Dividend AX**. Der ganzzahlige Teil des **Divisionsergebnis wird in AL und der Rest in AH** abgelegt.
Bei der Wort–Division ist der **Dividend DX:AX**. Der ganzzahlige Teil des **Divisionsergebnis wird in AX** und der **Rest in DX** abgelegt.
Bei der Doppelwort-Division ist der **Dividend EDX:EAX**. Der ganzzahlige Teil des **Divisionsergebnis wird in EAX** und der **Rest in EDX** abgelegt.
Wenn das|die **Zielregister nicht ausreicht** um das Ergebnis aufzunehmen (bei kleinen Divisoren) tritt der sog. **DIVISIONSFEHLER** ein. In diesem Fall wird → **INT 0** ausgelöst.

```
Bsp.    DIV DL      ; AL = AX/DL,  AH = AX % DL
        DIV SI      ; AX = DX:AX / SI,    DX = DX:AX % SI
        DIV EBX     ; EAX = EDX:EAX / EBX,   EDX = EDX:EAX % EBX
```

- **Syntax: INC|DEC Operand**
  Operand: reg8/16/32/mem8/16/32

INC erhöht den Operanden um 1.
DEC erniedrigt den Operanden um 1.

- **Syntax: NEG Operand**
  Operand: reg8/16/32/mem8/16/32

Negiert den Operanden im Zweierkomplement, d.h wechselt dessen Vorzeichen.

Typumwandlung

Beispiele:
Byte -> Word

- 00101100 ⇒ 00000000 00101100 (vorzeichenlos);
- 00101100 ⇒ 00000000 00101100 (vorzeichenbehaftet);
- 10101100 ⇒ 00000000 10101100 (vorzeichenlos);
- 10101100 ⇒ 11111111 10101100 (vorzeichenbehaftet);

Word -> Doubleword

- 00000000 00101100 ⇒ 00000000 00000000 00000000 00101100 (vorzeichenlos);
- 00000000 00101100 ⇒ 00000000 00000000 00000000 00101100 (vorzeichenbehaftet);
- 00000000 10101100 ⇒ 00000000 00000000 00000000 10101100 (vorzeichenlos);
- 11111111 10101100 ⇒ 11111111 11111111 11111111 10101100 (vorzeichenbehaftet);

Vorzeichenlose Konvertierungen (Typumwandlungen)

Es gibt keine Befehle für vorzeichenlose Konvertierungen, man setzt den Wert aus dem hohenwertigen Teil auf Null, z.B:

- um den Wert aus AL zu AX zu konvertieren: MOV AH, 0
- um den Wert aus AX zu DX:AX zu konvertieren: MOV DX, 0

Vorzeichenbehaftete Konvertierungen (Typumwandlungen)

Man muss die restlichen höherwertige Bits (high bits) mit dem Vorzeichenbit (sign bit) ausgefüllen und dafür gibt es Konvertierungsanweisungen.

- **Syntax: CBW**
  - **Hat keine Operanden!**
  - Wandelt der Byte **AL** in das Wort **AX** um!

- **Syntax: CWD**
  - **Hat keine Operanden!**
  - Wandelt das Wort **AX** in das Doppeltwort **DX:AX** um!

- **Syntax: CWDE**
  - **Hat keine Operanden!**
  - Wandelt das Wort **AX** in das Doppeltwort **EAX** um!
- **Syntax: CDQ**
  - **Hat keine Operanden!**
  - Wandelt das Doppeltwort **EAX** in das Vierfachwort (Quadword) **EDX:EAX** um!