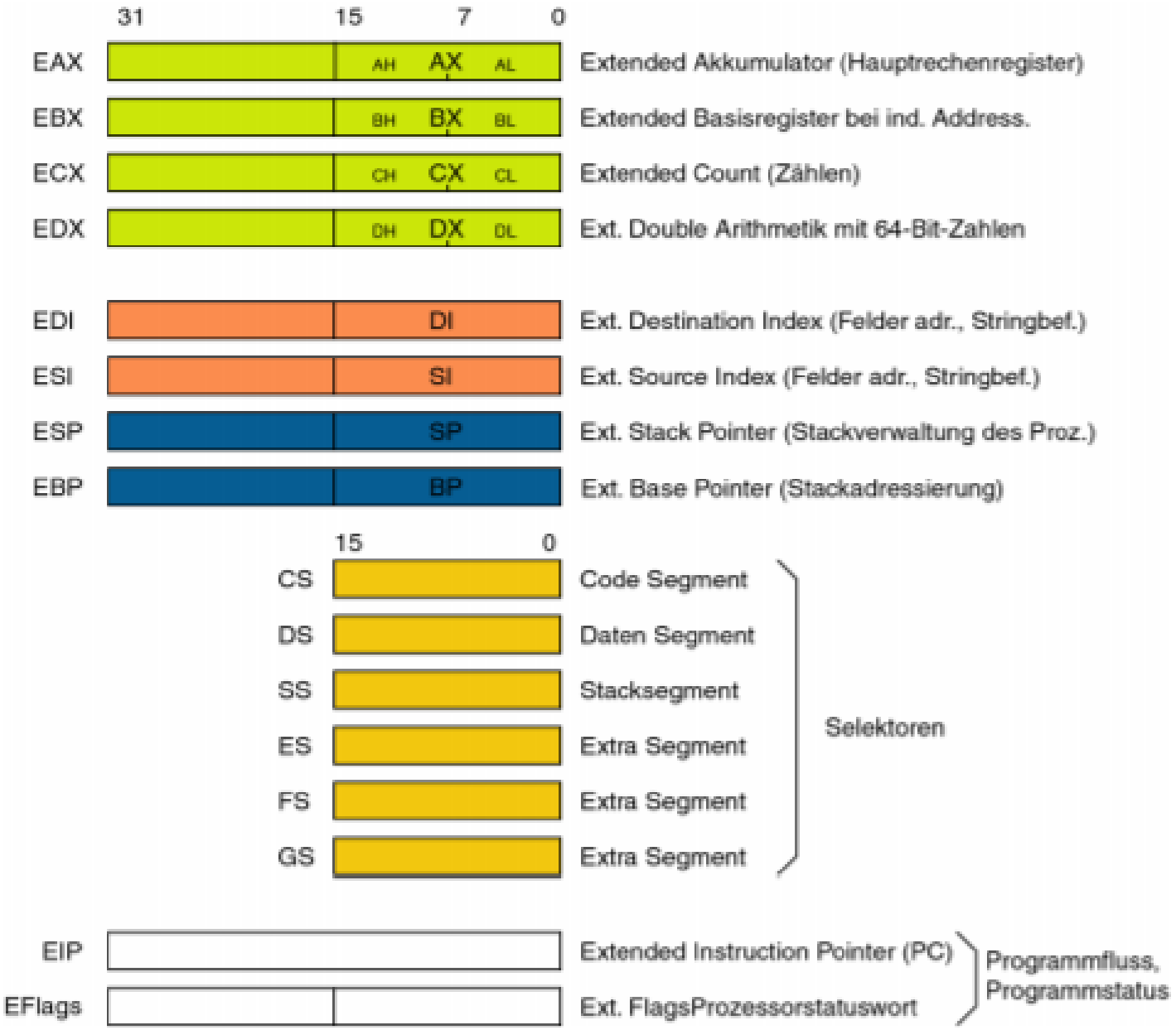
Seminar 2

Rechnerarchitektur

Register

- Für die Verarbeitung der Daten verfügt der Computer über einige **interne Speicherplätze**, die sog. **Register**.
- Allzweckregistern: **EAX, EBX, ECX und EDX** (E kommt von extended)
  - Die unteren 16 Bit lassen sich 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** (muss bei einigen Rechenbefehlen zwingend benutzt werden und bei anderen ist es günstig)
- **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 (Stapel)**, 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**


31
15
7
0
EAX
AH
AX
AL
Extended Akkumulator (Hauptrechenregister)
EBX
BH
BX
BL
Extended Basisregister bei ind. Address.
ECX
CH
CX
CL
Extended Count (Zählen)
EDX
DH
DX
DL
Ext. Double Arithmetik mit 64-Bit-Zahlen
EDI
DI
Ext. Destination Index (Felder adr., Stringbef.)
ESI
SI
Ext. Source Index (Felder adr., Stringbef.)
ESP
SP
Ext. Stack Pointer (Stackverwaltung des Proz.)
EBP
BP
Ext. Base Pointer (Stackadressierung)
15
0
CS
Code Segment
DS
Daten Segment
SS
Stacksegment
ES
Extra Segment
FS
Extra Segment
GS
Extra Segment
Selektoren
EIP
Extended Instruction Pointer (PC)
EFlags
Ext. FlagsProzessorstatuswort
Programmfluss,
Programmstatus

Flags, Variablen, Konstante

- Ein **Flag** (1 Bit) **ist gesetzt** falls Flag = 1 und ein **Flag ist gelöscht** falls Flag = 0.
- **Variable Declaration mit Anfangswert**
  - DB, DW, DD, DQ
  - Beispiele:
    - a DB 0A2h
    - c DD 1230023h
- **Variable Declaration ohne Anfangswert**
  - RESB, RESW, RESD, RESQ
  - Beispiele:
    - a RESB 2 ; 2 Bytes sind reserviert
    - b RESW 1 ; 1 Wort ist reserviert
- **Definition von Konstanten**
  - zehn EQU 10

Übertragsflag/Carryflag (CF)

- Das Carryflag, CF, **Übertragsflag,** ist gesetzt, wenn bei der letzten Operation der **vorzeichenlose Wertebereich überschritten** wird.
- D.h. wenn die Anzahl der vorhandenen Bits für das Ergebnis nicht ausreicht
- Beispiel:
  - mov al,250
  - add al,10
- Beispiel:
  - mov al, 2
  - sub al, 4

Überlaufsflag/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.
- Beispiel:
  - mov al,120
  - add al,10 ;Ergebnis 130 überschreitet den Wertebereich
- Welches ist der maximale Wert auf einem vorzeichenbehaftete Byte?

Befehle für arithmetische Operationen

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

MOV

- **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.
- 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**)
- Beispiele:
  - MOV AX, 20
  - MOV BX, AX
  - MOV [a], AX

ADD

- **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.
- Beispiele:
  - ADD AX, 3
  - ADD word [a], 5 - warum muss man hier den Datentyp bestimmen?

ADC

- **Syntax: ADC operand1, operand2** ; operand1 = operand1+operand2+CF
  - 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.

SUB

- **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
- Beispiel:
  - MOV AX, [a]
  - SUB AX, [b] ; AX = a - b

Bemerkung!

- Die Befehle ADD, ADC, SUB, SBB arbeiten sowohl für vorzeichenbehaftete als auch für vorzeichenlose Zahlen korrekt!

- **Für Multiplikation und Division** gibt es **unterschiedliche Befehle für vorzeichenbehaftete und vorzeichenlose Zahlen**!

MUL

- **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.
- **Multiplikator = Byte ⇒ AX = AL * Multiplikator**
- **Multiplikator = Wort ⇒ DX:AX = AX * Multiplikator**
- **Multiplikator = Doppelwort ⇒ EDX:EAX = EAX * Multiplikator**

Bsp.

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

IMUL

- 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!**
- Da ein **Overflow** nicht möglich ist, werden **die Flags CF und OF dann gesetzt**, wenn das Ergebnis die Bitbreite der Quelloperanden überschreitet

DIV und IDIV

- **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**.
- 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.

DIV

- Falls Divisor = Byte ⇒ AL = AX / Divisor (Quotient)
  AH = AX % Divisor (Rest)
- Falls Divisor = Wort ⇒ AX = DX:AX / Divisor
  DX = DX:AX % Divisor
- Falls Divisor = Doppelwort ⇒ EAX = EDX:EAX / Divisor
  EDX = EDX:EAX % Divisor
- Beispiele:
  - 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

INC und DEC

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

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

NEG

- **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:

- Beispiele:
  - 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) ausfüllen und dafür gibt es Konvertierungsanweisungen
- **Syntax: CBW**
  - Wandelt der Byte **AL** in das Wort **AX** um!
- **Syntax: CWD**
  - Wandelt das Wort **AX** in das Doppeltwort **DX:AX** um!
- **Syntax: CWDE**
  - Wandelt das Wort **AX** in das Doppeltwort **EAX** um!
- **Syntax: CDQ**
  - Wandelt das Doppeltwort **EAX** in das Vierfachwort (Quadword) **EDX:EAX** um!
- **Bem**. Die Befehle für Typumwandlug haben **keine Operanden**! Diese konvertieren immer die Werte für die oben angegebene Registern.

Aufgaben

1. Schreibe ein Programm, das den Ausdruck x:= a + b – c = 3 + 4 – 2 = 5 berechnet, wobei a,b, c – word .

```
bits 32
global start

extern exit
import exit msvcrt.dll
segment data use32 class=data
  ...
segment code use32 class=code
start:
        ...
    push    dword 0          ; exit(0)
    call    [exit]
```

Aufgaben

2. Schreibe ein Programm, das den Ausdruck x:=(a-b*c) / d in der vorzeichenlosen Interpretation berechnet, wobei a, b, c, d – Bytes.
3. Berechne die Differenz zweier Quadwords gespeichert in EDX:EAX und ECX:EBX
4. Berechne die Summe zweier Doppelwörter gespeichert in DX:AX und CX:BX
5. Berechne die Summe zwischen dem Wort BX und dem Doppelwort DX:AX in der vorzeichenlose Interpretation.
6. * ** Subtrahiere das Doppelwort EBX aus dem Quadword EDX:EAX in der **vorzeichenbehaftete** Interpretation.
7. Schreibe ein Programm, das den Ausdruck (a*b)/d – c in der vorzeichenbehaftete Interpretation berechnet, wobei *a, c, d* - Bytes und *b* - Wort.