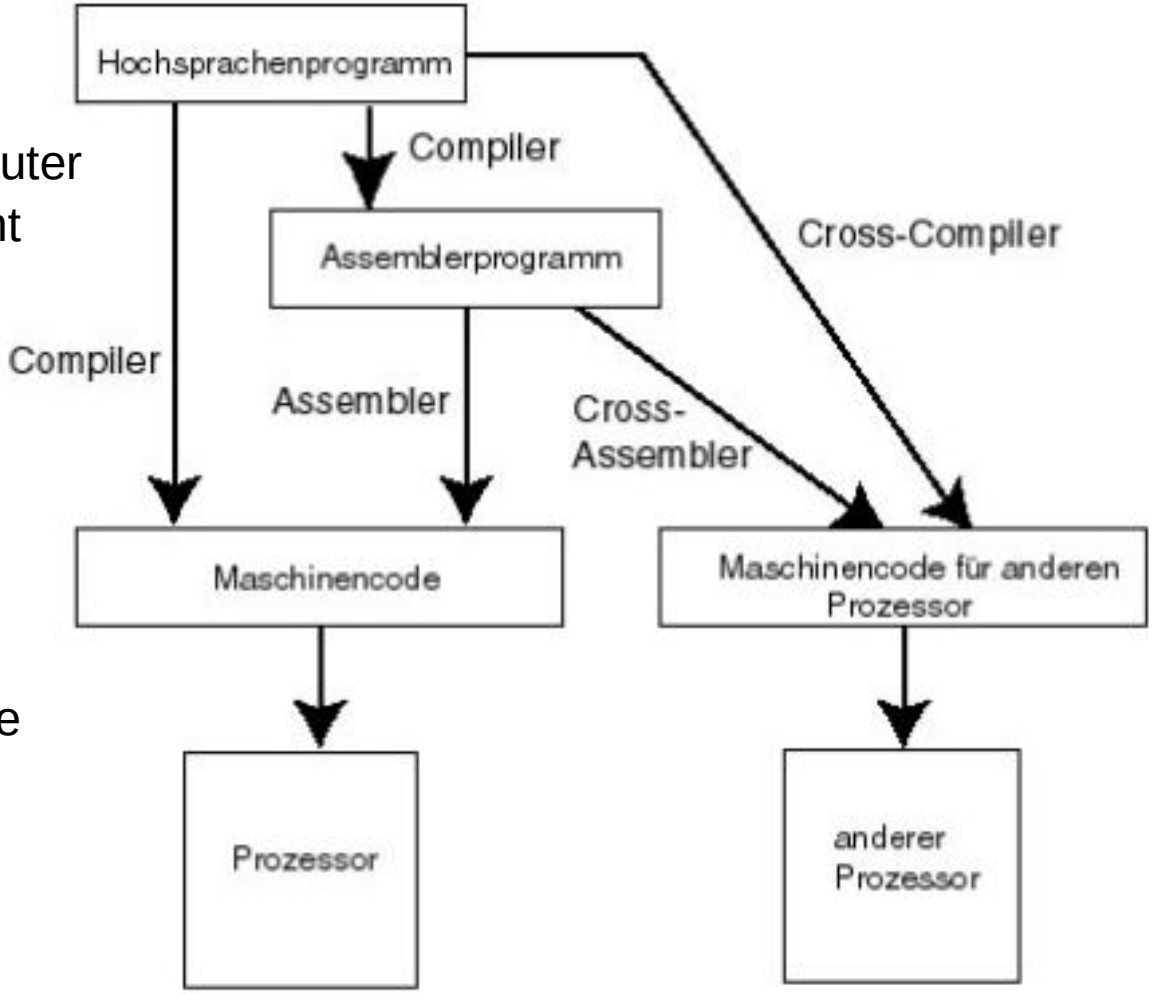
Assembler bei der Codegeneration

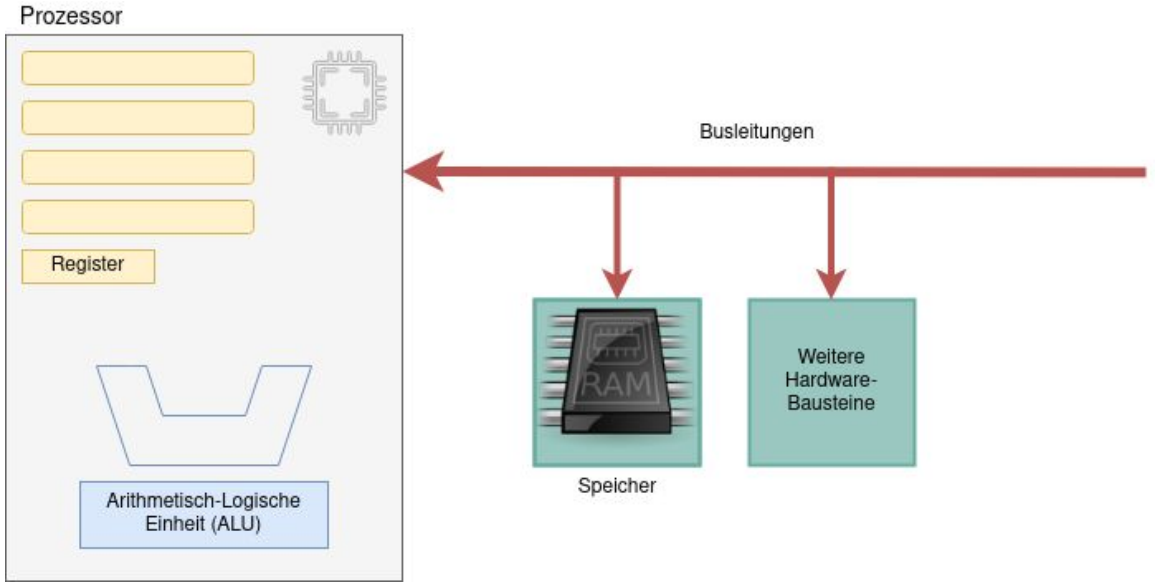
- Optimale Prozessorausnutzung, guter Assemblercode ist sehr performant
- Vollständige Kontrolle über die Prozessorhardware
- Kompakter Code

- Programmierer braucht Kenntnisse des Prozessors
- Jeder Prozessor eigene Assemblersprache
- Reduzierte Portabilität
- Keine Bibliotheken
- Fehler sind oft schwerwiegender
- Bei großen Programmen unhandlich



In Anlehnung an https://homepages.thm.de/~hg6458/AS.pdf (Vorlesungsskript: Die Assemblersprache der intel 80x86-Prozessoren. Klaus Wüst, Gießen, 2003) mit freundlicher Genehmigung des Autors.

Der Prozessor



Bestandteile des Prozessors

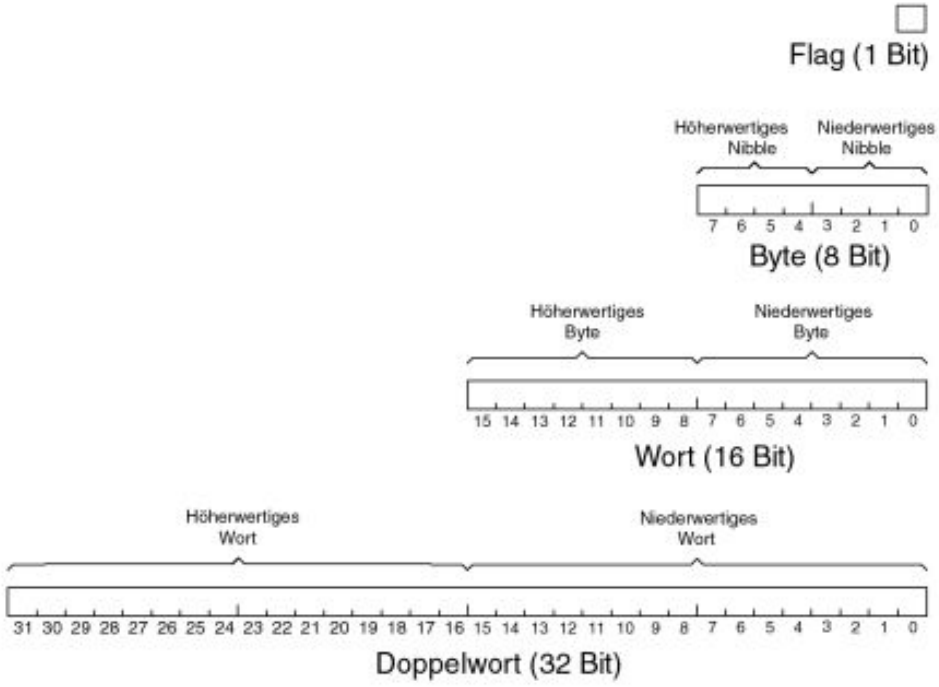
- Das **Steuerwerk** erzeugt die notwendigen Signale für die internen und externen Steuerleitungen (Busschnittstelle)
- Das **Adresswerk** erzeugt auf den Adressleitungen das notwendige Bitmuster, um die im Assemblerbefehl beschriebene Speicherzelle anzusprechen.
- Das **Operationswerk** führt die bitweisen und arithmetischen Operationen auf Datenoperanden aus
- Der **Registersatz** als spezieller *Teil des Operationswerkes* enthält eine gewisse Anzahl prozessorinterner Speicherzellen und Flags

In Anlehnung an https://homepages.thm.de/~hg6458/AS.pdf (Vorlesungsskript: Die Assemblersprache der intel 80x86-Prozessoren. Klaus Wüst, Gießen, 2003) mit freundlicher Genehmigung des Autors.

Register

Ein **Register** ist eine Gruppe von Flipflops (1 Bit-Speicher) mit gemeinsamer Steuerung. Register umfassen meist 8, 16, 32 oder 64 Bit.

1 Bit	–	1 Flag
4 Bit	–	1 Nibble
8 Bit	–	1 Byte
16 Bit	–	1 Wort (bei 80686 Prozessoren, das hat historische Gründe)



MSB LSB

In Anlehnung an https://homepages.thm.de/~hg6458/AS.pdf (Vorlesungsskript: Die Assemblersprache der intel 80x86-Prozessoren. Klaus Wüst, Gießen, 2003) mit freundlicher Genehmigung des Autors.

Register des x86

1985 Intels 8086-Prozessor kommt auf den Markt, der ersten 16-Bit-Prozessor.
(Hauptrechenregister hat 16 Bit, daher die Einheit „Wort“ s.o.)

Auf dem Weg zu den heutigen x86 Prozessoren wurde diese Architektur stets **erweitert –** heute 64 Bit.

Register der 32-Bit Prozessoren

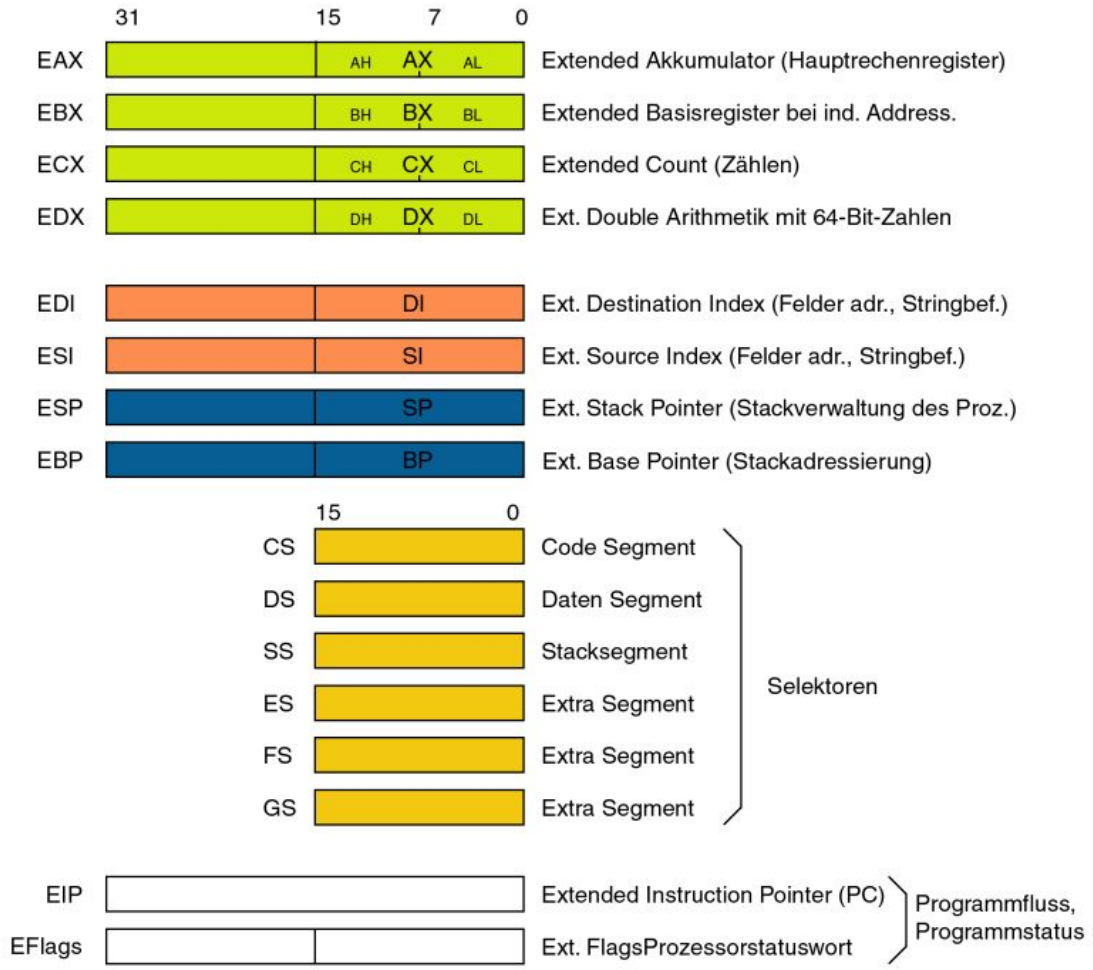
Registernamen beginnen mit einem E für „extended“, weil diese Register von 16 auf 32 Bit erweitert wurden.

Für diese acht Register gilt, dass jeweils die unteren 16 Bit unter dem Namen des früheren 16-Bit-Registers angesprochen werden können.

Damit haben die neueren Prozessoren stets alle Register ihrer Vorgänger, nutzen deren Fähigkeiten aber nicht aus.

Bei den vier Allzweckregistern EAX, EBX, ECX und EDX lassen sich die unteren 16 Bit als AX, BX, CX und DX ansprechen, und diese zusätzlich auch noch byteweise als AL und AH, BL und BH, CL und CH, DL und DH (L für „Low“, H für „High“).

Bei den aktuellen 64Bit Prozessoren gibt es die Register rax, rbx … die wiederum die eax, ebx Register „beinhalten“.



In Anlehnung an https://homepages.thm.de/~hg6458/AS.pdf (Vorlesungsskript: Die Assemblersprache der intel 80x86-Prozessoren. Klaus Wüst, Gießen, 2003) mit freundlicher Genehmigung des Autors.

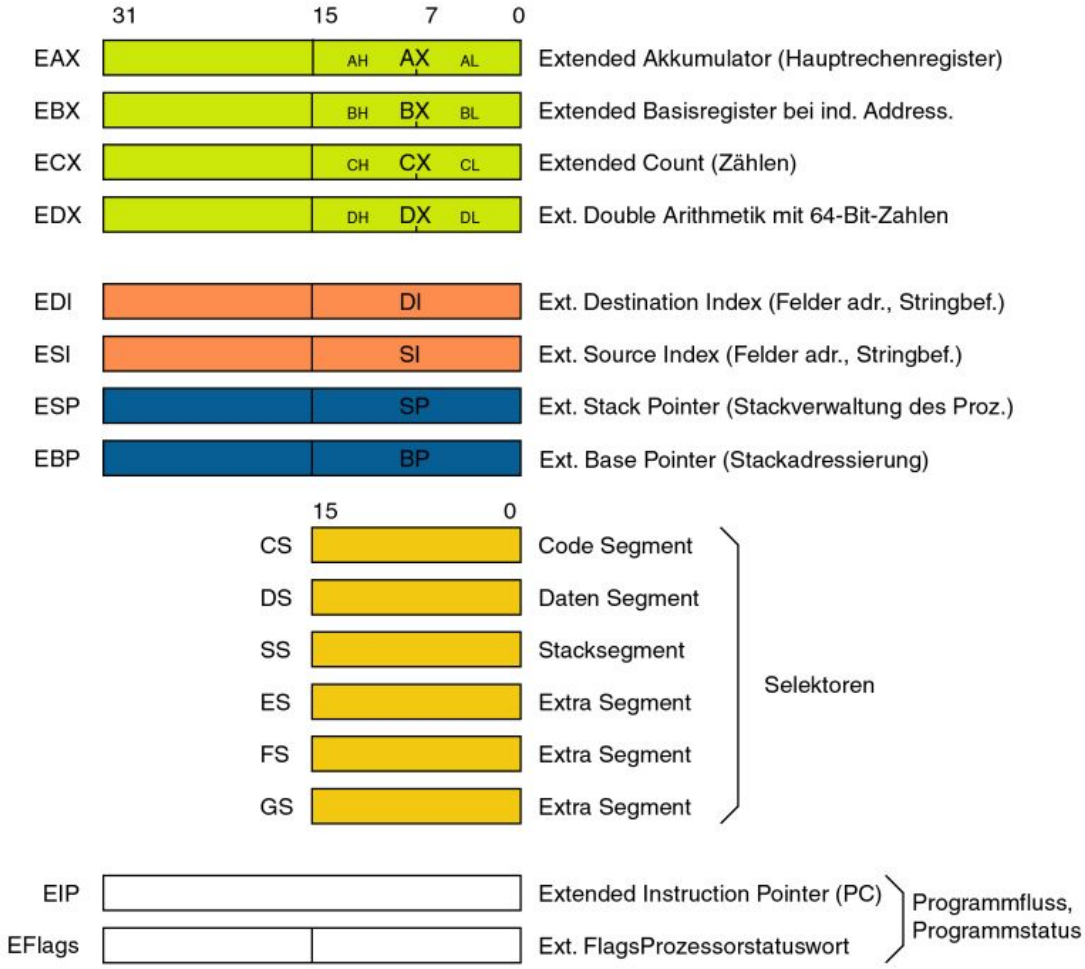
Register des x86

AX ist der primäre Akkumulator; er wird bei der Eingabe/Ausgabe und den meisten arithmetischen Anweisungen verwendet. Bei einer Multiplikationsoperation beispielsweise wird ein Operand je nach Größe des Operanden im Register EAX oder AX oder AL gespeichert.

BX wird als Basisregister bezeichnet, da es bei der indizierten Adressierung verwendet werden kann.

CX wird als Zählregister bezeichnet, da in den Registern ECX und CX die Schleifenanzahl bei iterativen Operationen gespeichert wird.

DX wird als das Datenregister bezeichnet. Es wird auch bei Eingabe-/Ausgabeoperationen verwendet. Es wird auch zusammen mit dem AX-Register und DX für Multiplikations- und Divisionsoperationen mit großen Werten verwendet.



In Anlehnung an https://homepages.thm.de/~hg6458/AS.pdf (Vorlesungsskript: Die Assemblersprache der intel 80x86-Prozessoren. Klaus Wüst, Gießen, 2003) mit freundlicher Genehmigung des Autors.

Flags

Das **EFlag**-Register unterscheidet sich von an den anderen Registern. Die Flipflops in diesen Registern werden einzeln gesteuert, jedes Flipflop hat eine bestimmte Bedeutung – es ist ein Flag (Flagge, Fähnchen).

Sprechweise:

- „Flag gesetzt“ bedeutet Flag=1; auch „ein Flag setzen“
- „Flag gelöscht“ bedeutet Flag=0; auch: „der Befehl löscht das Flag“

Es gibt zwei Gruppen von Flags: **Statusflags** und **Steuerflags**.

Statusflags sind Flags, die der Prozessor nach arithmetischen oder bitweise logischen Operationen setzt, um etwas über das Resultat dieser Operation auszusagen.

Der Programmierer kann diese Flags dann in bedingten Sprungbefehlen abfragen und Programmverzweigungen vom Zustand der Flags abhängig machen.

Das if- des Assemblers

Abschnitte eines Assemblerprogramms

Ein Assemblerprogramm hat drei Abschnitte:

- data

Deklaration von Konstanten, Reservierung von Speicherbereichen

- Bss

Variablen

- text

Programmcode. Der text – Abschnitt muss stets so beginnen, damit der Linker den Start findet:

```
.text
   global _start
_start:
```

Speichersegmente

Segmentiertes Speichermodell: Systemspeicher in Gruppen unabhängiger Segmente unterteilt, auf die mit Zeigern in **Segmentregistern** verwiesen wird. Jedes Segment dient dazu, einen bestimmten Datentyp zu speichern. Ein Segment dient zur Aufnahme von **Befehlscodes**, ein anderes Segment speichert die **Datenelemente** und ein drittes Segment enthält den **Programmstapel**.

Datensegment - .data- und .bss Abschnitt. Der **.data**-Abschnitt wird verwendet, um den Speicherbereich zu deklarieren, in dem Datenelemente für das Programm gespeichert werden. Dieser Abschnitt kann nach der Deklaration der Datenelemente nicht mehr erweitert werden und bleibt während des gesamten Programms statisch.

Der Abschnitt **.bss** ist ebenfalls ein statischer Speicherbereich, der Puffer für Daten enthält, die später im Programm deklariert werden. Dieser Pufferspeicher ist mit Nullen gefüllt.

Codesegment - **.text** -Abschnitt. Dieser definiert einen Bereich im Speicher, in dem die Befehlscodes gespeichert werden. Dies ist ebenfalls ein fester Bereich.

Stack - Dieses Segment enthält Datenwerte, die an Funktionen und Prozeduren innerhalb des Programms übergeben werden.

Speicherbereiche reservieren

Initialisierter Speicher in .data

Directive	Purpose	Storage Space
DB	Define Byte	allocates 1 byte
DW	Define Word	allocates 2 bytes
DD	Define Doubleword	allocates 4 bytes
DQ	Define Quadword	allocates 8 bytes
DT	Define Ten Bytes	allocates 10 bytes

Nicht initialisierter Speicher in .bss

Directive	Purpose
RESB	Reserve a Byte
RESW	Reserve a Word
RESD	Reserve a Doubleword
RESQ	Reserve a Quadword
REST	Reserve a Ten Bytes

Beispiele:

```
Zaehler2 DB 0 ;Def. der Byte-Variablen Zaehler2, Vorbesetzung mit 0

Startchar DB 65 ;Vorbesetzung mit ASCII-Zeichen #65 = ’A’

Startchar DB ’A’;gleiche Wirkung, besser lesbar

Regmaske DB 00110101b ;Vorbesetzung mit binärem Wert (Bitmuster)

Pixely DW 01AFh ;Wort-Variable, Vorbesetzung mit hexadezimalem Wert
```

Mit einer Anweisung können auch gleich mehrere Speicherplätze gleichen Typs, also Felder , definiert werden:

```
Meldung1 DB `Divisionsfehler!`

Meldung1 DB `Hallo Welt`,13,10     ;Vorbesetzung mit einer Zeichenkette, und Steuerzeichen, 12 Byte
                                   ;Speicherplatz
```

In der .bss-Sektion sieht dass folgendermaßen aus:

```
antwort resb 5      ;5 beschreibbare Bytes ohne spezielle Vorbelegung.
```