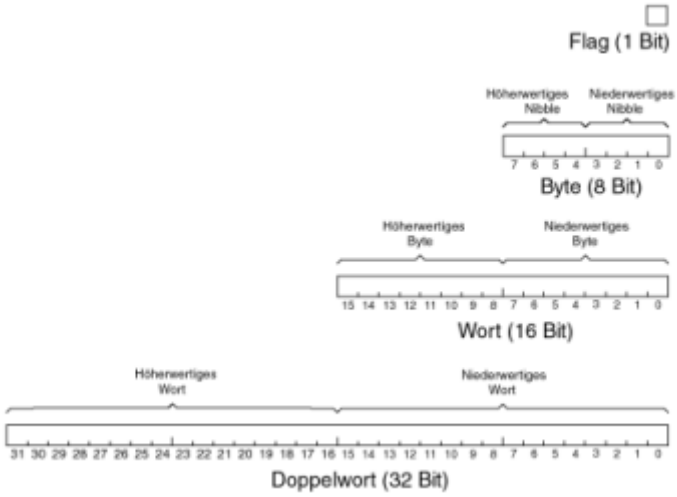
Register und Speicherbereiche

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)



Die Bits der Register sind numeriert, vom LSB (LeastSignificantBit) zum MSB (MostSignificantBit).

Die Register des 80x86

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

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

Das folgende Bild zeigt die Register für die 32Bit 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“.

```
section .data
     msg db 'Displaying 9 stars',0xa ;Message, Aneinanderreihung von "byte"
(db) Bereichen
```

```
    len equ $ - msg  ;Laenge der Message
    s2  times 9 db '*' ; 9 x ein byte (db) mit dem Inhalt "*"


section .text
   global _start     ;must be declared for linker

_start:          ;startadresse
   mov  edx,len  ;Laenge nach edx
   mov  ecx,msg  ;Message nach ecx
   mov  ebx,1    ;file descriptor (stdout)
   mov  eax,4    ;system call number (sys_write)
   int  0x80     ;call kernel

   mov  edx,9    ;message length
   mov  ecx,s2   ;message to write
   mov  ebx,1    ;file descriptor (stdout)
   mov  eax,4    ;system call number (sys_write)
   int  0x80     ;call kernel

   mov  eax,1    ;system call number (sys_exit)
   int  0x80     ;call kernel
```

Hier kommt wieder der **Linux-Systemaufruf** 4 zum Einsatz. Linux-Systemaufrufe funktionieren grob folgendermaßen:

- Lege die Nummer des Systemaufrufs in das EAX-Register
- Speichere die Argumente für den Systemaufruf in den Registern EBX, ECX, usw.
- Rufe den Interrupt (80h) auf
- Das Ergebnis des Systemaufrufs wird normalerweise im EAX-Register zurückgegeben.

Um das erfolgreich zum Einsatz zu bringen, muss man wissen, was ein Systemaufruf in welchem Register erwartet, damit er funktioniert, beim System-Call 4 (Write) ist es folgendermaßen:

Name	**%eax**	**%ebx**	**%ecx**	**%edx**	**%esx**	**%edi**
Write	4	unsigned int (Output Stream)	const char * (Inhalte)	size_t (Länge)	-	-

Das kann man sich ein wenig wie eine Funktion/Methode mit Argumenten vorstellen: Man befüllt zunächst die Register mit den Argumenten und ruft dann den Systemaufruf auf.