x86 Assembler Übung: Größter gemeinsamer Teiler

Klaus Kusche

Gesucht ist wieder eine Assembler-Funktion zum gegebenen Hauptprogramm.
Sie soll den ggT berechnen, und zwar nach dem *Euklid'schen Verfahren mit Restrechnung*:

```
unsigned int ggt(unsigned int a, unsigned int b)
{
    while (b != 0) {
        r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

Hinweise:

- Im Unterschied zur Addition usw., wo man Quelle und Ziel frei wählen kann,
  hat der vorzeichenlose Divisionsbefehl **`div`** *nur einen frei wählbaren Operanden*:
  Die Zahl, durch die man dividiert.

  Die Zahl, die *dividiert wird*, ist immer *doppelt so lang* wie der Operand:
  Im 32-bit-Fall muss der *untere* Teil im **`eax`**-Register liegen und der *obere* im **`edx`**
  (dividiert man so wie wir hier eine Zahl *einfacher Länge*,
  muss man **`edx`** vor dem **`div`** auf **`0`** setzen).

  `div` liefert das *Divisionsergebnis* (das wir nicht brauchen) immer in **`eax`**
  und zugleich den *Divisionsrest* in **`edx`**.

- **`div`** hinterlässt die *Flags* in *undefiniertem* Zustand.
  Man muss also mit einem separaten Befehl prüfen, ob der Divisionsrest **`0`** war
  (mach diese Prüfung von **`edx`** am gleich *Anfang* der Schleife,
  dann erkennt sie auch gleich einen Aufruf mit **`b`** gleich **`0`**).

- Die Herausforderung des Beispiels besteht darin, die Register so zu verwenden,
  dass man möglichst wenig **`mov`**-Befehle braucht.

  Ich empfehle, das **`div`** nicht am Anfang sondern am *Ende* der Schleife zu machen.

  Unmittelbar *vor* dem **`div`** muss man

  - das bisherige **`a`** (wo liegt das am Anfang?) ins **`eax`** kopieren,
    weil es ja die Zahl ist, die dividiert werden soll,
  - das bisherige **`b`** aus dem **`edx`** wegkopieren (weil das **`div`** ja **`edx`** braucht),
    und zwar für den nächsten Schleifendurchlauf gleich dorthin,
    wo bisher das **`a`** war (aus dem **`b`** wird ja das **`a`** des nächsten Durchlaufes),
  - und dann das **`edx`** auf **`0`** setzen.

  Nach dem **`div`** liegt der Rest in **`edx`**, was ganz praktisch ist,
  denn der Rest ist ja das nächste **`b`**, und die Schleife erwartet **`b`** in **`edx`**.

- *Nach der Schleife* brauchst du noch ein **`mov`**,
  um den letzten Wert von **`a`** als *Returnwert* zurückzuliefern.