GTP Aufgabe2 - Euklidische Division

1.1 Hintergrund

Wenn wir einen Zähler z durch einen Nenner n dividieren, können wir das tun, indem wir den Nenner so oft vom Zähler subtrahieren, bis eine Zahl übrig bleibt, die kleiner oder gleich dem Nenner ist.
Die Zahl, die dann übrig bleibt wird Rest r genannt.
z / n = q Rest r

Ziel dieser Aufgabe ist es zwei Assembler Programme zu entwickeln, die den ganzzahligen Quotienten q und den Rest r so ermitteln, dass gilt:
n * q + r = z

Dieses Programm soll zunächst für positive z und n funktionieren.
Danach soll in einem weiteren Programm auch die Division von negativen Zahlen ermöglicht werden.

1.2 Vorüberlegungen

1.2.1 Algorithmus

Folgender Algorithmus setzt die Euklidische Division um: (z/n = q Rest r)

```
unsigned int z = 35;
unsigned int n = 3;
unsigned int q = 0;
unsigned int r = z;
while (r >= n)
{
   r=r-n;
   q++;
}
```

1.2.2 Carry und Overflow

Bildet folgende Summen und gebt an, ob der Carry C und ob der Overflow V gesetzt ist.
Dazu soll das Excel File herunter geladen werden und ausgefüllt werden.

a) 0x23 + 0xfffffffd
b) 0x02 + 0xfffffffd
c) 0x03 + 0xffffffdd
d) 0x03 + 0xfffffffe
e) 0x03 + 0xffffffd
f) 0xC0000000 + 0x80000000
g) 0x40000000 + 0x40000000

1.2.3 Sprungbefehle

Wann wird zum Schleifenende gesprungen (schlende) ?

```
schlanf     cmp  r0, r1
            bcc schlende
            ...
            b  schlanf
schlende
```

☐ r0 < r1
☐ r0 > r1
☐ r0 = r1

```
schlanf     cmp  r0, r1
            bcs schlende
            ...
            b  schlanf
schlende
```

☐ r0 < r1

☐ r0 > r1

☐ r0 = r1

1.3 Aufgabenstellung

1.3.1 Vorzeichenlos

Es ist der Algorithmus aus Abschnitt 1.2.1 in Assembler zu realisieren.

Die Werte für z , n , q , r sollen in den Registern r0 , r1 , r2 , r3 gespeichert sein.

Folgende Tests sollen durchgeführt werden:

- 71/10
- 10/71
- 0xf0000000/0xf000000

1.3.2 Vorzeichenbehaftet

Die Werte für Zähler und Nenner können nun auch negativ sein.
Um den Algorithmus trotzdem durchführen zu können, müssen die negativen Zahlen in positive gewandelt werden. Wenn das Vorzeichen geändert wurde, dann muss man sich dieses in einem Flag s merken.
Am Ende muss das Ergebnis wieder in ein negatives gewandelt werden, wenn genau ein Wert negativ war.

Der erweiterte Algorithmus dazu sieht so aus:

```
s=1;            //Flag = 1
if (z<0)       //Wenn Zähler negativ, dann Zähler positiv machen und Flag setzen
{
   z=0-z;
   s=0-s;
}

if (n<0)    //Wenn Nenner negativ, dann Nenner positiv machen und Flag setzen
{
   n=0-n;
   s=0-s;
}
.....
hier normale Berechnung aus 1.3.1
.....
if (s<0)   //Wenn Flag gesetzt, Ergebnis und Rest negativ setzen
{
   q=0-q;
   r=0-r;
}
```

Der Assembler Code soll nun laut diesem erweiterten Algorithmus umgesetzt werden

Folgende Tests sollen durchgeführt werden:

- -20 / 10
- -30 / -5
- 75 / -3

1.4 Vorzubereiten

Flags, Verzweigungsbefehle, Bedingungscodes