# CT Praktikum

# Multiplikation

## 1 Lernziele

In diesem Praktikum lernen Sie exemplarisch, wie die binäre Multiplikation realisiert wird. Sie werden dazu selbst eine Multiplikationsprozedur in Assembler schreiben.

## 2 Einleitung

Erinnern Sie sich an Ihre Primarschulzeit, wie sie damals schriftlich multipliziert haben? Prozessoren multiplizieren mit den genau gleichen Verfahren wie Sie das in der Primarschule getan haben. Aus Geschwindigkeitsgründen sind die Verfahren natürlich optimiert und an die Möglichkeiten der Digitaltechnik angepasst.

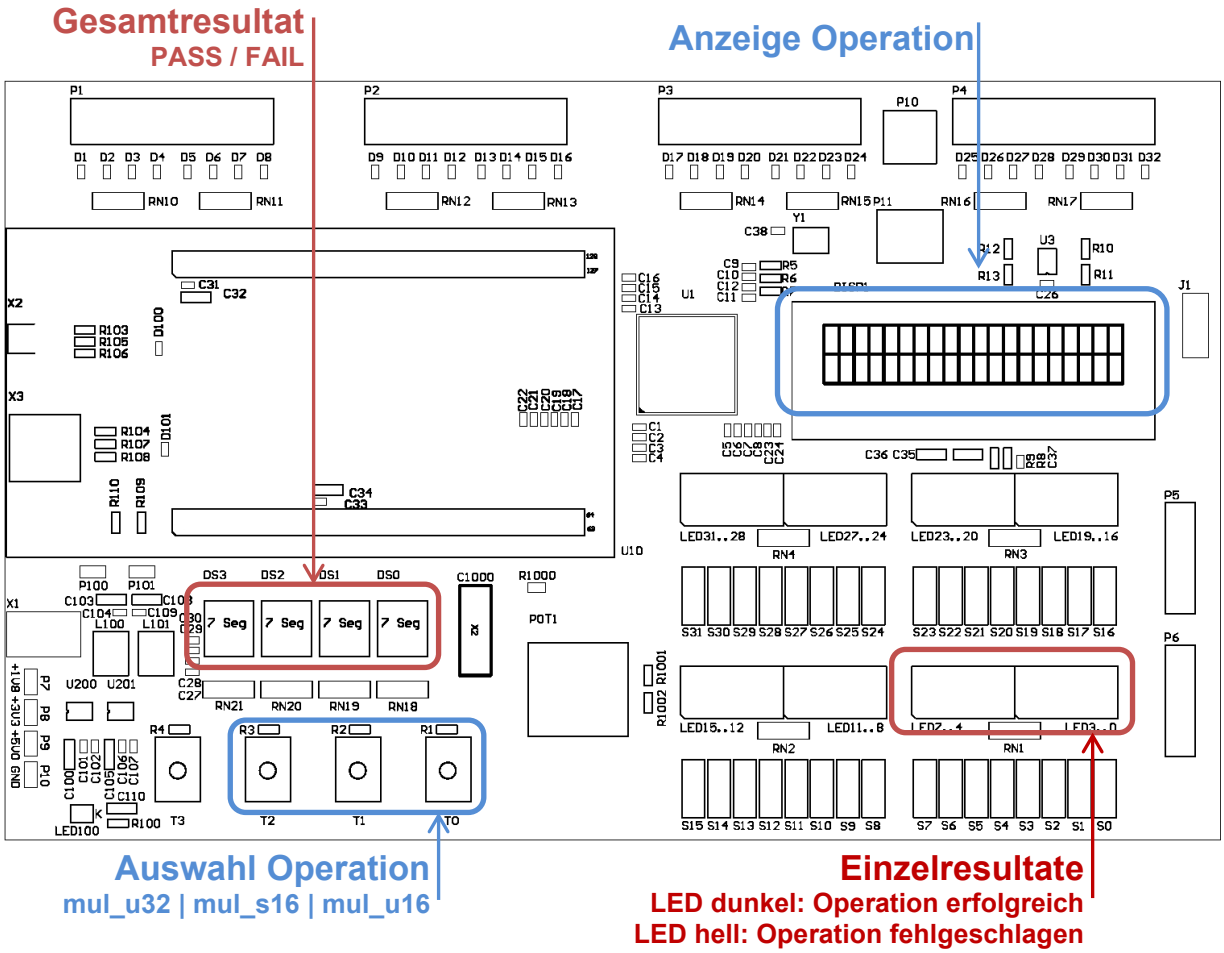


Abbildung 1: Anzeige und Auswahl der Operationen

# 3 Das Grundprinzip der schriftlichen Multiplikation

Beim binären Multiplizieren ist es wichtig, die resultierende Anzahl Stellen zu kennen. Vergessen Sie hier vor allem auch nicht die Problematik bei negativen 2er-Komplement Operanden. Es gilt: die Anzahl Stellen des Produkts zweier Zahlen ist gleich der Summe der Stellen von Multiplikator und Multiplikand.

Eine Plausibilitätserklärung: der grösste Wert einer zweistelligen Zahl ist *99*, der Maximalwert des Produktes von zwei zweistelligen Zahlen beträgt damit *p = 99 * 99 = 9801* und ist eine vierstellige Zahl. Dies lässt sich mit Probieren auf die gleiche Art und Weise für eine beliebige Anzahl von Stellen des Multiplikators und Multiplikanden nachprüfen. In der Rechnertechnik spricht man oft von einer *Verdoppelung der Anzahl Stellen*, weil Multiplikator und Multiplikand die gleiche Anzahl Stellen haben (Wortbreite des Prozessors).

Was heisst aber Multiplizieren eigentlich? Betrachten wir dazu das Beispiel *p = 3 * 12*. In Worten ausgedrückt heisst das, wir müssen die *12* drei mal addieren ... voilà die erste Methode zur Berechnung eines Produktes: den Multiplikator als Zähler verwenden und entsprechend oft den Multiplikanden aufsummieren.

Die Methode ist allerdings für grosse Zahlen nicht sehr effizient und die Rechendauer abhängig vom Wert des Multiplikators. *Schriftliche* Multiplikation hat dieses Problem nicht: die Anzahl Rechenschritte ist nur von der Anzahl Stellen des Multiplikators abhängig.

Als Beispiel betrachten wir im folgenden die Multiplikation *p = 23 * 74* (Multiplikator * Multiplikand):

```
23 * 74 = ?
-------
    222
   148
   ----
   1702
   ====
```

Die gleiche Multiplikation mit Binärzahlen ergibt:

```
10111 * 1001010 = ?
---------------
        1001010
       1001010
      1001010
     0000000
    1001010
    -----------
    11010100110
    ===========
```

Im Binärsystem ist das viel einfacher, denn es gibt nur die Ziffern 0 und 1, d.h. der Multiplikand muss nur addiert werden, wenn die entsprechende Ziffer des Multiplikators gleich 1 ist, sonst kann auf die Addition verzichtet werden.

# 4 Aufgaben

Für die Implementation in Assembler stellen wir Ihnen die Prozeduren *mul_u16.s, mul_s16.s* und *mul_u32.s* zur Verfügung. Diese enthalten einen sogenannten Unit Test[1] und einen Rahmen für die zu implementierende Funktion. Der Unit Test prüft die Funktion der Multiplikation indem er Operanden aus Tabellen ausliest, die Multiplikation durchführt und die Resultate mit Referenzwerten vergleicht. Bei korrekter Ausführung aller Multiplikationen erscheint ein „*PASS*“ auf der 7-Segmentanzeige; andernfalls ein „*FAIL*“.

Die gewünschte Operation kann im Hauptprogramm (*main.s*) mittels der Tasten T0 bis T2 ausgewählt werden. Auf Tastendruck wird die entsprechende Operation ausgeführt.

Das LCD zeigt die zuletzt ausgeführte Operation an. Die einzelnen LEDs zeigen an, welche Teiloperationen fehlgeschlagen sind: LED0 informiert über die erste Teiloperation, LED1 über die zweite, etc. (Siehe Abbildung 1).

## 4.1 Aufgabe: 16 Bit x 16 Bit unsigned Multiplikation

Realisieren Sie eine Multiplikation mit *unsigned* 16 Bit Operanden. Der Multiplikator wird im Register R0 übergeben, der Multiplikand im Register R1. Das Resultat soll im Register R0 zurückgegeben werden.

a) Machen Sie sich mit dem gegebenen Unit Test vertraut. Was macht der Test?

b) Führen Sie das Programm aus. Da die entsprechende Unterfunktion (Label *„operation“*) noch nicht fertig ausprogrammiert ist erscheint ein *„FAIL“* auf der 7-Segmentanzeige. Ausserdem werden auf den LEDs 0 bis 7 diejenigen Rechnungen angezeigt, welche nicht korrekt waren.

c) Setzen sie die Assembler Instruktion *MULS* in die Unterfunktion ein: Auf der 7-Segmentanzeige sollte nun ein *„PASS“* erscheinen und die LEDs 0 bis 7 sollten alle dunkel sein.

d) Implementieren Sie die Multiplikation zuerst auf Papier (z.B. ein Struktogramm) und diskutieren Sie die Lösung untereinander bevor Sie mit dem Programmieren beginnen.

e) Fügen Sie Ihren Code am markierten Ort in der Prozedur *mul_u16.s* ein und testen Sie Ihre Implementation.

- Falls nicht bereits geschehen, wird die genaue Funktion eines Prozeduraufrufes in Kürze in der Vorlesung behandelt.

[1] *“In computer programming, unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual function or procedure. In object-oriented programming a unit is usually an interface, such as a class.”* Aus Wikipedia “unit testing”.

### 4.2 Aufgabe: 16 Bit x 16 Bit signed Multiplikation

In diesem Schritt soll die Multiplikationsroutine aus 4.1 nun auf *signed* 16 Bit Operanden umgeschrieben werden.

Fügen Sie Ihren Code am markierten Ort in der Prozedur *mul_s16.s* ein und testen Sie Ihre Implementation.

- Die Sign Extension auf dem Multiplikanden kann vor der Speicherung in R0 / R1 mit dem Befehl `SXTH` gemacht werden.
- Bei der *signed* Operation werden 32 Schleifendurchläufe benötigt.

### 4.3 Aufgabe: 32 Bit x 32 Bit unsigned Multiplikation

Realisieren Sie nun die Multiplikation mit *unsigned* 32 Bit Operanden. Der Multiplikator wird im Register R0 übergeben, der Multiplikand im Register R1. Das Resultat soll in den Registern R0 (lower word) / R1 (upper word) zurückgegeben werden.

Fügen Sie Ihren Code am markierten Ort in der Prozedur *mul_u32.s* ein und testen Sie Ihre Implementation.

## 5 Bewertung

Die lauffähigen Programme müssen präsentiert werden. Die einzelnen Studierenden müssen die Lösung/Quellcode verstanden haben und erklären können.

| Kapitel / Aufgabe | Bewertungskriterien | Gewich-tung |
|---|---|---|
| 4.1 | Das Programm erfüllt die in Aufgabe 4.1 geforderte Funktionalität. | 2/4 |
| 4.2 | Das Programm erfüllt die in Aufgabe 4.2 geforderte Funktionalität. | 1/4 |
| 4.3 | Das Programm erfüllt die in Aufgabe 4.3 geforderte Funktionalität. | 1/4 |