Der STACK Teil 3 - Unterprogramme mit variabler Parameteranzahl

Der STACK sollte Lesern meiner bisherigen 2 Beiträge eigentlich kein "Buch mit sieben Siegeln" mehr sein. Im vorläufig letzten Teil zum Thema STACK möchte ich abschließend noch auf das Thema "Unterprogramme mit variabler Parameteranzahl" eingehen. Dieses Thema ist sehr komplex und für jedes Problem mag es diverse praktikable Lösungen geben. Sie unterscheiden sich aber meist nur in Details. Wie zuvor ist dieser Beitrag somit auch nur als Anregung zu sehen.

Was ist eine "variable Parameteranzahl" ? Stellen wir uns ein Programm vor, bei dem ein Benutzer über Tastatur eine Zahl mit 1 bis 5 Stellen eingeben kann. Das Ende seiner Eingabe signalisiert er dem Programm durch drücken der Taste "Enter", eine fehlerhafte Stelleneingabe kann er mit der Taste "BackSpace" löschen. Somit ein Alltagsproblem.

Leider, aus Sicht des Prozessor, gibt der Mensch aber in seinem gewohnten Dezimalsystem ein, nicht binär, wie es der µP gerne hätte. Also muss der Programmierer für eine Umwandlung der Eingabe sorgen. Gut, die Problemstellung dürfte klar sein.

Was bekommt der Programmierer vom Anwender geliefert: Eine Anzahl von Tastendrücken. Jede "Zahlentaste" bedeutet eine Ziffer der Dezimaldarstellung. 0 bis 9, ist klar. "BackSpace" bedeutet: "Lösche letzte Zifferneingabe"; auch klar. Und mit "Enter" sagt der Anwender letztlich "Fertig mit der Eingabe". Logo.

Ich gehe davon aus, dass die Behandlung der Tastaturabfrage erschlagen wurde. Auch das Benutzerecho auf einem Display ist fertig programmiert und läuft. Was fehlt ist noch die Speicherung der Eingaben und die Umwandlung in eine Binärzahl für den µP. Die Eingabe-kette brauche ich nicht speichern. Ich kann sie ja jederzeit aus ihrem Binäräquivalent rekonstruieren.

Ich setze wie folgt an:
Jeder Tastendruck einer Zifferntaste wird im Zähler "Stellen" gezählt. Jede "BackSpace" vermindert Zähler "Stellen" wenn möglich. Wie üblich wird die Eingabe der Zahl mit ihrem höchsten Stellenwert begonnen. Also "1", "2", "3" für "Hundertdreiundzwanzig". "Enter" beendet die Eingaberoutine und startet die Umrechnung. Somit anwendergerecht.

Ich würde jetzt z.B. so vorgehen:
- Bei jedem Tastendruck einer "Zahlentaste" schreibe ich deren BCD-Wert auf den STACK und erhöhe den Zähler "Stellen"
- Bei "BackSpace", wenn möglich, hole ich das letzte Zeichen vom STACK und vermindere "Stellen".
- Bei "Enter" schreibe ich noch die Anzahl der Zeichen aus "Stellen" auf den STACK und starte die Konvertiertung.

Ob der Anwender jetzt 1 oder 5 Stellen eingegeben hat, ist Nebensache. Dieses hat dem Unterprogramm egal zu sein. Heisst: Anzahl Parameter ist variabel.

Das erste was das UP (Unterprogramm) auf dem STACK findet ist natürlich die Rück-kehradresse ins Hauptprogramm. Danach findet es die Anzahl der Tastendücke des Anwenders. Darauf folgen dann die einzelnen Dezimalstellen. Sinnigerweise zuerst die Einerstelle, dann die Zehner usw.
Gut. Die Anzahl Tasten hat jeder schnell als Schleifenzähler interpretiert. Aber wie krieg ich die ?

Aus Teil 2 kann ich doch ableiten: Meine "Anzahl Eingabestellen" liegt drei Bytes hinter der aktuellen Position von SPL. Richtig ? Bedenke: Der Stack wächst von oben nach unten! Also müsste doch die Kopie des aktuellen SPL in einem Indexregister plus 3 auf dieses Byte zeigen. Wenn dies stimmt, kann ich diesen Wert doch leicht in ein Arbeitsregister kopieren. LD Rxy, Indexregister

OK, jetzt weiss mein UP wieviel Stellen es verarbeiten muss.
Jetzt muss ich mir Gedanken machen, wie ich diese verarbeite. Ich muss also einen "Algorythmus", eine Verfahrensvorschrift, definieren. Dies ist meist der schwierigste Teil der Materie. Oft hat man hier das sprichwörtliche "Brett vor'm Kopf", da man etwas ähnliches schon mal "irgendwie anders" erschlagen hat. Daher fallen die folgenden Ansätze etwas langatmig aus. Selbst die "beste" Lösung kann optimiert werden.

Ansatz 1:
- Ich Lösche das (die) Ergebnissregister.
- Ich fange mit der Einerstelle an. Diese brauche ich nur im Ergebniss speichern.
- Die zweite Stelle, die Zehnerstelle, müsste ich mit 10 multiplizieren und auf das Ergebniss addieren.
- Die dritte Stelle, die Zehnerstelle, müsste ich mit 100 multiplizieren und auf das Ergebniss addieren.
- ...
- Die n-te Stelle, die Zehnerstelle, müsste ich mit 10^(n-1) multiplizieren und auf das Ergebniss addieren.
- Am Schleifenende müsste also das Ergebniss im (in den) Ergebnissregister(n) stehen.

Das sieht alles irgendwie nicht sehr einfach aus. Ich muss für jedes Zeichen einen Multiplikator berechenen. Dann addieren. Das über alle Eingabezeichen. Hmm. Geht's vielleicht auch anders ?

Ansatz 2:
- Ich Lösche das (die) Ergebnissregister.
- Ich fange mit der letzten Eingabestelle an. Diese brauche ich nur im Ergebniss speichern.
- Vor der vorletzten Stelle müsste ich das Ergebniss mit 10 multiplizieren und darauf die vorletzte Stelle addieren.
- Vor der drittletzten Stelle müsste ich das Ergebniss mit 10 multiplizieren und darauf die drittletzte Stelle addieren.
- ...
- Vor der letzten Stelle müsste ich das Ergebniss mit 10 multiplizieren und darauf die letzte Stelle addieren.
- Am Schleifenende müsste also das Ergebniss im (in den) Ergebnissregister(n) stehen.

Das scheint schon besser für eine Schleife geeignet. Ich brauche keine Multiplikatoren mehr berechenen. Bis auf Schritt 2 nur addieren. Das über alle Eingabezeichen. Geht's vielleicht noch optimaler ?

Ansatz 3:
- Ich Lösche das (die) Ergebnissregister.
- Vor der letzten Stelle müsste ich das Ergebniss mit 10 multiplizieren und darauf die letzte Stelle addieren. Das macht nix, denn das Ergebniss der Multiplikation ist eh NULL.
- Vor der vorletzten Stelle müsste ich das Ergebniss mit 10 multiplizieren und darauf die vorletzte Stelle addieren.
- Vor der drittletzten Stelle müsste ich das Ergebniss mit 10 multiplizieren und darauf die drittletzte Stelle addieren.
- ...
- Vor der letzten Stelle müsste ich das Ergebniss mit 10 multiplizieren und darauf die letzte Stelle addieren.
- Am Schleifenende müsste also das Ergebniss im (in den) Ergebnissregister(n) stehen.

Das scheint schon optimal für eine Schleife geeignet. Ich brauche keine Multiplikatoren mehr berechenen. Nur mit 10 multiplizieren und addieren. Das über alle Eingabezeichen.

Ansatz 3 scheint somit der optimalere Ansatz zu sein. Nur, wie komme ich jetzt an die erste vom Anwender eingegebene Stelle ?

Nun, schauen wir uns den Stack mal am Beispiel Eingabe "123" an:

Adr.	Inhalt	Bemerk
SPL	frei	nächstes freies Byte im STACK
SPL+1	RetL	Rückkehradresse Hauptprogramm Low
SPL+2	RetH	Rückkehradresse Hauptprogramm High
SPL+3	3	Anzahl Eingabezeichen -> Kopie der Adresse ist in Indexregister
SPL+4	BCD "3"	Letzte Eingabe Einerstelle für 3
SPL+5	BCD "2"	Vorletzte Eingabe Zehnerstelle für 20
SPL+6	BCD "1"	Erste Eingabe Hunderterstelle für 100
SPL+7	unknown	Irgendwas anderes auf STACK

Nun. In meinem Indexregister Z steht eine Kopie von SPL + 3. Sie war ja notwendig um die Anzahl der Eingabezeichen (=3) als Schleifenzähler in ein Arbeitsregister zu kopieren.
Wenn ich darauf jetzt die Anzahl Zeichen addiere sollte er auf die erste Eingabestelle zeigen. Mathematisch SPL + 3 + Anzahl Zeichen = SPL+3+3 = SPL+6. Das scheint richtig zu sein.
Eine Schleife der Form (VORSICHT: Der Code ist idealisiert und so nicht lauffähig !)

```
...
        ;       Z = Indexregister
        MOV     Z, SPL                  ; SPL kopieren
        ADD     Z, 3                    ; SPL korrigieren auf "Anzahl"
        LD      Schleifenzähler, Z      ; Anzahl zeichen merken für Schleife
        EOR     Ergebnis, Ergebis       ; Lösche Ergebnisregister
        ADD     Z, Schleifenzähler      ; Zeiger auf erstes Eingabezeichen
Schleife:
        CALL    16BitMUL10dez           ; Multipliziere Ergebnis mit 10 dezimal
        LD      Eingabe, Z              ; hole Eingabeziffer vom STACK
        DEC     Z                       ; Zeiger decrementieren
        ADD     Ergebnis, Eingabe       ; Addiere BCD-Wert
        DEC     Schleifenzähler         ; Schleifenzähler bearbeiten
        BRNE    Schleife
```

sollte die Berechnung des Binärwertes erschlagen.

Anzumerken ist hier folgendes:
Sowohl Z als auch Ergebnis sind Register-Worte, also jeweils zwei Byte. AVR kennt jedoch keinen Befehl DECW für "Dekrement Wort" und kein ADDW für "Addiere Register auf Wort". Der Übersichtlichkeit / Lesbarkeit halber habe ich hier die entsprechenden Registerbefehle verwendet. Mitdenken ist also angebracht, wenn man ähnliches im eigenen Programm realisieren möchte. Ich verstehe meine Beiträge grundsätzlich NICHT als Kochrezepte. Auch "16BitMUL10dez" deutet nur die Multiplikation mit 10 an, ohne auf Details wie Parameterübergabe ein zu gehen. Natürlich kann diese problemlos über den STACK erfolgen.

Jetzt fehlt noch die Rückgabe des Ergebnisses an's Hauptprogramm. Auch das erschlage ich über den STACK. Wohin zeigt meine veränderte Kopie des STACK-Pointer jetzt ? Auf das Byte vor der Einerstelle, also auf "Anzahl Zeichen". Hier könnte ich jetzt problemlos das High-Byte des Ergebnisses reinschreiben, gefolgt vom Low-Byte über die Einerstelle. Gut, machbar. Aber im Hauütprogramm habe ich vielleicht schon vergessen wieviele Parameter ich übergeben habe und somit wieder vom STACK runter holen muss. Also würde ich diese Information ungerne verlieren. Also sollte ich das Ergebnis auf der Einer- und Zehnerstelle übergeben. Auch das geht recht einfach:

```
        INC     Z
        ST      Z,      ErgebnisHigh
        INC     Z
        ST      Z,      ErgebnisLow
```

bzw. kürzer

```
        ST      +Z,     ErgebnisHigh
        ST      +Z,     ErgebnisLow
```

Zu guter letzt bleibt noch im Hauptprogramm das Ergebnis zu holen und den STACK zu bereinigen. Der folgende Codeabriss sollte soweit selbst erklärend sein, dass sich eine Beschreibung erübrigt.

```
        POP     Anzahl
        POP     ErgebnisHigh
        POP     ErgebnisLow
        SUBI    Anzahl, 2
ClearStack:
        POP     Temp
        DEC     Anzahl
        BRNE    ClearStack
```

Die obige Form des Unterprogrammaufrufes mit variabler Parameteranzahl findet man in Büchern oft speziell auf die Sprache C bezogen als

```
        UnterprogrammName     Agr_C, Arg_Vekt
```

beschrieben. Arg_C steht für "Argument Count" also "Anzahl Parameter"; Arg_Vekt steht für "Argument Vektor" also etwa "Parameterkette". Genau genommen ist Unser STACK eine solche Parameterkette oder Vektor. Nur gibt es für den STACK keinen direkten Zähler "Anzahl Werte". Dieser ist aber über RAMEND und SPL leicht zu berechnen. SPL stellt einen Zeiger auf das nächste freie Byte des Vektor STACK dar. Siehe später.
C ist in seiner inneren Struktur sehr nah an Assembler ausgerichtet. Die Idee einer derartigen Parameterübergabe stammt aber aus Assembler. C wurde erst später "darüber gelegt"; was einige Eigenarten von C erklärt. Wie die Verarbeitung von Ausdrücken von rechts nach links mit ihren anfänglich meist nervenden Seiteneffekten. Der Mensch liest und denkt halt von links nach rechts.

Verschweigen möchte ich auch nicht, dass variable Parameterzahl auch ohne Zähler "Anzahl Parameter" möglich ist. Es muss nur ein eindeutiges "Endekennzeichen" definiert sein. Viele Dateien auf einem Speichermedium wie Festplatte verwenden auch heute noch "Control-Z" als Endekennzeichen. Wer mal im DOS-Modus eingibt "TYPE Programmname.EXE" kann dies leicht erkennen. Auch wenn die .EXE etliche MB gross ist, bricht die Ausgabe meist nach ein paar Zeilen ab. TYPE interpretiert Control-Z als Endekennzeichen.

Eine weitere Form dieser Art ist die Darstellung von Strings (ASCII-Zeichenketten) in Assembler und C. Beide kennen einen derartigen String-Variablentyp direkt nicht. Sie verwenden Ketten von Bytes CHAR(). Ihr Endekennzeichen ist 0x00. Auch hier kann auf die Übergabe einer Anzahl verzichtet werden. Man muss halt die Kette der Parameter nur bis zum Endekennzeichen bearbeiten. Eine derartige Kette nennt man deshalb auch ASCIIZ-Kette. Für ASCII-Zero, also mit Null abgeschlossene Kette.

Zuletzt sei noch erwähnt: Man muss nicht die gesamte Kette übergeben. Es reicht, wenn man einen Parameter übergibt der sagt, wo die Kette steht. Genauso wie SPL sagt, wo das nächste freie Byte in STACK steht. Sowas nennt mann Pointer oder Zeiger. Diese Technik kann man in fast beliebige Tiefen treiben. Ein Zeiger auf eine Liste von Zeigern die wiederum jeweils auf eine Liste von Zeigern zeigen. C unterstützt diese Technik hervorragend und sie war wohl auch entscheidend für den enormen Erfolg von C. Natürlich kann auch die Rückgabe des UP auf diese weise erfolgen. "Ihre Antwort finden sie bei 0x1234."

Womit wir eigendlich beim 4.ten Teil meines Beitrages zum Thema STACK wären.

Manipulationen am STACK.

Wie erspare ich mir mühsame STACK-Bereinigungen. Wie gebe ich mehr Parameter über den STACK zurück, wie übergeben wurden. Oder wie kehre ich im Hauptprogramm an einen völlig anderen Codeabschnitt zurück. Zugegeben ein sehr interessantes Kapitel zur Operation am offenen Herzen eines Systems.
Jedoch bin ich der Meinung, wer die bisherigen 3 Teile zum STACK gelesen und verstanden hat sollte in der Lage sein sich dieses Kapitel selbst zu erarbeiten. Ich lasse mich aber vielleicht noch überreden. Vorläufig arbeite ich aber nicht an einem Teil 4.