Der STACK Teil 2 - Parameterübergabe an Unterprogramme
==========

Wer den ersten Teil, "Der STACK - Funktion und Nutzen", gelesen hat kennt sich mit diesem Ding schon recht gut aus. Hier möchte ich einen Weg aufzeigen, Unterprogramme recht flexibel in ihrer Verwendung zu machen.

Wer schon etwas mit Hochsprachen wie C, Pascal oder Basic gearbeitet hat, kennt von dort die nützliche Möglichkeit Unterprogramme und Funktionen mit Übergabeparametern zu versorgen. In Assembler scheint es diesen Weg nur über Register zu geben. Das schränkt einen zumal recht ein. Aber es geht auch anders. Wie ? Nun, über den gleichen Weg wie Hochsprachen dies erschlagen: Parameterübergabe über den STACK.

Schauen wir uns zunächt die übliche Registerübergabe mal an:

```
MAIN:
        LDI R16, $10
        LDI R17, $20
        CALL Addiere
        ...
END

Addiere:
        ADD R16, R17
RET
```

Im Grunde ist dieses UP schon eine Funktion, denn es liefert in R16 einen Wert ans Hauptprogramm zurück. Aber dieser feine Unterschied zwischen "Subroutine" und "Function" ist in heutigen Hochsprachen schon fast nicht mehr gegeben. Aber das wäre ein anderer Beitrag.

Es gibt nur ein kleines Manko bei diesem Beispiel. Egal wo im Programm oder in welchem Programm. Immer muss ich R16 und R17 nutzen. Habe ich Werte in anderen Registern muss ich diese erst dorthin MOVen. Bei kleinen Problemen sicher ein gangbarer Weg. Aber was ist, wenn ich mal 20 oder 30 Werte übergeben möchte und nicht soviele Register frei habe ? Möglich wäre, alles ins SRAM zu schieben und das UP darüber laufen lassen.

Genau DAS macht eine "Parameterübergabe über STACK". Alle Werte auf den STACK PUSHen, den CALL absetzen und dann aber wieder den STACK frei POPen. Ich beschäftige mich hier erst mal mit fester, kleiner Parameterzahl. Im dritten Teil gehe ich dann auf viele Parameter ein. Wie sähe nun ein solcher Aufruf für mein Beispiel aus ?

```
MAIN:
        LDI Temp, $10
        PUSH Temp
        LDI Temp, $20
        PUSH Temp
        CALL Addiere
        POP Ergebnis
        POP Temp
        ...
```

```
        ; oder
        PUSH R25        ; mal irgendwelche beliebige
        PUSH R3         ; Register .....
        Call Addiere
        POP Ergebnis
        POP Temp
        ...
END
```

Sieht ja schon irgendwie recht universell aus. Aber wie jetzt weiter ? Im UP weiss ich, die Parameter liegen auf dem STACK. Aber wo, und wie komme ich da ran ?

Fragen wir uns deshalb erst mal was im Hauptprogramm passiert. Gesetzt den Fall der STACK wäre leer und wir hätten 128 Byte SRAM. Es wäre also RAMEND = $DF definiert. Das wären dezimal 223. Das ist richtig, da das SRAM bei $60 = 96 dezimal beginnt und Bestandteil des DATA-Memory (R0 bis R31, Statusregister und SRAM) ist. Der Stackpointer SPL wäre somit mit 223 besetzt.

Mit den beiden PUSH Temp würde er auf 221 herunter gezählt. Das ist auch noch klar. Meine Parameter liegen also bei SRAM(223) und SRAM(222). Jetzt kommt der CALL Addiere. Er schreibt die Rückkehradresse in MAIN auf den STACK und vermindert SPL auf 219. Mein UP Addiere wird aktiv und sähe eben diese 219 auf SPL. Auch wenn zwischenzeitlich ein oder mehrere Interrupts aufgetreten wären, wäre der STACK wieder bereinigt. Müsste auch klar sein.

Im UP merke ich mit zuerst einmal SPL. Sinnvollerweise in einem Indexregister wie ZLow.
ZLow ist somit zu einem Zeiger auf das nächste freie Byte im STACK geworden. Was jetzt mit SPL passiert, kann mir egal sein. Prima, aber was nützt das ? Nun, ich weiss ja, dass 4 Byte davor bzw. dahinter, je nach Betrachtungsweise, mein erster Parameter liegt. 219 + 4 = 223. Also korrigiere ich ZLow ADD ZLow, 4. Ich habe somit einen Pointer (Zeiger) auf meinen ersten Parameter. Was nun?

Also zuerst brauche ich mal 2 Arbeitsregister. Ich nehme R20 und R21. Einfach so, ohne mich um irgendwas zu kümmern. Die Namen gefallen mir einfach. Nein, ich kümmere mich schon. Ich schiebe sie auf den STACK um ihren Inhalt nicht zu verlieren.

```
PUSH   R20
PUSH   R21
```

Dann hole ich mir meinen ersten Parameter nach R20, den Zweiten nach R21. Ich nutze dazu die Indirekte Adressierung.

```
LD      R20, Z
LD      R21, -Z
```

R21 lade ich mit PRE-DECREMENT um den Zeiger ZLow zu korrigieren.
Dann rechne ich und schreibe das Ergebnis zurück auf den STACK zurück.

```
ADD     R20, R21
ST      Z, R20
```

Hole die ursprünglichen Werte von R20 und R21 zurück und RET.

Das ganze sieht dann so aus:

```
Addiere:
        MOV    ZLow, SPL
        ADD    ZLow, 4
        PUSH   R20
```

```
        PUSH   R21
        LD     R20, Z
        LD     R21, -Z
        ADD    R20, R21
        ST     Z, R20
        POP    R21
        POP    R20
RET
```

Eigentlich schon fast perfekt. Nur ein kleiner Schönheitsfehler ist drin. Ich nutze Z ungefragt ohne es zu sichern. Ich wollte aber erst mal keine zusätzliche Verwirrung zur Korrektur der SPL-Kopie in ZLow schaffen. Richtiger sähe es so:

Addiere:

```
        PUSH   ZHigh
        PUSH   ZLow
        MOV    ZLow, SPL
        ADD    ZLow, 6          ;  wegen PUSH ZHigh / -Low
        PUSH   R20
        ...
        POP    R20
        POP    ZLow
        POP    ZHigh
RET
```

Eine nach diesem Schema gestrickte Routine kann man in jedes Programm problemlos einbinden. Man muss nur wissen was und in welcher Folge die Parameter vorher gePUSHt und nachher gePOPt werden. In der Routine verwendete Register interessieren nicht. Diese sorgt schon selbst dafür, dass keine Register ungewünscht verändert werden.