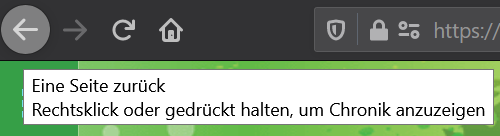
## Der Stack (Stapel)

Beim Stack handelt es um einen Speicherbereich im RAM, in dem Werte zwischengespeichert werden können. Von der Funktion her handelt es sich um einen LiFo-Speicher=Lastin-Firstout, d.h. die Werte, die zuletzt abgespeichert wurden, werden als erstes wieder ausgelesen. Man kann sich das, und darauf deutet der Name auch hin, wie einen Papierstapel vorstellen auf dem ständig was abgelegt wird und dann von oben beginnend wieder weggenommen wird.

Ein Beispiel für die konkrete Anwendung ist die Zurück-Funktion im Browser.

Der Stack wird von der CPU des uC immer dann benutzt wenn in ein Unterprogramm (UP) oder eine Interrupt-Routine (IR) gesprungen wird. Dann speichert die CPU den aktuellen Inhalt des Programmzählers auf dem Stack.

Da der Programmzähler auf die Adresse des nächsten auszuführenden Befehl zeigt, ist dies also die Stelle, an der nach der Ausführung des UP/IR im Hauptprogramm (HP) weitergemacht werden soll.

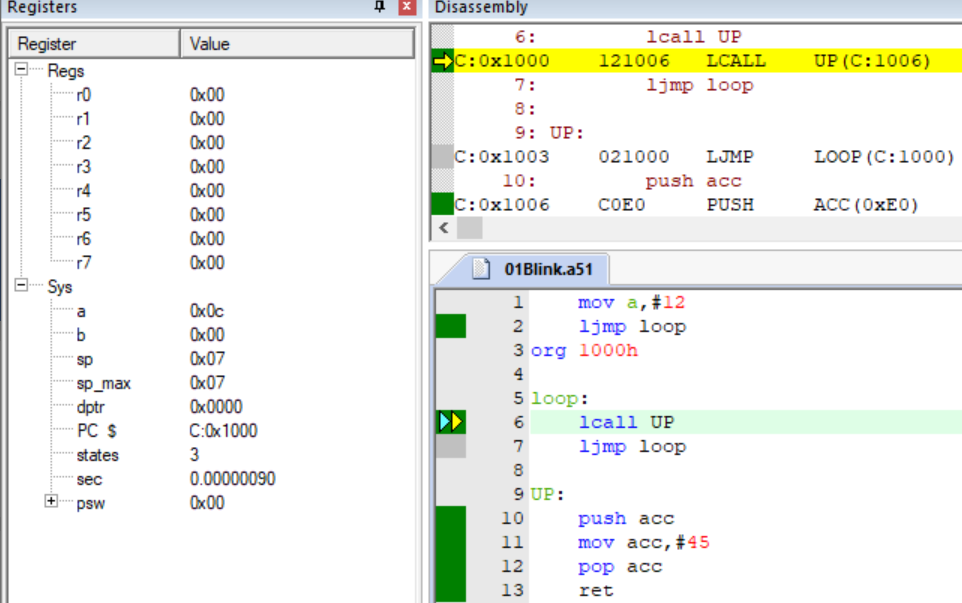
Eine weitere Anwendung des Stacks ist das Zwischenspeichern und Rücklesen von Werten mit den Befehlen push und pop. So kann mit push acc Dies muss man z. B. mit dem Akku und Registern tun, wenn eine IR ausgeführt wird und in dieser ebenfalls der Akku oder die gleichen Register verwendet werden. Andernfalls sind die im UP veränderten Werte auch für das HP „gültig“.

### Stackpointer (SP, 1Byte)

Da auf den Stack lesend und schreibend zugegriffen wird muss es sich um einen Bereich im RAM handeln. Der Beginn liegt an der Adresse 7h, zeigt in aufsteigender Richtung und wird vom Stackpointer angezeigt. Bei dem handelt es sich um ein SFR (81h). Der Resetwert 07h kann einfach verändert werden (mov SP,#xx). Dies kann erforderlich sein, wenn ein tief verschachteltes Programm ausgeführt wird. Also wenn aus einem UP wieder ein UP usw. aufgerufen wird.

### Abläufe beim Aufruf eines UP/einer IR

Betrachten wir die Abläufe bei der Ausführung des folgenden einfachen Programms.

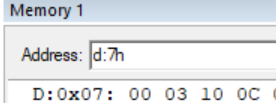


Wir haben folgende Adressverteilung im ROM:

* Der erste Befehl ist ein mov a-Befehl der 2 Byte lang ist und an der Adresse 0000h-0001h steht.
* Der ljmp-Befehl benötigt 3 Byte an der Adresse 0000h-0002h.
* Mit der org-Anweisung wurde der Beginn des HP an die Adresse 1000h verschoben.
* Somit beginnt das UP mit dem push acc-Befehl an der Adresse 1000h.
* Der ljmp-Befehl im HP (3 Byte) steht an der Adresse 1003-1005.
* Das UPbeginnt an der Adresse 1006h

Die Adressen sind absichtlich mit 4-Hex-Stellen angegeben, da dies der Länge des PC entspricht.

Ablauf:

* Nach dem Beschreiben des Akkus wird an die Adresse 1000h gesprungen (PC=1000h).
* Bei der Ausführung des lcall-Befehls wird zunächst der SP um eins erhöht SP=SP+1=8h.
* Dann wird der hintere Teil (LSB) der Adress 1003h, also 03h an der RAM-Adresse 8h abgespeichert.   
  Der Wert 1003h ist ja die Adresse, an der der ljmp-Befehl des HP steht.
* Der SP wird nochmals um 1 erhöht SP=9hDer zweite Teil der Adresse (MSB) 1003h, also 10h wird an der RAM-Adresse 09h abgespeichert.
* Im UP wird nunmit push zunächst der Wert des Akkus(=12=0Ch) an die Adresse 0ah geschrieben. Der SP wird um 1 erhöht SP=SP+1=0ah
* Der Akku wird mit dem Wert 45 überchrieben
* Beim Ausführen des pop- und ret-Befehls sind die Abläufe genau umgekehrt:
  + Wert 0Ch vom Stack in den Akkus schreiben  
    0Ch🡪acc
  + SP=SP-1=09h
  + Wert im RAM der durch SP adressiert wird in PC (MSB) schreiben:   
    MSB von PC =10h
  + SP=SP-1=08h
  + Wert im RAM der durch SP adressiert wird in PC(LSB) schreiben:  
    LSB von PC=03h
  + SP=SP-1=07h (Anfangswert)
* Ausführen des ljmp-Befehls.

Bei der Interrutpausführung sind die Abläufe enstpechend.

### Überlauf des Stacks

Mit jedem call-Befehl, genauso wie mit jeder I-Ausführung erhöht sich der SP um 2. Wirft man einen Blick auf die RAM-Belegung des Controllers, sieht man, dass oberhalb von 07h, an der Adresse 20h der bitadressierbare Bereich beginnt. Es liegen somit 20h-08h=18h=24 Adressen zwischen 07h und 20h und somit ist Platz für 12 Rücksprungadressen frei. Bei der 13. Adresse wird zwangsläufig in den bitadressierbaren Bereich geschrieben. Dies muss der Programmierer beachten und im Zweifelsfall den Stack verschieben (s.o. mov SP..).

Zu einem Überlauf des Stacks kann es kommen, wenn der SP den Wert von 254 erreicht hat und ein weiteres UP aufgerufen wird. Dann ergibt sich der Wert SP=0, was bedeutet, dass an die Adresse 0 im RAM geschrieben wird. Dies ist aber die Adresse von R0, der somit unabsichtlich überschrieben wird, was zu unerwünschten Folgen führen wird.

Das gleiche passiert zwangsläufig immer wenn die Befehle call- und ret sowie push und pop nicht paarweise verwendet werden. Wenn also im HP mit call ein UP aufgerufen, aber aus dem UP mit jmp wieder ins HP verzweigt wird. Hier hat sich der SP nach jedem Durchlauf um 2 erhöht. Hierbei kommt es **immer** zum Überlauf bzw. es wird sogar ins das komplette RAM geschrieben.

**Achtung**: call/ret und push/pop stets paarweise verwendet!