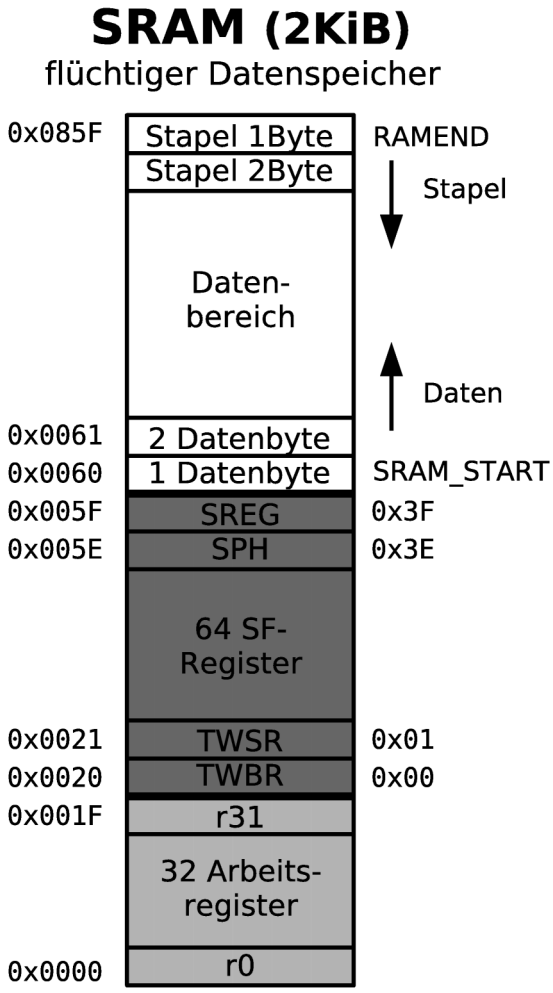
B1 Stapelspeicher (stack)

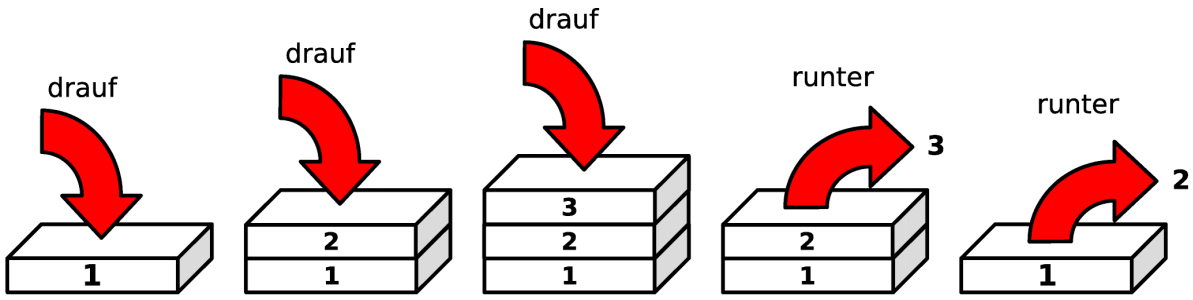
Arbeitsweise des LIFO-Stapelspeichers

Im Kapitel "Unterprogramme" wurde schon erwähnt, dass Unterprogramme einen so genannten Stapelspeicher (Kellerspeicher, **Stapel,** *stack*) benötigen um ihre Rücksprungadresse abzuspeichern. Dies gilt ebenfalls für Interrupt-Routinen. Auf den Stapel können auch Variablen zwischengespeichert werden um sie vor Veränderung zu schützen oder zeitweise aufzubewahren.



Der Stapel ist ein frei definierbarer Speicherbereich im SRAM. Seine maximale Größe ist von der Größe des SRAM abhängig. Das Abspeichern der Variablen passiert in Form eines Stapels. Zuletzt Abgespeichertes wird als erstes wieder ausgelesen, genau so wie man beim Papierstapel das letzte Blatt Papier als Erstes wieder herunter nimmt. Aus dieser Eigenschaft entstand auch die Bezeichnung **LIFO (Last In First Out)**. Die zuletzt abgespeicherte Variable wird dem Stapelspeicher also als Erstes wieder entnommen.

LIFO: Last in First out



Der Stapelzeiger SP

Der Stapelspeicher wird über den **Stapelzeiger (*stack pointer*, SP)** angesteuert. Es handelt sich hierbei um ein SF-Register-Doppelregister (Sonderfunktions-Register), welches als Adresszeiger fungiert (vergleiche indirekte Adressierung). Das Doppelregister besteht aus den beiden SF-Registern **SPL** und **SPH.**

Die SF-Register SPL und SPH

SPL-Register: SF-Register-Adresse **0x3D** (SRAM-Adresse **0x005D**)
Befehle: **in**, **out** (**sbi**, **cbi**, **sbic**, **sbis** nicht da Adresse > 32 (0x1F)!)

SPL = Stack Pointer Low

Bit	7	6	5	4	3	2	1	0
SPL 0x3D	**SPL7**	**SPL6**	**SPL5**	**SPL4**	**SPL3**	**SPL2**	**SPL1**	**SPL0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPH-Register: SF-Register-Adresse **0x3E** (SRAM-Adresse **0x005E**)
Befehle: **in**, **out** (**sbi**, **cbi**, **sbic**, **sbis** nicht da Adresse > 32 (0x1F)!)

SPH = Stack Pointer High

Bit	7	6	5	4	3	2	1	0
SPH 0x3E	**SPH7**	**SPH6**	**SPH5**	**SPH4**	**SPH3**	**SPH2**	**SPH1**	**SPH0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Die Initialisierung des Stapelspeichers

Zur Initialisierung des Stapelspeichers muss der Stapelzeiger **SP** am Anfang des Hauptprogramms mit einer Grundadresse geladen werden. Fehlt das Initialisieren des Stapelzeigers mit einer Grundadresse, so wird bei der Benutzung des Stapelspeichers der Inhalt an einer beliebigen Adresse im SRAM abgelegt, und kann dadurch Daten, die SF-Register oder die Arbeitsregister überschreiben (eventuell Absturz des Programms).

Der Stapel muss zwingend initialisiert werden bei der Benutzung von

1. **Unterprogrammen,**
2. **Interrupt-Routinen und**
3. **bei der Benutzung des Stapels als Zwischenspeicher mittels "push"- und "pop"-Befehlen.**

Bei vielen Mikroprozessoren und Mikrocontrollern wächst der Stapelzeiger nach "Unten"[3], so auch beim ATmega32. Der Stapelzeiger adressiert dabei mit zunehmender Füllung des Stapelspeichers immer niedrigere Adressen. Damit Konflikte mit den anderen Daten im SRAM welche von "Unten" nach "Oben" abgespeichert werden zu vermieden werden, wird der Stapel meist auf der höchst liegenden SRAM-Adresse initialisiert. In der Definitionsdatei des Controllers wurde dieser Wert im Namen **RAMEND** (**0x085F** für den ATmega32) festgelegt. In der Definitionsdatei wurde auch die Adresse des Stapelzeigers den Namen **SPL** (**0x3D**) und **SPH** (**0x3E**) zugeordnet.

Wie schon in der Assemblervorlage gesehen, kann die Initialisierung des Stapelbereichs also folgendermaßen aussehen:

```
.DEF    Tmp1 = r16              ;Register 16 dient als erster Zwischenspeicher

        ;Stapel initialisieren (fuer Unterprogramme bzw. Interrupts)
        ldi     Tmp1,HIGH(RAMEND)       ;RAMEND (SRAM) ist in der Definitions-
        out     SPH,Tmp1                ;datei festgelegt
        ldi     Tmp1,LOW(RAMEND)
        out     SPL,Tmp1
```

Im Ruhezustand (leerer Stapelspeicher) ist der Stapelzeiger mit der Anfangsadresse des Stapelspeichers geladen. Bei jedem Schreiben in den Stapelspeicher wird der Stapelzeiger um 1 ("**push**"-Befehl, Speichern eines Byte) oder 2 (Rücksprungadresse, Speichern von 2 Byte) erniedrigt nachdem die Daten abgelegt wurden (post-decrement). Beim Lesen wird der Stapelzeiger um 1 ("**pop**"-Befehl) oder 2 (Rücksprungadresse) erhöht bevor die Daten abgeholt werden (pre-increment).

Durch den Stapelzeiger wird also so ein beliebig großer Stapelspeicher realisiert. Der Prozessor benötigt dazu nur ein 16-Bit-Register.

3 Wenn man davon ausgeht, dass die niedrigste Adresse unten liegt.

Es ist darauf zu achten, dass der Stapelspeicher während des laufenden Programms keine anderen Speicherzellen in denen sich Programmcode oder Daten befinden überschreibt.

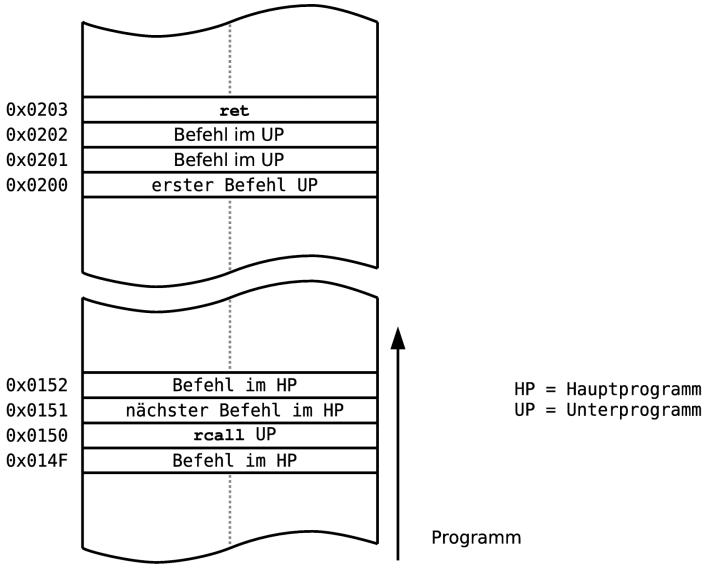
Das Abspeichern der Rücksprungadresse

Beim Aufruf eines Unterprogramms oder einer Interrupt-Routine wird das Hauptprogramm unterbrochen um das Unterprogramm oder die Interrupt-Routine auszuführen. Dabei muss eine Rücksprungadresse gesichert werden damit bei einem "**`ret`**" bzw. "**`reti`**"-Befehl an der Stelle des Hauptprogramms fortgefahren werden kann wo die Unterbrechung erfolgte.

Beim Aufruf des Unterprogramms oder der Interrupt-Routine wird der Inhalt des Befehlszähler **PC**, der bereits auf den folgenden Befehl des Hauptprogramms zeigt, als Rücksprungadresse an der momentanen Adresse des Stapelzeigers abgelegt. Hierbei wird zuerst niederwertige Byte (LByte) abgelegt, und dann erst das höherwertige Byte (HByte). Der Stapelzeiger wird dabei um zwei dekrementiert.

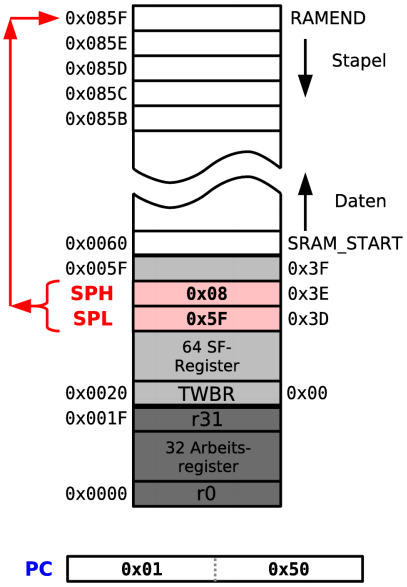
Nach dem Retten der Rücksprungadresse zeigt der Stapelzeiger auf die nächste freie Adresse im Stapel. Die auf dem Stapel abgespeicherte Rücksprungadresse darf während der Ausführung des Unterprogramms nicht verändert werden. Falls doch, muss der alte Inhalt vor Erreichen des Rückkehr-Befehls am Ende des Unterprogramms wiederhergestellt werden. Nachdem das Unterprogramm abgearbeitet wurde zeigt der Stapelzeiger wieder auf den Anfang des Stapels. Der Stapel ist wieder leer, auch wenn die alten Werte sich noch in den Speicherzellen befinden. Beim nächsten Aufruf des Stapels werden sie überschrieben.

Programmspeicher (Flash)

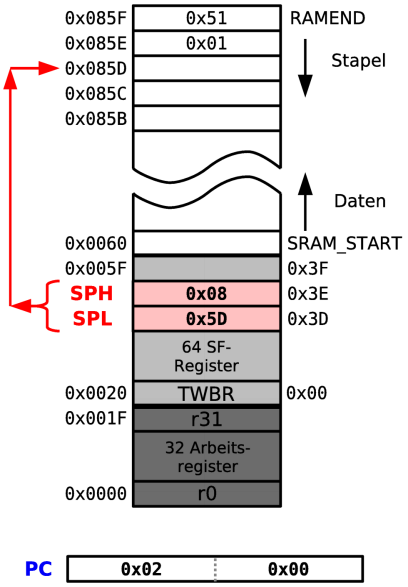


Datenspeicher (SRAM)

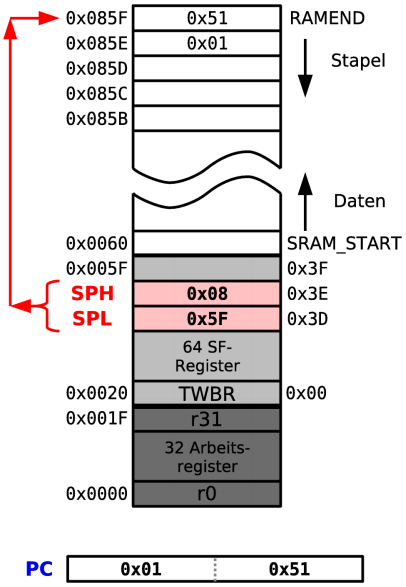
nach der Initialisierung
(vor `rcall`)



nach dem Aufruf des Unterprogramms (`rcall`)



nach dem Rücksprung ins Hauptprogramm (`ret`)
Stapel ist wieder leer!



✎ **B100** a) Teste das Programm "**B100_test_stack_return_adress_1.asm**" im Debug-Modus des Studio 4. Beobachte beim schrittweisen debuggen (Step Into, F11) den Stapelzeiger (I/O View unter CPU oder im Speicherfenster (View Memory), I/O) sowie den Stapel (Speicherfenster, Data).

```
;-------------------------------------------------------------------------;
;       Hauptprogramm
;-------------------------------------------------------------------------
MAIN:   rcall   SR1             ;Unterprogramm aufrufen
        nop                     ;no operation
        rcall   SR1             ;Unterprogramm nochmals aufrufen
        rjmp    MAIN            ;Weiter

;-------------------------------------------------------------------------
;       Unterprogramme
;-------------------------------------------------------------------------
SR1:    ret                      ;Unterprogramm besteht nur aus Ruecksprungbefehl
```

b) Ändere das Unterprogramm so um, dass ein verschachtelter Aufruf zu einem zweiten Unterprogramm erfolgt, und teste das geänderte Programm nochmals wie in Punkt a).
Speichere das Programm als "**B100_test_stack_return_adress_2.asm**".

```
;-------------------------------------------------------------------------
;       Unterprogramme
;-------------------------------------------------------------------------
SR1:    rcall   SR2         ;verschachteltes Unterprogramm
        ret
SR2:    ret                 ;Unterprogramm besteht nur aus Ruecksprungbefehl
```

Funktionsweise bei "push" und "pop"

Programme und Unterprogramme arbeiten mit den gleichen Arbeitsregistersatz. Möchte man in Unterprogrammen mit lokalen Variablen arbeiten, so muss man die benötigten Register unbedingt vor ihrer ersten Verwendung zwischenspeichern (retten) und sie kurz vor Verlassen des Programms wiederherstellen. Dazu bietet sich der Stapel an. Mit den Befehlen "**push**" und "**pop**" lässt sich dies sehr einfach bewerkstelligen.

Der Stapelspeicher kann aber auch im Hauptprogramm beliebig als Zwischenlager eingesetzt werden (Bsp:. Sicherstellung momentaner Flag-Zustände).

Push bedeutet in den Stapel stoßen oder auf den Stapel laden. Bei jedem "**push**"-Befehl wird der Stapelzeiger **SP** automatisch um 1 dekrementiert. ***Pop*** bedeutet aus dem Stack ziehen oder vom Stack wegnehmen. Bei jedem "**pop**"-Befehl wird der Stapelzeiger **SP** automatisch um 1 inkrementiert.

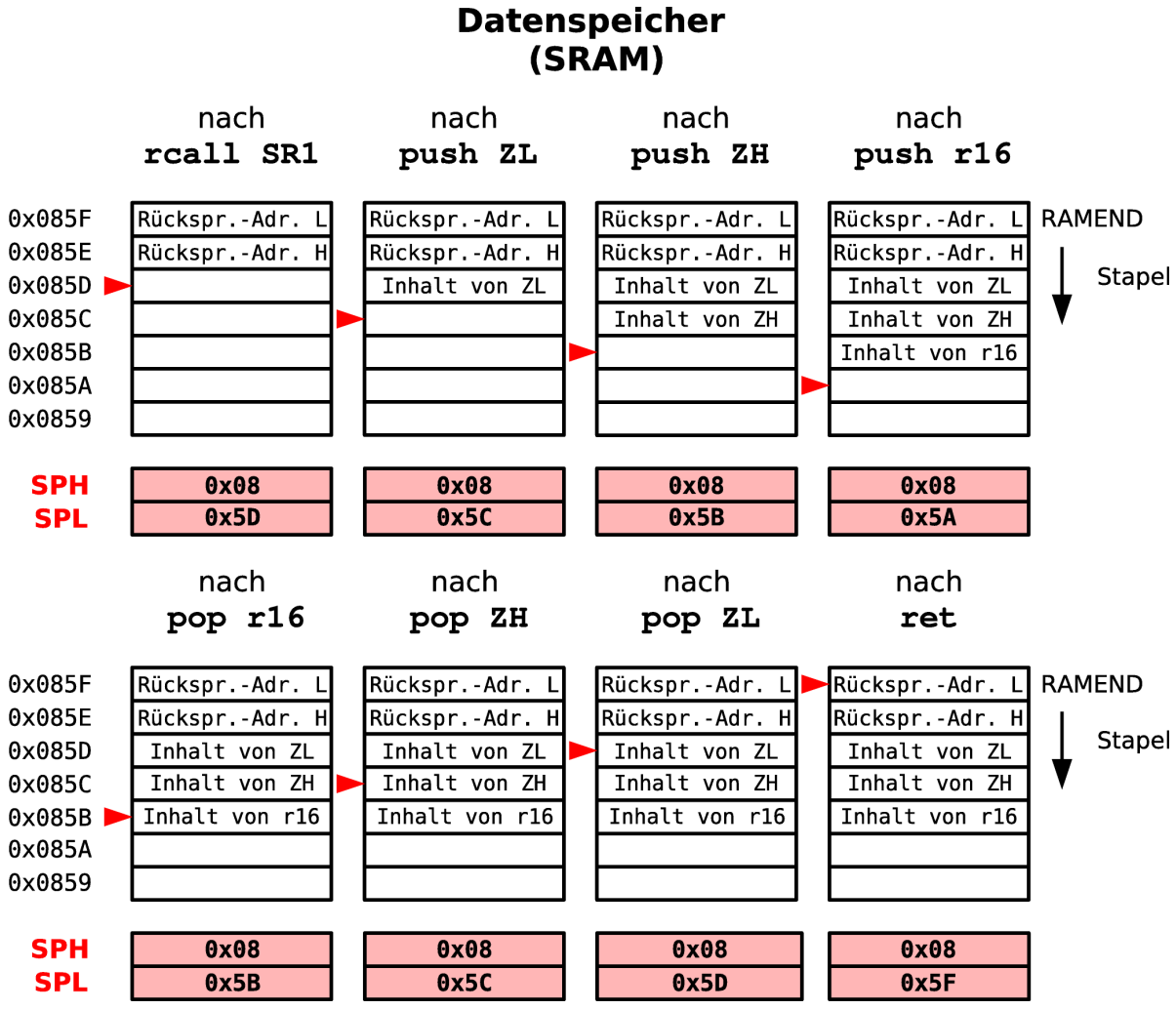
Mit den "**push**"-Befehlen werden Registerinhalte auf den Stapel geladen, während die "**pop**"-Befehle die Registerinhalte wieder restaurieren.

Als Operand können alle Arbeitsregister verwendet werden.

Wegen des LIFO-Prinzip (Last In First Out) müssen die Register in umgekehrter Reihenfolge vom Stapel genommen werden ("pop") wie sie auf den Stapel gelegt wurden ("push")!

Beispiel für ein Unterprogramm:

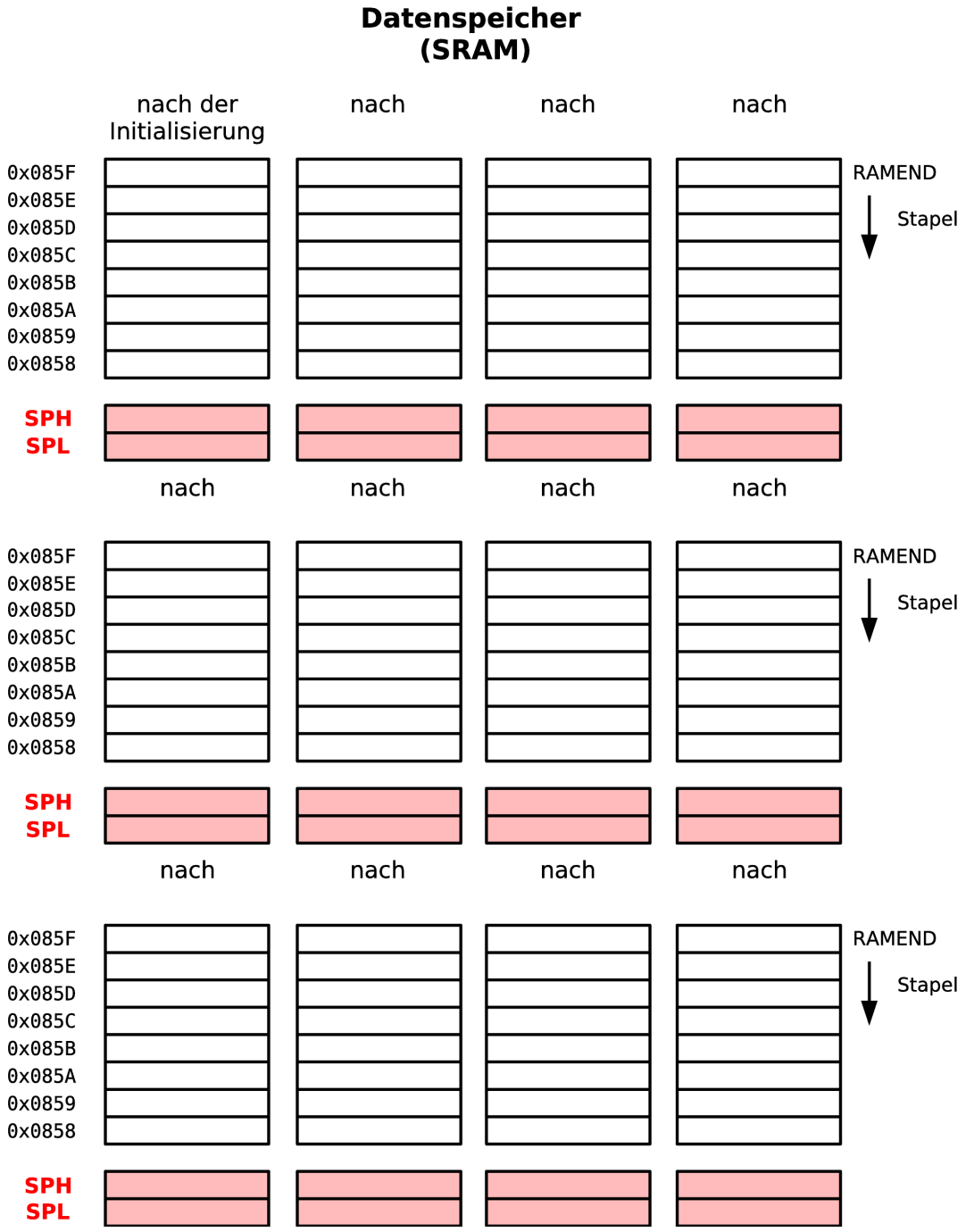
```
SR1:    push    ZL              ;Verwendeten Register retten
        push    ZH
        push    r16

        nop                     ;eigentlicher Code des Unterprogramms

        pop     r16             ;Verwendete Register wiederherstellen
        pop     ZH
        pop     ZL
        ret                     ;Zurueck ins Hauptprogramm
```



✎ **B101** Teste das obige Beispiel für ein Unterprogramm im Debug-Modus des Studio 4. Beobachte beim schrittweisen Debuggen (Step Into, F11) den Stapelzeiger ("I/O View" unter CPU oder im Speicherfenster (View Memory), I/O) sowie den Stapel (Speicherfenster, Data).
Speichere das Programm als "**B101_test_stack_push_pop_1.asm**".

✎ **B102** Überlege beim Programm "**B102_test_stack_push_pop_2.asm**" (hier der Auszug aus der List-Datei) wie der Stapel sich verändert und trage die Inhalte der Speicherzellen und des Stapelzeigers in das folgende Diagramm ein (Der Stapel wurde natürlich initialisiert). Markiere ebenfalls die Position des Stapelzeigers:

```
49: ;-----------------------------------------------------------------------
50: ;       Hauptprogramm
51: ;-----------------------------------------------------------------------
52: 00002E   E111  ldi     r17,0x11        ;Register 17 initialisieren
53: 00002F   E222  ldi     r18,0x22        ;Register 18 initialisieren
54: 000030   D002  rcall   SR1             ;Unterprogramm aufrufen
55: 000031   0000  nop                     ;no operation
56: 000032   CFFB  rjmp    MAIN            ;Weiter
57:
58: ;-----------------------------------------------------------------------
59: ;       Unterprogramme
60: ;-----------------------------------------------------------------------
61: 000033   931F  push    r17             ;SR1: Verwendete Register retten
62: 000034   932F  push    r18
63: 000035   E313  ldi     r17,0x33        ;r17 fuer lokale Verwend. SR1 init.
64: 000036   D003  rcall   SR2
65: 000037   912F  pop     r18             ;Verwendete Reg. wiederherstellen
66: 000038   911F  pop     r17
67: 000039   9508  ret                     ;Zurueck ins Hauptprogramm
68: 00003A   931F  push    r17             ;SR2: Verwendeten Register retten
69: 00003B   E414  ldi     r17,0x44        ;r17 fuer lokale Verwend. SR2 init.
70: 00003C   911F  pop     r17             ;Verwendete Reg. wiederherstellen
71: 00003D   9508  ret                     ;Zurueck ins Unterprogramm SR1
```



✎ **B103**

a) Entwickle das Flussdiagramm zu einem ein Programm das in einer Schleife fünf im Zahlenraum nacheinander folgende 16-Bit-Zahlen im Stapelspeicher ablegt und diese in einem zweiten Schritt über eine Schleife wieder ausliest und im SRAM ab Adresse **`0x0060`** ablegt (Lowest Byte First).

b) Schreibe das Programm und nenne es "**`B103_push_pop_stack.asm`**"

c) Betrachte die Funktionsweise des Programms mit Hilfe des Studio 4.