

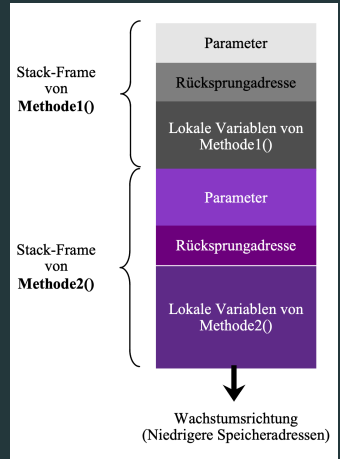
Aufgabe 1

- Überlege, wie ein Assemblerprogramm aussieht, das das Hochsprachenkonzept eines Methodenaufrufs umsetzt
 - Eine Methode wird aus einer anderen Stelle im Programm aufgerufen
 - Der Methode können Parameter übergeben werden
 - Die Methode kann ein Ergebnis zurückgeben
 - Nach Abarbeitung der Methode wird die Programmausführung direkt “hinter” dem Aufruf fortgesetzt
- Schreibe ein einfaches Assemblerprogramm, das eine “Methode” **addiere** realisiert
 - Die Methode soll zwei Zahlen als Parameter erhalten und das Ergebnis zurückgeben

- Die selbe Methode kann aus sehr vielen unterschiedlichen Stellen aufgerufen werden
 - Die “richtige” Rücksprungadresse muss beim Aufruf irgendwo gespeichert werden
- Eine Methode kann “in sich selbst” aufgerufen werden, also bevor der letzte Aufruf zurückgekehrt ist
 - Rekursion

Der Stack (1)

- Speicherbereich, der dynamisch wächst (und schrumpft)
- Speichert “aufrufspezifische” Daten
 - Rücksprungadresse
 - Übergebene Parameter
 - Methodenlokale Variablen
- Jeder weitere Methodenaufruf erzeugt einen neuen **Stack Frame** auf dem Stack
- Jede Rückkehr aus einer Methode baut den “obersten” Stack Frame ab



Der Stack (2)

- Der sogenannte **Stack Pointer** ist ein spezielles Prozessorregister, das immer auf die “Spitze” des Stacks zeigt
 - Meistens wächst der Stack “nach unten”, also in Richtung niedrigerer Speicheradressen
- Oft gibt es ein weiteres Register namens **Frame Pointer**, dessen Wert auf den Anfang des aktuellen Stack Frames verweist
 - Vereinfacht Zugriff auf Parameter und Abbau des Stack Frames nach Rückkehr
- Die sogenannte **Calling Convention** legt Details des Aufrufs fest, z. B.:
 - Reihenfolge der übergebenen Parameter
 - Zuständigkeit für das “Aufräumen” des Stacks (callee vs. caller)

Beispielhafter Ablauf

1. Parameter für die Funktion auf den Stack legen, Stack Pointer anpassen
2. Rücksprungadresse auf den Stack legen, Stack Pointer anpassen
 - Wie bestimmt man die Rücksprungadresse?
3. Sprung zur Adresse des Methodencodes
4. Lokale Variablen auf den Stack legen, Stack Pointer anpassen
5. Vor der Rückkehr: Lokale Variablen entfernen
 - Der Aufrufer weiß nichts von diesen Variablen!
6. An Rücksprungadresse springen
7. Parameter vom Stack entfernen

Aufgabe 2

- Informiere Dich im Manual der Minimaschine über die Möglichkeit, einen Stack zu verwenden
- Passe das Programm aus Aufgabe 1 so an, dass die Methode **addiere** unter Verwendung des Stacks aufgerufen wird
- Hinweis: Es ist möglich, auf Adressen relativ zum Stack Pointer zuzugreifen, z. B.:

```
load 2(sp) # Auf sp+2 zugreifen
```