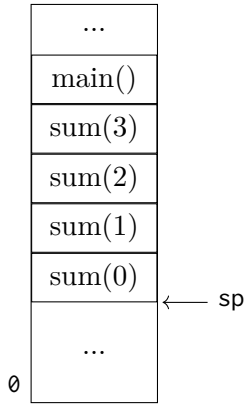
# 7.2: Funktionsaufrufe – Mit Rekursion

Die Aktivierungsrahmen der beiden Funktionen beinhalten mindestens die folgenden Informationen:

| `sum()` |
| --- |
| • `ret_sum`: Rückgabewert |
| • `n` |
| • `addr_sum`: Rücksprungadresse |
| • `s` |

| `main()` |
| --- |
| • `ret_main`: Rückgabewert |
| • `addr_main`: Rücksprungadresse |
| • `r` |

Im Folgenden sollen die Aktivierungsrahmen vollständig auf dem Stack liegen (d.h. auch Parameter und Rückgabewerte werden ausschließlich über den Stack übergeben). Dabei sollen die in obiger Darstellung weiter *oben* eingetragenen Felder an *höheren* Adressen liegen. Der Stack soll von hohen Adressen in Richtung niedrigerer Adressen wachsen, das Register `sp` soll den *Stackpointer* beinhalten (und zeigt damit im folgenden Beispiel auf das `r`-Feld des Aktivierungsrahmens von `sum(0)`):




Das Beispiel kann damit wie folgt übersetzt werden:

```
sum:
  sp -= 8          // Platz für lokale Variable s

  ...

  24(sp) = 0(sp)   // Wert von s als Rückgabewert setzen

  sp += 8          // Platz für lokale Variable s entfernen
  ret              // Rücksprung entfernt implizit die Rücksprungadresse vom Stack

main:
  sp -= 8          // Platz für lokale Variable r

  0(sp) = 3        // r := 3;

  sp -= 8          // Platz für Rückgabewert
  push 8(sp)       // Argument für sum(), Achtung: anderer Offset!
  call sum         // Funktionsaufruf, legt implizit Rücksprungadresse auf Stack
  sp += 8          // Argument vom Stack entfernen
  pop 8(sp)        // Ergebnis von sum() nach r kopieren, Achtung: anderer Offset!

  16(sp) = 0(sp)   // Wert von r als Rückgabewert setzen

  sp += 8          // Platz für lokale Variable r entfernen
  ret              // Rücksprung, entfernt implizit die Rücksprungadresse vom Stack
```

Die Felder des jeweiligen Aktivierungsrahmens werden dabei relativ zum Stackpointer `sp` adressiert, bspw. wird bei `16(sp)` der Wert 16 auf die Adresse im `sp`-Register addiert und der an der resultierenden Adresse liegende (8-Byte große) Wert gelesen bzw. geschrieben. Problematisch dabei ist, dass sich der Stackpointer `sp` beim Aufbau eines Aktivierungsrahmens für einen Funktionsaufruf ändert und sich dadurch die *Offsets verschieben*. Je nach „Zustand“ des Stacks muss also beispielsweise ein anderer Offset für den Zugriff auf die lokale Variable `r` in `main()` verwendet werden. Der Übersetzer muss also die Elemente auf dem Stack „mitzählen“.

Um dieses Problem zu umgehen, kann zusätzlich zum Stackpointer noch ein sog. *Framepointer* `fp` (ebenfalls ein Register) verwendet werden. Dieser zeigt immer auf eine *definierte Position* im Kellerrahmen und bleibt während der Ausführung einer Funktion *konstant*. Insbesondere bedeutet dies, dass sich der Framepointer *nicht ändert*, wenn zusätzliche Werte auf den Stack gelegt werden, wodurch die Offsets für die Zugriffe auf die Werte des Aktivierungsrahmens immer *gleich* sind.

Das Beispiel kann damit wie folgt übersetzt werden:

```
sum:
  push fp          // alten Framepointer für spätere Restaurierung sichern
  fp = sp          // Framepointer setzen

  sp -= 8          // Platz für lokale Variable s

  ...

  24(fp) = -8(fp) // Wert von s als Rückgabewert setzen

  sp = fp          // Stack abräumen (statt 'sp += 8')
  pop fp           // alten Framepointer restaurieren
  ret              // Rücksprung, entfernt implizit die Rücksprungadresse vom Stack

main:
  push fp          // alten Framepointer für spätere Restaurierung sichern
  fp = sp          // Framepointer setzen

  sp -= 8          // Platz für lokale Variable r

  -8(fp) = 3       // r := 3;

  sp -= 8          // Platz für Rückgabewert
  push -8(fp)      // Argument für sum(), jetzt gleicher Offset!
  call sum         // Funktionsaufruf, legt implizit Rücksprungadresse auf Stack
  sp += 8          // Argument vom Stack entfernen
  pop -8(fp)       // Ergebnis von sum() nach r kopieren, jetzt gleicher Offset!

  16(fp) = -8(fp) // Wert von r als Rückgabewert setzen

  sp = fp          // Stack abräumen (statt 'sp += 8')
  pop fp           // alten Framepointer restaurieren
  ret              // Rücksprung, entfernt implizit die Rücksprungadresse vom Stack
```

Die Übersetzung ist nun *einfacher*, da sich die Offsets für die Zugriffe nun nicht mehr ändern und der Übersetzer deshalb nicht die Elemente auf dem Stack „mitzählen“ muss. Dafür wird nun ein *zusätzliches Register* `fp` zur Verwaltung des Framepointers benötigt. Da i.A. mehrere Funktionen dieses Register verwenden, muss dieses bei einem Aufruf *gesichert* und nach Ausführung der Funktion wieder *restauriert* werden (hier: *Callee-Save-Register*, aufgerufene Funktion sichert und restauriert; siehe auch Aufgabe 7.3). Dies beudetet auch, dass die *Aktivierungsrahmen größer* werden und jede Funktionsaufführung deshalb *mehr Speicher* benötigt.

Außerdem: Aktueller Framepointer zeigt auf gesicherten Framepointer, der wiederum auf gesicherten Framepointer zeigt usw. Damit bilden die Framepointer eine „Kette“, über die man sich von Aktivierungsrahmen zu Aktivierungsrahmen hangeln kann. Dies wird beispielsweise von *Debuggern* verwendet.