

# Übung 04: Rekursion und Calling Convention

Einführung in die Rechnerarchitektur

**Niklas Ladurner**

School of Computation, Information and Technology  
Technische Universität München

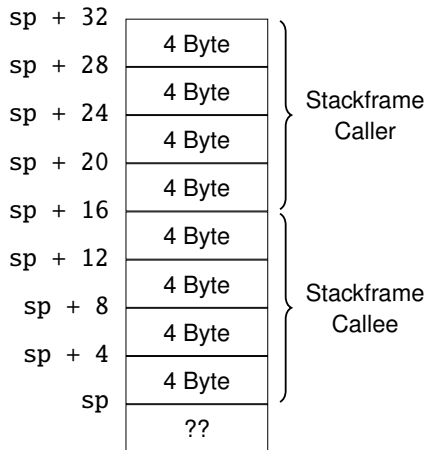
7. November 2025



*TUM Uhrenturm*

Keine Garantie für die Richtigkeit der Tutorfolien.  
Bei Unklarheiten/Unstimmigkeiten haben VL/ZÜ-Folien recht!

- Speicherbereich für lokale Variablen
- wächst von hohen Adressen zu niedrigen Adressen
- Anpassen des Stackpointers durch Beschreiben des `sp`-Registers
- alles an Adressen  $< \text{sp}$  ungeschützt
- besonders wichtig für Rekursion (Stackframes)



## Caller und Callee (1)

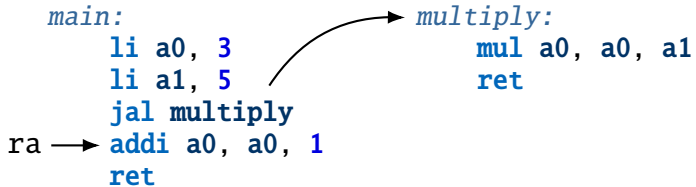
```
main:
    li a0, 3
    li a1, 5
    jal multiply
    addi a0, a0, 1
    ret
```

```
multiply:
    mul a0, a0, a1
    ret
```

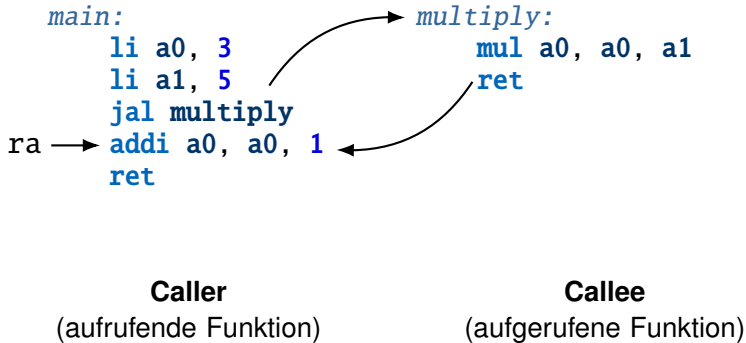
## Caller und Callee (1)

```
main:
    li a0, 3
    li a1, 5
    jal multiply
ra → addi a0, a0, 1
    ret

multiply:
    mul a0, a0, a1
    ret
```



## Caller und Callee (1)



## Caller und Callee (2)

*caller:*

# aufrufende Funktion (Caller)

# Wir speichern die Rücksprungadresse auf  
# den Stack -> ra ist Caller-saved!

**addi sp, sp, -16**

**sw ra, 0(sp)**

# ... irgendwas, was t0 verwendet

# da t0 caller-saved ist, müssen wir uns  
# t0 absichern, wenn wir den Inhalt später  
# noch brauchen

**sw t0, 4(sp)**

# ...

**jal ra, callee** # Sprung zur Unterfunktion

# ...

**lw t0, 4(sp)**

# ... wieder irgendwas mit t0

**lw ra, 0(sp)**

**addi sp, sp, 16**

**jalr zero, 0(ra)**

*callee:*

# aufgerufene Funktion (Callee)

# hier dürfen wir t0-t6 bspw. verändern  
# falls wir s0-s6 verändern wollen würden,  
# würden wir das so machen:

**addi sp, sp, -16**

**sw s2, 0(sp)**

**sw s3, 4(sp)**

# s2, s3 können jetzt verwendet werden!

**lw s2, 0(sp)**

**lw s3, 4(sp)**

**addi sp, sp, 16**

**jalr zero, 0(ra)**

fürs Selbststudium :)

# Calling Convention

- „Aufrufkonvention“ → lediglich eine Vereinbarung
- definiert Parameterüberabe, Rückgabe, Registersicherung, Stack etc.
  1. Datentypen  $\leq 4$  Byte in a-Register, sign-extension falls vorzeichenbehafteter Datentyp
  2. Datentypen = 8 Byte in zwei a-Register, niedrigwertige Hälfte zuerst
  3. Datentypen  $> 8$  Byte als Pointer auf Caller-Stack
  4. Falls zu wenige Register: Übergabe über Stack
- Stackpointer muss immer ein Vielfaches von 16 Byte sein!



Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-11	a0-1	Function arguments/return values	Caller
x12-17	a2-7	Function arguments	Caller
x18-27	s2-11	Saved registers	Callee
x28-31	t3-6	Temporaries	Caller

# Rekursion

```
1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)
```

**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung

## ■ Funktion die **sich selbst aufruft**

### ■ Aufbau:

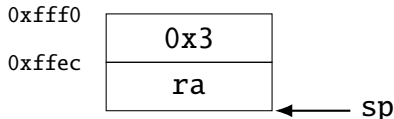
1. Abbruchbedingung(en)
2. Sicherung von ra und evtl. Parametern
3. Vorbereitung der Parameter für den rekursiven Aufruf
4. Rekursiver Aufruf
5. Ergebnis des Aufrufs verwerten
6. Wiederherstellung von ra, sp
7. Rücksprung

### ■ Schritte 3-5 können mehrmals vorkommen (vgl. Fibonacci)

# Rekursion

```
1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)
```

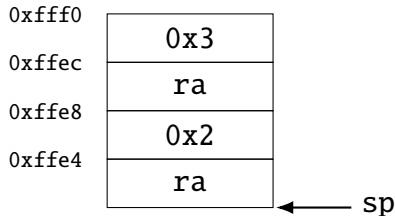
Aufruf mit `a0 = 3`:



**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung

```
1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)
```

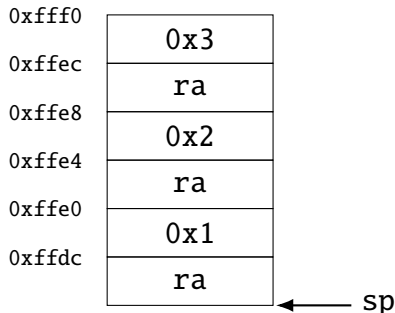
Aufruf mit a0 = 3:



**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung

```
1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)
```

Aufruf mit `a0 = 3`:



**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung

# Rekursion

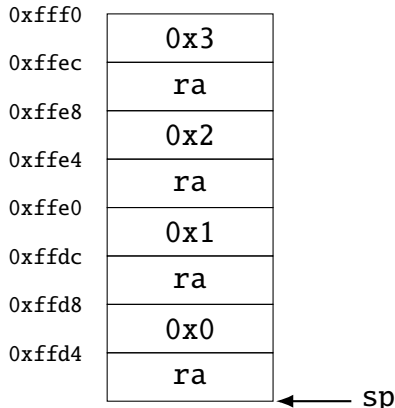
```

1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)

```

**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung

Aufruf mit a0 = 3:



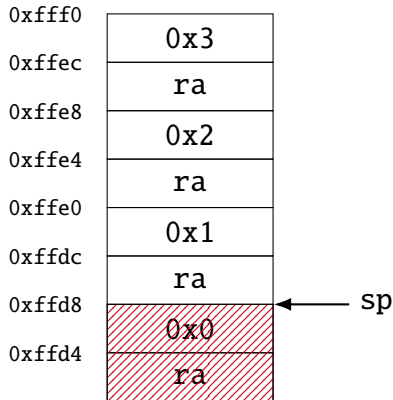
# Rekursion

```

1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)
  
```

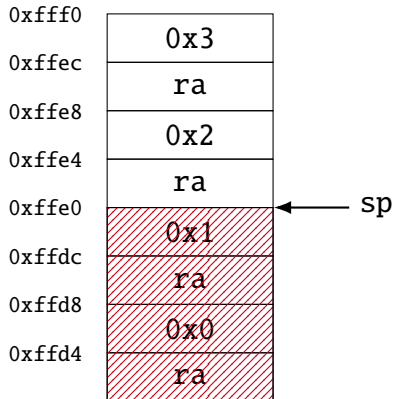
**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung

Aufruf mit a0 = 3:



```
1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)
```

Aufruf mit `a0 = 3`:

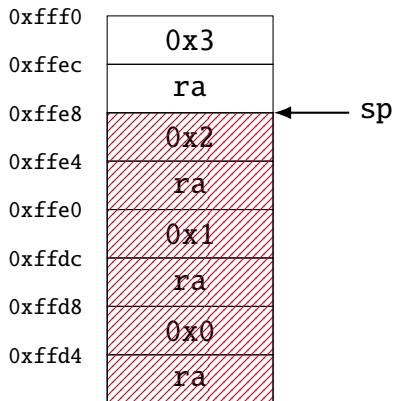


**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung



```
1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)
```

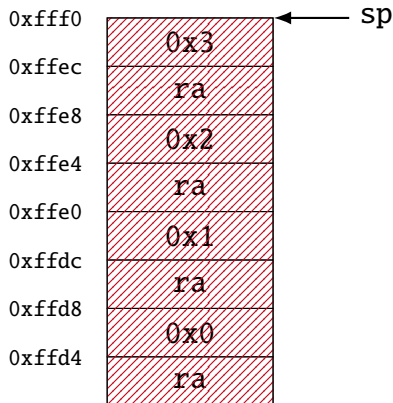
Aufruf mit  $a0 = 3$ :



**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung

```
1 fun:
2     addi sp, sp, -8
3     sw ra, 0(sp)
4     sw a0, 4(sp)
5     beq a0, zero, end
6     addi a0, a0, -1
7     jal fun
8 end:
9     lw ra, 0(sp)
10    addi sp, sp, 8
11    jalr zero, 0(ra)
```

Aufruf mit a0 = 3:



**Achtung:** 8 Byte nicht CC-konform, nur zur besseren Darstellung

Fragen?

- Zulip: „ERA Tutorium – Mi-1600-3“ bzw. „ERA Tutorium – Fr-1500-1“
- ERA-Moodle-Kurs
- ERA-Artemis-Kurs
- Übersicht an RISC-V-Instruktionen
- Übersicht an RISC-V-Pseudoinstruktionen

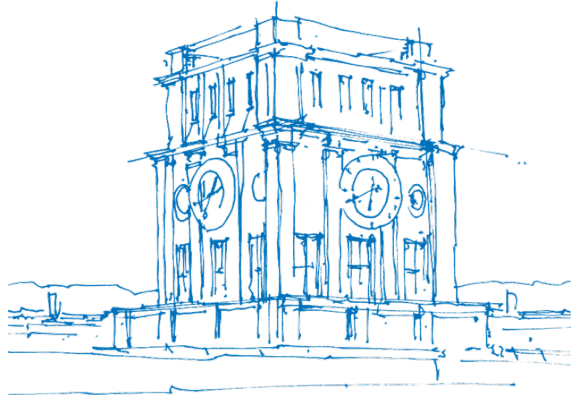
# Übung 04: Rekursion und Calling Convention

Einführung in die Rechnerarchitektur

**Niklas Ladurner**

School of Computation, Information and Technology  
Technische Universität München

7. November 2025



*TUM Uhrenturm*