

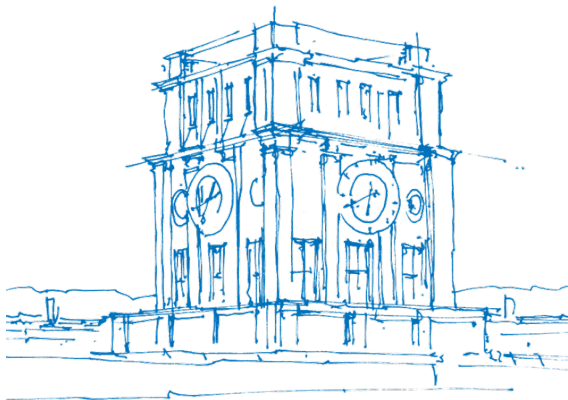
Übung 03: Rekursion und Calling Convention

Einführung in die Rechnerarchitektur

Niklas Ladurner

School of Computation, Information and Technology
Technische Universität München

8. November 2024



TUM Uhrenturm

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

Caller vs. Callee

```

caller:
# hier befinden wir uns in der aufrufenden
# Funktion (Caller)

# Wir speichern die Rücksprungadresse auf
# den Stack -> ra ist Caller-saved!
addi sp, sp, -16
sw ra, 0(sp)

# ... irgendwas, das t0 verwendet, bspw.
addi t0, zero, 2

# 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:
# hier befinden wir uns in der aufgerufenen
# 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 :)

Caller- und Callee-saved Register

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	—
x5–7	t0–2	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
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Abbildung 1 Übersicht über die RISC-V-Register

Calling Convention

- „Aufrufkonvention“ → lediglich eine Vereinbarung
- definiert Parameterüberabe, Rückgabe, Registersicherung, Stack etc.
- Datentypen ≤ 4 Byte in a-Register, signextension
- Datentypen = 8 Byte in 2 a-Register, niedrigwertige Hälfte zuerst
- Datentypen > 8 Byte als Pointer (Zeiger auf Speicher)
- Falls zu wenige Register: Übergabe über Stack
- Stackpointer muss immer Vielfaches von 16 Byte sein!

Rekursion

- Funktion die **sich selbst aufruft**
- \exists äquivalente iterative Funktion für jede rekursive Funktion
- 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 verwenden
 6. Wiederherstellung von ra, sp
 7. Rücksprung

Rekursion: Beispiel

```
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)
```

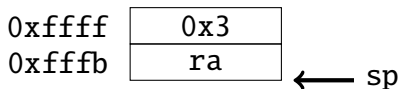
Aufruf mit $a0 = 3$:

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

Rekursion: Beispiel

```
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: Beispiel

```

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 $a0 = 3$:

0xfffff	0x3
0xffffb	ra
0xffff7	0x2
0xffff3	ra

← sp

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

Rekursion: Beispiel

```

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 $a0 = 3$:

0xfffff	0x3
0xffffb	ra
0xffff7	0x2
0xffff3	ra
0xffef	0x1
0xffeb	ra

← sp

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

Rekursion: Beispiel

```

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 $a0 = 3$:

0xfffff	0x3
0xffffb	ra
0xffff7	0x2
0xffff3	ra
0xffef	0x1
0xffeb	ra
0xffe7	0x0
0xffe3	ra

← sp

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

Rekursion: Beispiel

```

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:

0xfffff	0x3
0xffffb	ra
0xffff7	0x2
0xffff3	ra
0xffef	0x1
0xffeb	ra
0xffe7	0x0
0xffe3	ra

← sp

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

Rekursion: Beispiel

```

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:

0xfffff	0x3	
0xffffb	ra	
0xffff7	0x2	
0xffff3	ra	
0xffef	0x1	← sp
0xffeb	ra	
0xffe7	0x0	
0xffe3	ra	

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

Rekursion: Beispiel

```

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:

0xfffff	0x3	← sp
0xffffb	ra	
0xffff7	0x2	
0xffff3	ra	
0xffef	0x1	
0xffeb	ra	
0xffe7	0x0	
0xffe3	ra	

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

Rekursion: Beispiel

```

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 $a0 = 3$:

0xfffff	0x3	← sp
0xffffb	ra	
0xffff7	0x2	
0xffff3	ra	
0xffef	0x1	
0xffeb	ra	
0xffe7	0x0	
0xffe3	ra	

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

Fragen?

(Die ZÜ-Folien sind sehr gut, schaut euch die an)

- „H04 — Tribonacci“ bis 17.11.2024 23:59 Uhr
- mehrfache rekursive Aufrufe in einer Unterfunktion, Sicherung von Parametern
- Einhaltung der CC ab sofort Pflicht!

- Zulip: „ERA Tutorium - Do-1600-1“ bzw. „ERA Tutorium - Fr-1500-2“
- RISC-V-Spezifikation
- ERA-Moodle-Kurs
- ERA-Artemis-Kurs
- Übersicht an RISC-V-Instruktionen
- Übersicht an RISC-V-Pseudoinstruktionen

Übung 03: Rekursion und Calling Convention

Einführung in die Rechnerarchitektur

Niklas Ladurner

School of Computation, Information and Technology
Technische Universität München

8. November 2024



TUM Uhrenturm