



Grundlagen digitaler Systeme

Kapitel 10: Unterprogramme und Unterprogrammaufrufe

Holger Karl

Was bisher geschah

- Auf Basis der RISC-V Instruction Set Architecture den konkreten Instruktionssatz von RISC-V besprochen
 - Zumindest relevante Teile davon
- Assembler als Sprache, Assembler als Übersetzungswerkzeug in Maschinensprache
- Überführung von Pseudo-Code / Hochsprachen-Programm nach Assembler

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung
Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 2/73

Plan für dieses Kapitel

2.9, [22]

- Wir führen Unterprogramme als Strukturierungsmöglichkeit für Programme ein
- Mechanismen, wie Unterprogramme aufgerufen werden können
 - Für direkte, indirekte Rekursion
 - Mit Übergabe von Parametern
- Zur Zusammenarbeit von Programmierern führen wir **Konventionen** für den Aufruf ein

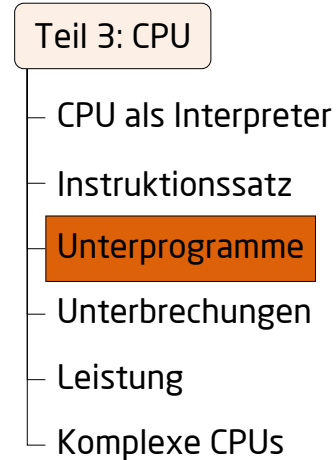


Abbildung 10.1: Instruktionssatz

Unterprogramme
Aufrufe
Unterprogramme in
RISC-V
Zusammenfassung
Material

Sie sind in der Lage,

- Unterprogramme in Assembler zu schreiben und
- vorhandene Unterprogramme aufzurufen.
- Dabei beachten Sie relevante Aufrufkonventionen.

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung
Material

Inhaltsverzeichnis

1. Unterprogramme

1.1 Motivation

1.2 Anforderung

2. Aufrufe

3. Unterprogramme in RISC-V

4. Zusammenfassung

5. Material

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung
Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 5/73

Direkter Code mit goto

- Bis jetzt: Lineares Programm, mit Sprüngen (goto)
 - Mehr oder minder beliebige Modifikation des Programmzählers
- Im Prinzip reicht das, um beliebige Programme zu schreiben
 - Wird schnell beliebig unübersichtlich
 - “Goto considered harmful” [Dij68]
- Struktur?

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 6/73

Motivation: Wiederbenutzung

- Mehrfaches Benutzen des gleichen Codes
 - Von verschiedenen Stellen des Programms aus
- Benutzen des Codes anderer Programmierern

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 7/73

Motivation: Parameter

- Den **identischen** Code mehrfach zu benutzen wäre starke Einschränkung
- Code mit **Parameter** versehen, die verarbeitet werden, Ablauf beeinflussen

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 8/73

Definition 10.1 (Unterprogramm)

- Wieder zu benutzender Code wird in **Unterprogramme** organisiert
- Unterprogramm ist Code-Sequenz, die
 - von anderen Code-Teilen aufgerufen werden kann,
 - **Parameter** entgegen nehmen kann,
 - beim **Aufruf** konkrete **Werte** für die Parameter entgegen nimmt,
 - am Ende der Ausführung optional einen Wert für den aufrufenden Programmteil bereitstellt (der **Rückgabewert**).

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 9/73

Semantik?

- Unterprogramme gibt es in allen Hochsprachen
 - Als Funktion, Prozedur, Methode, . . .
- Mit im Detail unterschiedlicher Semantik

Beispiele

C, C++

```
long some_fct(int x) {  
    // do something  
    return 2*x;  
}
```

Python

```
def some_fct (x):  
    # do something  
    return 2*x
```

Beispiele

Java

```
public static void main(String[] argv) {  
    // do something  
    ...  
}
```

Lisp

```
(defun some_fct (x)  
  ;; do something  
  (return 2*x) )
```

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 10/73

Inhaltsverzeichnis

1. Unterprogramme

1.1 Motivation

1.2 Anforderung

2. Aufrufe

3. Unterprogramme in RISC-V

4. Zusammenfassung

5. Material

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung
Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 11/73

Anforderungen allgemein

- Unterprogrammkonzepte, Semantik **beliebiger** Hochsprachen unterstützen
- Beliebige Semantik für Parameter unterstützen, insbesondere:
 - **Werteparameter** (call-by-value): Ein konkreter Wert wird übergeben
 - **Referenzparameter** (call-by-reference): Eine Referenz auf eine Variable wird übergeben
 - ... weitere (call-by-name,)
- Übergabe von **Ergebnissen**
- Unterprogramme dürfen **eigene Variablen** benutzen
- Effizient für **aufzufendenden** wie **aufgerufenen** Programmteil

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 12/73

Indirekte Rekursion

- Indirekte Rekursion: Ein Unterprogramm ruft ein anderes auf, dass ein drittes aufruft, . . .
 - Beliebiger tief verschachtelt
 - Mit beliebiger Parameterübergabe, Ergebnissen

```
void h() { /*...*/ }  
void g() { h(); }  
void f() { g(); }  
void main () { f(); }
```

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung
Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 13/73

Direkte Rekursion

■ Direkte Rekursion

- Unterprogramm ruft **sich selbst** auf
 - Für manche Eingabeparameter;
dann mit anderen Parametern
 - Sonst: Ende der Aufrufe

```
uint fak(uint n) {  
    if (n > 1) {  
        uint res = fak(n-1);  
        res = res * n;  
        return res;  
    } else  
        return n;  
}
```

Unterprogramme

Motivation

Anforderung

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung
Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 14/73

Inhaltsverzeichnis

1. Unterprogramme

2. Aufrufe

2.1 Strohmänner

2.2 Stack

2.3 Frame

2.4 Ablauf mit Stapel: Details

3. Unterprogramme in RISC-V

4. Zusammenfassung

5. Material

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 15/73

Sprung zu, von Unterprogramm

- Der Transfer der Ausführung **an** das Unterprogramm ist einfach: PC auf Startadresse setzen
- Aber: Wo **nach** Unterprogramm fortsetzen?
 - **Nicht** immer an gleicher Stelle
 - Woher weiß das Unterprogrammprogramm, wohin es am Ende springen soll?

```
void f() {  
    /* irgendwas */  
    return;  
}  
  
void main() {  
    /* Aufruf 1: */  
    f();  
    /* weiter nach Aufruf 1 */  
  
    /* Aufruf 2: */  
    f();  
    /* weiter nach Aufruf 2 */  
}
```

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in RISC-V

Zusammenfassung Material

GDS 10: Unterpro- grammaufrufe

H. Karl, WS 22/23

Folie 16/73

Rücksprungadresse als Parameter behandeln

Definition 10.2 (Rücksprungadresse)

Adresse, an der **nach** Beendigung eines Unterprogramms der Programmablauf fortgesetzt werden soll. In der Regel die Adresse, die auf den Aufruf des Unterprogramms folgt.

- Im Prinzip können wir Rücksprungadresse wie Parameter behandeln
- Also: Mechanismus für Parameter, Resultate benötigt

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 17/73

Definition 10.3 (Strohmann)

Hier: Eine Idee oder Argument, die/das vermutlich falsch ist und zurückgewiesen oder verbessert werden soll. Dient des Verständnisses von Unzulänglichkeiten einer ggf. naheliegenden Idee.

(Das ist **nicht** die klassische Definition der Rhetorik!)

Unterprogramme

Aufrufe

[Strohmann](#)

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 18/73

Parameterübergabe an festen Adressen

- (schlechte) Idee:
 - Aufrufer schreibt alle Parameter an eine feste, bekannte Adresse im Speicher
 - Auch: Rücksprungadresse
 - aufgerufenes Unterprogramm findet dort alle Werte vor

Unterprogramme

Aufrufe

Strohmann

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung


Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 19/73

Parameterübergabe an festen Adressen: Scheitert!

- Versagt bei direkter oder indirekter Rekursion!
 - Z.B. würde die erste Rücksprungadresse durch die zweite überschrieben
 - Nach Rückkehr aus zweitem Aufruf wüsste erstes Unterprogramm nicht mehr, wohin es zurückkehren soll
- Beantwortet nicht: Variablen innerhalb des Unterprogramms wo ablegen?
- 

Unterprogramme

Aufrufe

[Strohmann](#)

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 20/73

Parameterübergabe in festen Registern

- Alternatividee (auch schlecht):
 - Lege für Parameter, Rücksprungadresse Register fest
- Scheitert aus gleichen Gründen ☹️

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 21/73

Inhaltsverzeichnis

1. Unterprogramme

2. Aufrufe

2.1 Strohмänner

2.2 Stack

2.3 Frame

2.4 Ablauf mit Stapel: Details

3. Unterprogramme in RISC-V

4. Zusammenfassung

5. Material

Unterprogramme

Aufrufe

Strohмänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 22/73

Warum scheitert?

- Idee zu festen Speicheradressen, festen Registern scheitert an der **festen Lage** und der **festen Anzahl**
 - Wohingegen Unterprogrammaufrufe variabel sind, durch (in-)direkte Rekursion jede feste Anzahl übersteigen können
 - Ebenso Variablen in Unterprogrammen: Anzahl, Größe nicht im vorhinein festzulegen

Unterprogramme

Aufrufe

Strohmann

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 23/73

Warum scheitert?

- Idee zu festen Speicheradressen, festen Registern scheitert an der **festen Lage** und der **festen Anzahl**
 - Wohingegen Unterprogrammaufrufe variabel sind, durch (in-)direkte Rekursion jede feste Anzahl übersteigen können
 - Ebenso Variablen in Unterprogrammen: Anzahl, Größe nicht im vorhinein festzulegen
- Wir brauchen: **Datenstruktur variabler GröÙer!**
- Eigenschaft:
 - Zuletzt hinzugefügten Daten werden als erstes gebraucht

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 23/73

Stack (ganz allgemein)

Definition 10.4 (Stack (Kellerspeicher))

Ein **Stack** ist eine Datenstruktur beliebiger Größe; hier: einer beliebigen Anzahl von Daten fester Größe. Ein Stack besitzt insbes. zwei Operationen:

- **push**: Legt ein weiteres Datum auf den Stack
- **pop**: Entfernt das oberste Datum vom Stack und stellt es zur Benutzung bereit. (Undefiniert bei leerem Stack)

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 24/73

Stack: Implementierung

- Wir nutzen einen Teil des Hauptspeichers für einen Stack
- Wir benötigen Information, wo der Stack endet / wo ein neues Element hinzugefügt werden soll
 - Das ist lediglich **eine** Adresse des Hauptspeichers
 - Diese Adresse merken wir uns als **Stapelzeiger** (vgl. Folie 26, Def. 10.5)

Unterprogramme

Aufrufe

Strohmann

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 25/73

Definition 10.5 (Stapelzeiger, Stackpointer)

- Ein Stack benötigt einen **Stapelzeiger**
- Je nach Implementation kann das sein:
 - Die Adresse des letzten belegten Elementes (üblich wegen einfacheren Zugriffs)
 - Die Adresse des ersten freien Elementes
 - Im Prinzip äquivalent

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 26/73

Definition 10.5 (Stapelzeiger, Stackpointer)

- Ein Stack benötigt einen **Stapelzeiger**
- Je nach Implementation kann das sein:
 - Die Adresse des letzten belegten Elementes (üblich wegen einfacheren Zugriffs)
 - Die Adresse des ersten freien Elementes
 - Im Prinzip äquivalent
- Ein Stack wächst in eine Richtung: von unten nach oben oder von oben nach unten (üblich!)
 - Im Prinzip äquivalent

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 26/73

Stapelzeiger: belegt oder frei?

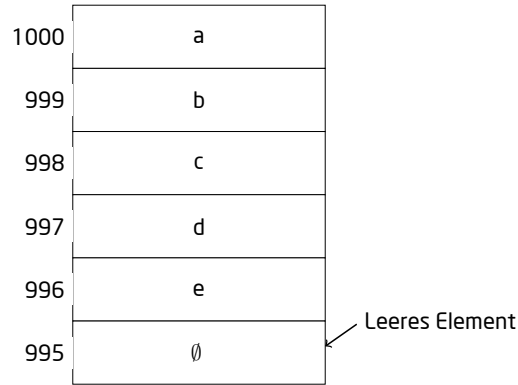


Abbildung 10.2: Stack, von oben nach unten wachsend, leeres Element unten, Stapelzeiger zeigt auf erstes leeres Byte

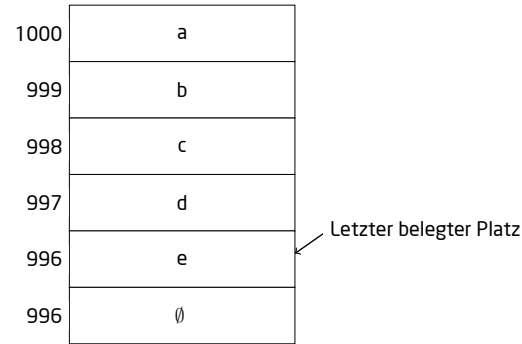


Abbildung 10.3: Stack, von oben nach unten wachsend, leeres Element unten, Stapelzeiger zeigt auf letztes belegtes Byte

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

GDS 10: Unterprogrammaufrufe

H. Karl, WS 22/23

Folie 27/73

Stack: Richtung

Stacks werden manchmal von oben nach unten, manchmal von unten nach oben gezeichnet/im Speicher abgelegt

- Kein Unterschied im Prinzip; top-down für Rechnerarchitektur verbreiteter

Stack von unten nach oben

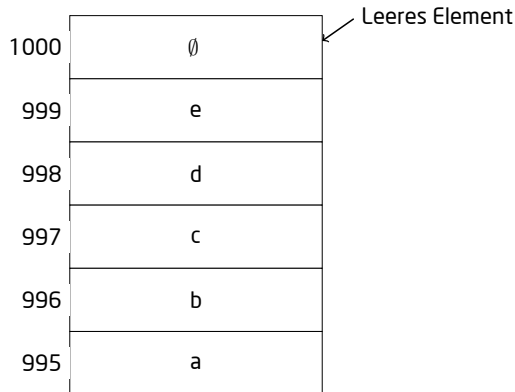


Abbildung 10.4: Stack, von unten nach oben wachsend, leeres Element oben

Stack von oben nach unten: Üblich!

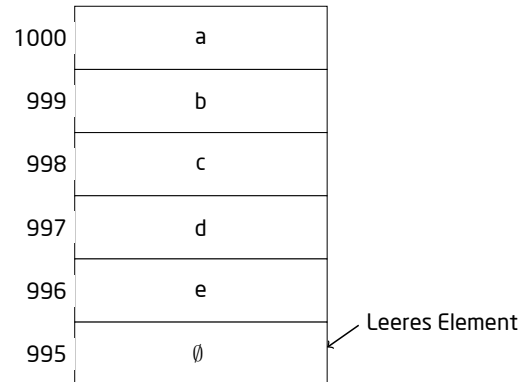


Abbildung 10.5: Stack, von oben nach unten wachsend, leeres Element unten - für Rechnerarchitektur üblich

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 28/73

Stack-Richtung und Stapelzeiger: Kombination für RISC-V

In RISC-V verwendet: Stapel von oben nach unten; Stapelzeiger zeigt auf niedrigste belegte Adresse

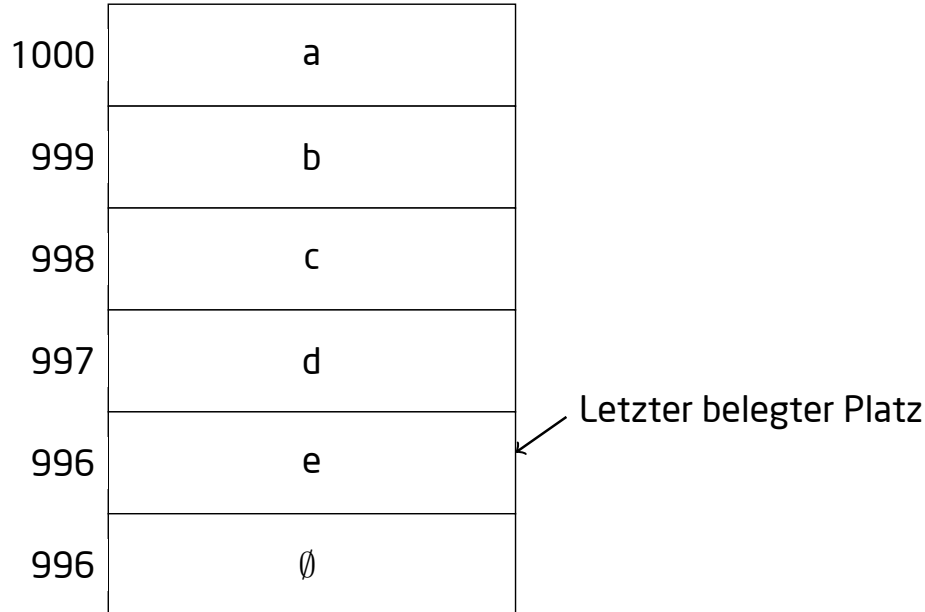


Abbildung 10.6: Stack, von oben nach unten wachsend, leeres Element unten, Stapelzeiger zeigt auf letztes belegtes Byte

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 29/73

Aufrufe mittels Stack

Grundidee:

- Bei Aufruf: Parameter und Rücksprungadresse auf Stack legen
 - Unterprogramm kann diese benutzen
- Zum Ende des Unterprogramms:
 - Nicht mehr benötigte Werte von Stack entfernen
 - Das sind die **obersten** Werte im Stack!
 - Rücksprungadresse nutzen

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 30/73

Aufrufe mittels Stack: Illustration

Vor Aufruf

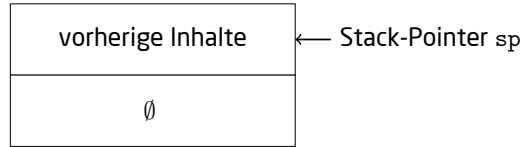


Abbildung 10.7: Nach unten wachsender Stack vor Aufruf eines Unterprogramms

Nach Aufruf, in Unterprogramm

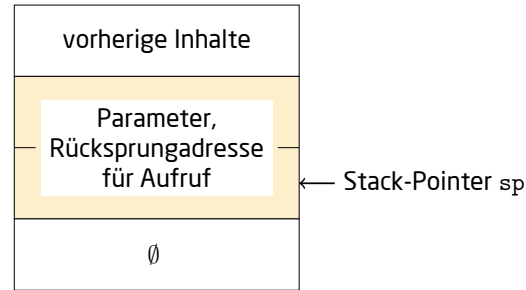


Abbildung 10.8: Nach unten wachsender Stack nach Aufruf, während Ablauf eines Unterprogramms

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 31/73

Aufrufe mittels Stack: Variablen des Unterprogramms?

Was macht man mit Variablen des Unterprogramms?

- Auf den Stack legen!
 - Durch Assembler, Compiler
 - Beliebig viel Platz
 - Am Ende, vor Rückkehr zu Aufrufer: vom Stack entfernen!

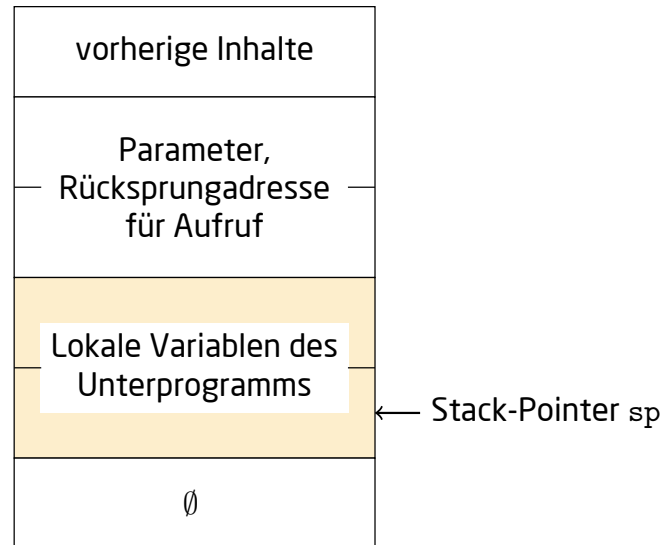


Abbildung 10.9: Variablen des Unterprogramms liegen ebenfalls auf Stack

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

GDS 10: Unterprogrammaufrufe

H. Karl, WS 22/23

Folie 32/73

Aufrufe mittels Stack: Rekursion?

Funktioniert das mit direkter und indirekter Rekursion? Ja!

- Bei Aufruf eines weiteren Unterprogramms: Gleiches Vorgehen
- Nach Ende des aufgerufenen Unterprogramms: Keine Überbleibsel auf Stack!

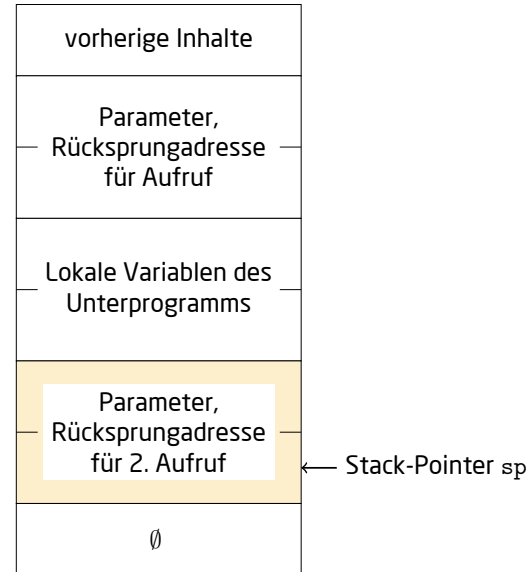


Abbildung 10.10: Verschachtelter Aufruf eines Unterprogramms funktioniert analog

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in
RISC-V

Zusammenfassung
Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 33/73

Erwartungen des Aufrufers?

- Ergebnis steht an vereinbarter Stelle im Stack
- Davon abgesehen: **Stapel** ist unverändert!
 - Unterprogramm darf Stapel während Ablauf **erweitern**, aber nicht auf vorhandene Stapelinhalte zugreifen
- **Register** sind unverändert!
 - Aber was, wenn Unterprogramm selbst Register braucht?
- **Speicherinhalte** sind unverändert
 - Programmierung: Keine Seiteneffekte
 - Ausnahmen für "call by reference" Parameter; hängt stark von Hochsprache/Programmiermodell ab
- **Einzige** Auswirkung: Ergebnis an vereinbarter Stelle

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 34/73

Unterprogrammaufruf: Grobkonzept

Fassen wir zusammen: Unterprogrammaufruf erfolgt in folgenden Schritten:

1. Aufrufer legt Parameter, Rücksprungadresse auf Stapel
2. Sprung zum Beginn des Unterprogramms
3. Bei Bedarf für Variablen des Unterprogramms Platz auf Stack nutzen
4. Eigentlichen Code des Unterprogramms ausführen
5. Rückgabewert im Stack ablegen, um für Aufrufer zugänglich zu machen
6. Sprung an Rücksprungadresse; Kontrolle geht an aufrufenden Programmteil zurück

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 35/73

Inhaltsverzeichnis

1. Unterprogramme

2. Aufrufe

2.1 Strohänner

2.2 Stack

2.3 Frame

2.4 Ablauf mit Stapel: Details

3. Unterprogramme in RISC-V

4. Zusammenfassung

5. Material

Unterprogramme

Aufrufe

Strohänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 36/73

- Aus Grobkonzept ergibt sich: Für jede Ausführung eines Unterprogramms gibt es einen **relevanten** Teil des Stacks: Der **frame**
 - Mit Rücksprung, Parameter, lokalen Variablen

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 37/73

Definition 10.6 (Frame)

Der Frame eines Unterprogrammaufrufs ist der Teil des Stack mit den für **diesen** Unterprogrammaufruf relevanten Daten.

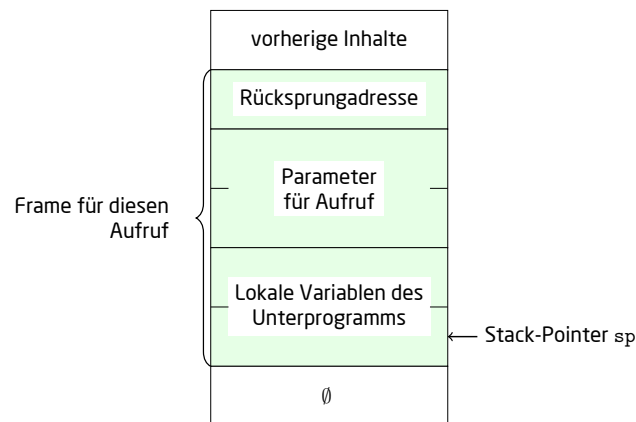


Abbildung 10.11: Frame für einen Unterprogrammaufruf

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 38/73

Frame für indirekte Rekursion

Beispiel-Code

```
int f (int a) {
    int d=a+1;
    return d;}

int g (int b) {
    int c = b+3;
    return 2*f(c);}

void main () {
    g(5);
}
```

Frame

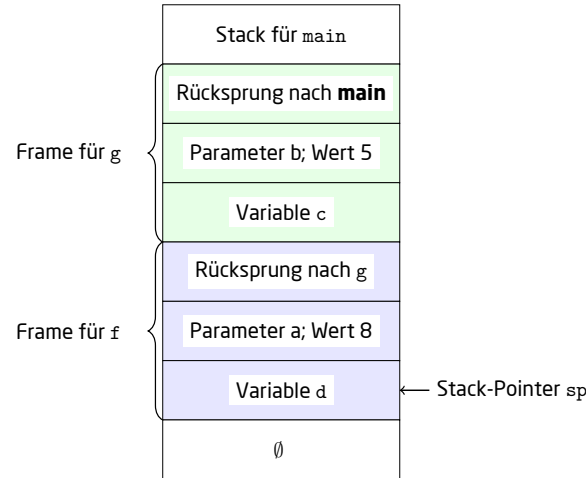


Abbildung 10.12: Frame für zwei verschachtelte Unterprogrammaufrufe

Unterprogramme

Aufrufe

Strohmann

Stack

Frame

Ablauf mit Stapel: Details

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 39/73

Inhaltsverzeichnis

1. Unterprogramme

2. Aufrufe

2.1 Strohmänner

2.2 Stack

2.3 Frame

2.4 Ablauf mit Stapel: Details

3. Unterprogramme in RISC-V

4. Zusammenfassung

5. Material

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in
RISC-V

Zusammenfassung
Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 40/73

Ablauf mit Stapel?

- Wie funktioniert nun ein Unterprogrammaufruf im Detail
 - Wenn wir nur einen Stapel benutzen?

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 41/73

Ablauf mit Stapel?

- Wie funktioniert nun ein Unterprogrammaufruf im Detail
 - Wenn wir nur einen Stapel benutzen?
- Beispiel folgt
- Das ist **nicht**:
 - das typische RISC-V Vorgehen, aber Grundlage dafür
 - das einzig mögliche Vorgehen, aber ein typisches Vorgehen für eine Architektur nur mit Stack

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 41/73

Konstruktion frame, vor Aufruf (Pseudo-Assembler, nicht RISC-V)

■ Aufrufer:

- `push rücksprung (PC+const):`
Rücksprungadresse auf Stack
- `push Parameter`
 - Ggf. mehrfach, 1x pro Parameter
 - `const` ergibt sich aus Anzahl `push` (genauer: gesamte Anzahl Bytes)

■ Aufruf: `jump` zum Start des Unterprogramms

```
aufrufer:
    push rücksprungAdr
    push param1
    push param2
    jump subroutine

rücksprung:
    pop resultatRegister
    pop null # param2
    pop null # param1
    pop null # rücksprungAdr

# somewhere far, far away:
subroutine:
    # eigentlicher Code
    # ....
    # Rucksprung:
    push resultat
    jump (sp + 3*4)
```

Abbildung 10.13: Pseudocode für Unterprogrammaufruf: Aufruf vorbereiten

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in RISC-V

Zusammenfassung

Material

GDS 10: Unterprogrammaufrufe

H. Karl, WS 22/23

Folie 42/73

Unterprogrammablauf (Pseudo-Assembler, nicht RISC-V)

- Relativ zum Stack Pointer ist die Lage der Variablen, Parameter fix!
 - im Beispiel-Code: alle Adressen, Variablen 4 Bytes
- Dadurch Zugriff möglich
 - Adressierung relativ zu `sp`, mit konstantem Offset
 - Z.B. in Register laden

```
aufrufer:
    push rücksprungAdr
    push param1
    push param2
    jump subroutine

rücksprung:
    pop resultatRegister
    pop null # param2
    pop null # param1
    pop null # rücksprungAdr

# somewhere far, far away:
subroutine:
    # eigentlicher Code
    # ....
    # Rücksprung:
    push resultat
    jump (sp + 3*4)
```

Abbildung 10.14: Pseudocode für Unterprogrammaufruf:
Eigentlicher Ablauf des Unterprogramms

Unterprogramme

Aufrufe

Strohmann
Stack
Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in RISC-V

Zusammenfassung

Material

GDS 10: Unterprogrammaufrufe

H. Karl, WS 22/23

Folie 43/73

Unterprogramm-Ende, vor Rücksprung (Pseudo-Assembler, nicht RISC-V)

- Ergebnis auf Stack legen
- Rücksprung: Adresse relativ zum `sp`
 - hier: 4 Einträge auf Stack (Ergebnis, Parameter 2, Parameter 1, Rücksprungadresse) à 4 Byte
 - Also Rücksprungadresse bei `sp + 3* 4 Bytes`

```
aufrufer:
    push rücksprungAdr
    push param1
    push param2
    jump subroutine

rücksprung:
    pop resultatRegister
    pop null # param2
    pop null # param1
    pop null # rücksprungAdr

# somewhere far, far away:
subroutine:
    # eigentlicher Code
    # ....
    # Ruecksprung:
    push resultat
    jump (sp + 3*4)
```

Abbildung 10.15: Pseudocode für
Unterprogrammaufruf: Vorbereitung des Rücksprungs

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 44/73

Nach Rückkehr (Pseudo-Assembler, nicht RISC-V)

- Nach Rückkehr aus Unterprogramm:
Aufrufer muss Stack aufräumen
 - Ergebnis an gewünschte Stelle kopieren (z.B. Register)
 - Ergebnis, Rücksprungadresse und Parameter von Stack entfernen (also `sp` entsprechend erhöhen)

```
aufrufer:
    push rücksprungAdr
    push param1
    push param2
    jump subroutine

rücksprung:
    pop resultatRegister
    pop null # param2
    pop null # param1
    pop null # rücksprungAdr

# somewhere far, far away:
subroutine:
    # eigentlicher Code
    # ....
    # Rücksprung:
    push resultat
    jump (sp + 3*4)
```

Abbildung 10.16: Pseudocode für Unterprogrammaufruf:
Aufrufer räumt nach Rückkehr auf

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 45/73

Vorgehen nicht eindeutig!

- Geschildertes Vorgehen funktioniert, aber nicht das einzig mögliche Vorgehen
- Viel Entscheidungsspielraum, z.B.:
 - ☐ mehrere Parameter von links nach rechts o.u. auf Stack legen?
 - ☐ erst Parameter, dann Rücksprungadresse?
 - ☐ Insbes.: Aufteilung Aufgaben zwischen aufrufendem und aufgerufenem Programmteil?

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 46/73

Problem: Aufwand

- Vorteil des Verfahrens: Es funktioniert, auch wenn **nur** ein Stack zur Verfügung steht 😊
- Nachteil: Sehr aufwändig, viele Speicherzugriffe
 - Bsp. Parameter:
 - Standen vor Aufruf vermutlich in Register (load/store!)
 - Werden im Unterprogramm vermutlich verarbeitet
 - Müssen dazu also wieder in Register geladen werden
 - Bsp. Rücksprungadresse:
 - Wenn aufgerufenes Unterprogramm direkt zurückkehrt hätte auch die in Register stehen können

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 47/73

Problem: Aufwand

- Vorteil des Verfahrens: Es funktioniert, auch wenn **nur** ein Stack zur Verfügung steht 😊
- Nachteil: Sehr aufwändig, viele Speicherzugriffe
 - Bsp. Parameter:
 - Standen vor Aufruf vermutlich in Register (load/store!)
 - Werden im Unterprogramm vermutlich verarbeitet
 - Müssen dazu also wieder in Register geladen werden
 - Bsp. Rücksprungsadresse:
 - Wenn aufgerufenes Unterprogramm direkt zurückkehrt hätte auch die in Register stehen können
- Wir kopieren also sinnlos Register-Speicher-Register 🤖
- Optimierung???
- Stack nur benutzen wenn nötig?

Unterprogramme

Aufrufe

Strohmänner

Stack

Frame

[Ablauf mit Stapel: Details](#)

Unterprogramme in

RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 47/73

Inhaltsverzeichnis

1. Unterprogramme

2. Aufrufe

3. Unterprogramme in RISC-V

3.1 Konvention

3.2 Rücksprung via Register

3.3 Parameter via Register

4. Zusammenfassung

5. Material

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 48/73

Unterprogramme, Stapel bei RISC-V

- Das Konzept bisher wird so oder sehr ähnlich bei allen modernen Rechnerarchitekturen verfolgt

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 49/73

Unterprogramme, Stapel bei RISC-V

- Das Konzept bisher wird so oder sehr ähnlich bei allen modernen Rechnerarchitekturen verfolgt
- Werden wir hier konkret: Wie funktioniert es **speziell bei RISC-V**?
- Grundidee: RISC-V ist kein reines Stapel-System, wir nutzen Register intensiv!

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 49/73

Unterprogramme, Stapel bei RISC-V

- Das Konzept bisher wird so oder sehr ähnlich bei allen modernen Rechnerarchitekturen verfolgt
- Werden wir hier konkret: Wie funktioniert es **speziell bei RISC-V**?
- Grundidee: RISC-V ist kein reines Stapel-System, wir nutzen Register intensiv!
- Zuvor: Ein paar Konventionen

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 49/73

Stapelrichtung, Speicheraufteilung

- RISC-V: per Konvention wächst der Stapel von oben nach unten
 - Oben: große (größte?) verfügbare Speicheradresse
 - Unten: Kleine Speicheradresse: Programm (das `.text`-Segment!), statische Daten (und Heap)

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 50/73

Register für Stapelzeiger

- Bisher: Register ohne besondere Zweckbindung
- Mit Stapelzeiger: ein Register mit Sonderrolle für Unterprogrammaufrufe
- Welches?
 - Jeder Programmierer sucht sich ein eigenes aus?
 - Allgemeine Konvention - erleichtert/ermöglicht Wiederverwendung von Programmen!

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 51/73

Register für Stapelzeiger

- Bisher: Register ohne besondere Zweckbindung
- Mit Stapelzeiger: ein Register mit Sonderrolle für Unterprogrammaufrufe
- Welches?
 - ☐ Jeder Programmierer sucht sich ein eigenes aus?
 - ☐ Allgemeine Konvention - erleichtert/ermöglicht Wiederverwendung von Programmen!
- RISC-V Konvention: Stapelzeiger ist in Register `x2`
 - ☐ Und bekommt sogar einen Alias: `sp`!

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 51/73

Konvention

Definition 10.7 (Konvention)

Die Festlegung einer üblichen Praxis. Eine Konvention ist durch eine ISA nicht erzwungen; innerhalb einer ISA hätte eine Konvention auch anders festgelegt werden können.

Die Einhaltung von Konventionen ist hilfreich, um z.B. separat entwickelte (oder übersetzte) Programme zusammenarbeiten zu lassen oder um Lesbarkeit von Programmen zu erhöhen

Beispiel 10.1 (Konvention: ja oder nein?)

- Endianness?
- Stack wächst nach unten?
- Register $x2$ ist Stapelzeiger?

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 52/73

Inhaltsverzeichnis

1. Unterprogramme

2. Aufrufe

3. Unterprogramme in RISC-V

3.1 Konvention

3.2 **Rücksprung via Register**

3.3 Parameter via Register

4. Zusammenfassung

5. Material

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 53/73

Konvention: Rücksprungadresse

- Bisher: Rücksprungadresse liegt auf Stack
 - Unvermeidlich, wenn das Unterprogramm selbst wieder Aufrufe durchführt

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 54/73

Konvention: Rücksprungadresse

- Bisher: Rücksprungadresse liegt auf Stack
 - Unvermeidlich, wenn das Unterprogramm selbst wieder Aufrufe durchführt
- Was, wenn aus Unterprogramm keine weiteren Aufrufe?
 - Sog. **leaf procedure**
 - Kann dann Rücksprungsadresse in Register stehen?

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 54/73

Konvention: Rücksprungadresse

- Bisher: Rücksprungadresse liegt auf Stack
 - Unvermeidlich, wenn das Unterprogramm selbst wieder Aufrufe durchführt
- Was, wenn aus Unterprogramm keine weiteren Aufrufe?
 - Sog. **leaf procedure**
 - Kann dann Rücksprungsadresse in Register stehen?
- Ja!
 - Wie kann ISA das unterstützen?

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 54/73

Rücksprungadresse in Register

- Idee: Sprungbefehl macht **zwei** Dinge
 - **Nächsten** Wert des PCs (ohne Sprung) in ein Register schreiben
 - Sprung an angegeben Stelle
- RISC-V: `jal jump-and-link`
 - Konvention: Register `x1` für Rücksprungadresse nutzen
 - Alias: `ra` (return address) für `x1`

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 55/73

Rückkehr aus Unterprogramm mit jalr

- Pendant: jump-and-link register jalr
 - Wie jal, nächsten PC in ein Register schreiben
 - Anders als jal: Sprungziel steht in (anderem) Register

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 56/73

Rückkehr aus Unterprogramm mit jalr

- Pendant: jump-and-link register jalr
 - Wie jal, nächsten PC in ein Register schreiben
 - Anders als jal: Sprungziel steht in (anderem) Register
- Anwendung für Rücksprung aus Unterprogramm:
 - Rücksprungadresse steht in ra
 - Aktueller PC wird nicht mehr gebraucht (nach x0 schreiben)
 - Also: jalr x0, x1, 0
 - Effekt: Wir kehren aus Unterprogramm zurück

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 56/73

Wissen bei Aufrufer, aufgerufenem Code?

- Kann der **Aufrufer** wissen, ob das aufgerufene Programmteil weitere Aufrufe macht?
 - ☐ Nein!
 - ☐ Nur das aufgerufene Unterprogramm selbst kann das wissen
- Funktioniert Mechanismus mit `jal` noch?
 - ☐ Ja, aber Register `ra` würde jetzt ja mehrfach benutzt
 - ☐ Also: Unterprogramm muss selbst Rücksprungadresse auf Stack legen, falls es selbst Aufrufe durchführen wird
 - Inhalt von `ra` wird gerettet
 - Dadurch ändert sich die Verantwortung, Reihenfolge der Stack-Nutzung!
- Aufrufer ist quasi optimistisch: Rücksprungadresse-auf-Stack ist Verantwortung des Unterprogramms

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 57/73

Beispiel: Verschaltelte Aufrufe

```

1  main:
2      # main code here
3      jal ra, f          # alias: jal f
4      # more main code here
5
6  f:
7      # f code here
8      # save ra on stack (4 Byte adress)
9      addi sp, sp, -4
10     sw ra, 0(sp)
11     # call g (overrides ra)
12     jal ra, g          # alias: jal g
13     # other f code here
14     # restore ra
15     lw ra, 0(sp)
16     addi sp, sp, 4
17     # more f code here
18     # return to caller
19     jalr zero, ra, 0    # alias: ret
20
21  g:
22     # g code here
23     # return to caller
24     jalr zero, ra, 0    # alias: ret

```

- Hauptprogramm main ruft f auf (Rücksprung nach Zeile 4)
- f ruft g auf
 - f muss die eigene Rücksprungadresse (abgelegt in ra) auf dem Stack sichern, da Aufruf von g diese überschreibt
 - Zeilen 9 und 10 sichern ra, schaffen dazu Platz auf Stack (addi sp, sp -4)
 - Zeilen 15 und 16 stellen die Rücksprungadresse für Funktion f wieder her (also Zeile 4)
- g ruft kein anderes Unterprogramm (leaf)

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

GDS 10: Unterprogrammaufrufe

H. Karl, WS 22/23

Folie 58/73

Inhaltsverzeichnis

1. Unterprogramme

2. Aufrufe

3. Unterprogramme in RISC-V

3.1 Konvention

3.2 Rücksprung via Register

3.3 Parameter via Register

4. Zusammenfassung

5. Material

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 59/73

Parameter in Register

- Erinnerung (Folie 47): Aufwandsproblem war unnötiges Kopieren von Register-Stack-Register
- Idee: Wenn Parameter schon in Register steht, dann bleibt der da einfach!
 - Gar keine Interaktion mit Stack!
- Ergebnisse: entsprechend

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 60/73

Parameter in Register: Mögliches Vorgehen

- Erinnerung (Folie 34) Erwartungshaltung Aufrufer: Alle Register bleiben unverändert
- Also Idee für Vorgehen:
 - **Aufrufer** kümmert sich nicht um Stack, packt alle Parameter in Register
 - (Ausnahme: was, wenn mehr Parameter als Register . . . ?)
 - **Aufgerufenes Unterprogramm** sichert ein Register auf Stack vor Benutzung
 - Holt vor Rücksprung Inhalt wieder von Stack, stellt Registerinhalt wieder her

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 61/73

Parameter in Register: Mögliches Vorgehen

- Erinnerung (Folie 34) Erwartungshaltung Aufrufer: Alle Register bleiben unverändert
- Also Idee für Vorgehen:
 - **Aufrufer** kümmert sich nicht um Stack, packt alle Parameter in Register
 - (Ausnahme: was, wenn mehr Parameter als Register . . . ?)
 - **Aufgerufenes Unterprogramm** sichert ein Register auf Stack vor Benutzung
 - Holt vor Rücksprung Inhalt wieder von Stack, stellt Registerinhalt wieder her
- Vorteil: Nur wirklich vom Unterprogramm benötigte Register müssen auf Stack kopiert werden
- Nachteil: Relativ kompliziert (welches Register wird wo im Stack zwischengelagert, . . .)

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 61/73

Anforderungen lockern

- Und selbst das braucht man in Praxis nicht unbedingt
- Viele Register enthalten ohnehin nur temporäre Daten, die nicht weiter benutzt werden
 - Sichern auf Stack unnötiger Aufwand

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 62/73

Anforderungen lockern

- Und selbst das braucht man in Praxis nicht unbedingt
- Viele Register enthalten ohnehin nur temporäre Daten, die nicht weiter benutzt werden
 - Sichern auf Stack unnötiger Aufwand

Pragmatische Lösung: Wir unterscheiden Register

- Manche Register dürfen von Unterprogramm überschrieben werden (x5 - x7, x28 - x31)
 - Konsequenz: Inhalte dieser Register nach Rückkehr aus Aufruf **undefiniert**
- Andere Register muss das aufgerufene Unterprogramm sichern; bei Benutzung vorher auf Stack legen und wieder herstellen (x8, x9, x18 - x27)
 - Sog. **register spill**

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 62/73

Stackaufbau bei RISC-V

- Mit Register für Rücksprung, Parameter verändert sich auch Stack-Aufbau, Verantwortlichkeit
 - Beides im Stack nur bei Bedarf
 - Damit Reihenfolge innerhalb eines Stackframes nicht mehr eindeutig!

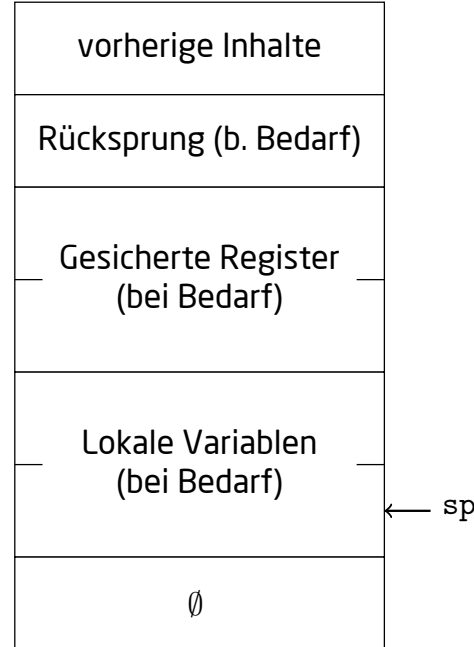


Abbildung 10.17: Stack bei RISC-V, höhere Variabilität durch Nutzung von Registern

Unterprogramme
 Aufrufe
 Unterprogramme in RISC-V
 Konvention
 Rücksprung via Register
 Parameter via Register
 Zusammenfassung
 Material

- Mit Programmcode, statischen Daten, Stack haben wir drei sehr verschiedenen Arten der Speichernutzung
 - Code, statische Daten: konstante Größe
 - Stack: Wächst und schrumpft, je nach Unterprogrammverschachtelung
- Mit Heap kommt später noch eine vierte Art hinzu
- Strukturierter Speicheraufbau bietet sich für diese vier Segmente an

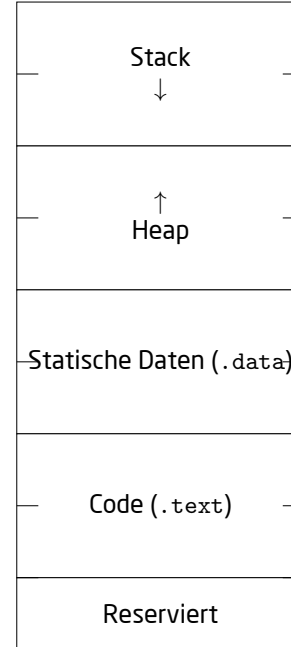


Abbildung 10.18: Typisches Layout der vier Segmente eines laufenden Programms

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

GDS 10: Unterprogrammaufrufe

H. Karl, WS 22/23

Folie 64/73

Konventionen für Register, Aliase

Vgl. [PH20, Abschnitt 2.8] (insbes. Abb. 2.11) oder [Wat+17, Tab. 18.2, p. 85]

Tabelle 10.1: Zusammenfassung Register-Konventionen bei RISC-V

Register	Alias	Verwendung	Sicherung durch
x0	zero	Immer 0	--
x1	ra	Rücksprungadresse	Aufrufer
x2	sp	Stack Pointer	Aufrufer
x5 - x7, x28 - x31	t0 - t6	Temporäre Register (darf Unterprogramm beliebig nutzen)	Aufrufer
x8, x9, x18 - x27	s0 - s11	Über Aufrufe hinweg erhalten (sog. "saved registers")	Unterprogramm
x10 - x17	a0 - a7	Unterprogramm Parameter / Ergebnisse	Aufrufer

Unterprogramme

Aufrufe

Unterprogramme in

RISC-V

Konvention

Rücksprung via Register

Parameter via Register

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 65/73

Inhaltsverzeichnis

1. Unterprogramme

2. Aufrufe

3. Unterprogramme in RISC-V

4. Zusammenfassung

5. Material

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 66/73

- Unterprogramme sind unverzichtbar für komplexe Programme und müssen effizient durch eine ISA unterstützt werden
- Grundmechanismus: Stack
 - Mit diversen Varianten; wichtig ist korrektes Zusammenspiel von Aufrufer und aufgerufenem Programmteil
- Effizienzsteigerung: Register benutzen
 - Insbes. für Rücksprungadresse, Parameter, Ergebnisse
 - Aber auf Stack zurückgreifen:
 - Bei Platzmangel (mehr Parameter als Register)
 - Wenn verschachtelte Aufrufe die gleichen Register benötigen
 - Verschiebt Aufgabenverteilung zwischen Aufrufer und aufgerufenem Programm!
 - Macht Konventionen wichtiger!

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung
Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 67/73

Inhaltsverzeichnis

- 1. Unterprogramme
- 2. Aufrufe
- 3. Unterprogramme in RISC-V
- 4. Zusammenfassung
- 5. Material

Unterprogramme
Aufrufe
Unterprogramme in
RISC-V
Zusammenfassung
Material

- [22] *RISC-V ABIs Specification*. <https://github.com/riscv/riscv-elf-psabi-doc/>. v. 1.0-rc4. Sep. 2022.
- [Dij68] W. Dijkstra. "Goto statement Considered Harmful". In: *CACM* 11 (1968), S. 125-133.
- [PH20] David A. Patterson und John L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. 2nd edition. Morgan Kaufmann Publishers Inc., 2020. ISBN: 9780128203316.
- [Wat+17] Andrew Waterman u. a. *The RISC-V Instruction Set Manual: Volume I User-level ISA*, v 2.2. 2017.

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 69/73

Abkürzungen I

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 70/73

Frame Ein Frame ist, im Kontext des Aufrufs von Unterprogramms, ein Teil des Stacks. Der Frame beinhaltet alle Daten, die zur Ausführung eines Unterprogramms notwendig sind, insbes. die Rücksprungadresse zum aufrufenden Programm, Parameter, die an das Unterprogramm übergeben wurden, sowie Variablen des Unterprogramms selbst. Der genaue Aufbau des Frames hängt von der benutzten Aufrufkonvention ab und ist nicht einheitlich festgelegt. 40

Instruction Set Architecture Die Schnittstelle zwischen Hardware und der niedrigsten Schicht von Software. Sie enthält alle Information darüber, welche Semantik jede Instruktion hat und wie die Hardware beschaffen ist (unveränderliche oder variable Aspekte, etwa Anzahl Register). 2

register spill Auslagern von Register-Inhalten von den Stack, z.B. wenn ein Register in einem Unterprogramm gebraucht wird aber vom Aufrufer als unverändert

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 71/73

erwartet wird. 73, 74

Stack Eine Datenstruktur beliebiger Größe, mit Elementen fester oder variabler Größe. Bei einem Stack sind mindestens zwei Operationen definiert: `push`, um ein Element auf den zu legen, und `pop`, um das zuletzt hinzugefügte Element zu entnehmen. 25

Stapelzeiger Der Stapelzeiger (stack pointer) zeigt auf das erste leere Element eines Stacks (manchmal: auf das letzte belegte Element). Häufig ist der Stapelzeiger in eine ISA tief integriert und wird durch ein dediziertes Register repräsentiert sowie durch explizite Instruktionen manipuliert. 26--28

Unterprogramm Eine Code-Sequenz, die aus anderen Code-Teilen heraus aufgerufen und dadurch mehrfach benutzt werden kann. Im allgemeinen haben Unterprogramme formale Parameter, die beim Aufruf mit konkreten Werten versorgt werden; am Ende der Ausführung des Unterprogramms kann ein

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 72/73

Ergebnis für den aufrufenden Programmteil bereitgestellt werden. Detail dieser Aufrufsemantik können sich je nach Programmiersprache unterscheiden. 9

Unterprogramme

Aufrufe

Unterprogramme in
RISC-V

Zusammenfassung

Material

**GDS 10: Unterpro-
grammaufrufe**

H. Karl, WS 22/23

Folie 73/73