Funktionen

Prof. Dr. Christian Becker

Universität Stuttgart, Institut für Parallele und Verteilte Systeme

15. Mai 2025

Inhalt dieser Vorlesung

Einführung

Sprünge

Funktionen

Calling Conventions

Inhalt dieser Vorlesung

Wiederholung Lec. 5

If statements und Loops wiederholen

Einführendes Beispiel

Beispiel (abs function, aber ohne function bisher)

```c
int a = ?;
int result;
if (a < 0) {
    result = -a;
} else {
    result = a;
}
```

Idee: Konzept von jump vermitteln

Inhalt dieser Vorlesung

Einführung

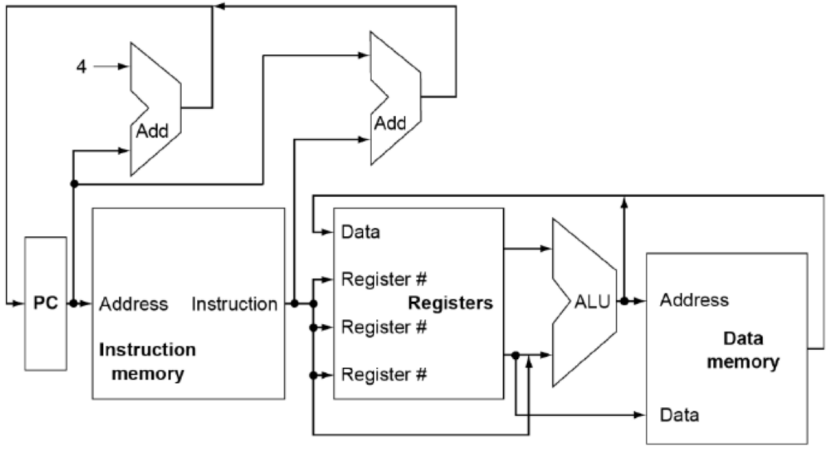
Sprünge

Funktionen

Calling Conventions

Program Counter

PC genauer erklären => Es gibt jumps



Beispiel (abs function, aber ohne function bisher)

```asm
    # ...
    lw      a5,12(sp)          # Lade a5 von Addr. von a
    bge     a5,zero,.L2        # if a5 >= 0 springe zu L2
    # if (a < 5)
    neg     a5,a5              # a5 = -a5
    sw      a5,28(sp)          # Speichere a5 in Addr. von result
    j       .L3                # Springe zu L3
.L2:
    # else
    lw      a5,12(sp)          # Lade a5 von Addr. von a
    sw      a5,28(sp)          # Speichere a5 in Addr. von result
.L3:
    # Ende if-statement
    lw      a5,28(sp)          # Lade result in a5
    # ... return result
```

Asm

Simplified compiler output

Idee: Konzept von jump vermitteln

- Sprünge im Program Counter einzeichnen
- Code in Simulator ausführen und zeigen (abgeänderte Variante die a davor speichert)

Inhalt dieser Vorlesung

Einführung

Sprünge

Funktionen

Calling Conventions

Warum Programme nur am Stück schreiben schlecht?

- Programme werden schnell unübersichtlich
- Wir wollen bestimmte Abschnitte wiederverwenden
- ...

=> wow, es gibt Funktionen

Beispiel Funktionen

Beispiel (abs function mit function call)

```c
void mainProgram() {
    int x = 2;
    int y = -5;
    int absX = abs(x);
    int absY = abs(y);
}

int abs(int a) {
    int result;
    if (a < 0) {
        result = -a;
    } else {
        result = a;
    }
    return result;
}
```

C

Code oben erklären, output zeigen, ...

Woher weiß Programm, an welche Stelle im Memory es schreiben darf?
=> Es gibt einen Stack

Stack Konzept mit Stack Pointer erklären
Frame Pointer lassen wir weg für simplicity (gcc compiler option: `-fomit-frame-pointer`)
(Stack Pointer in RISC-V is 16 Byte aligned)

Function Calls in RISC-V

main

```asm
.mainProgram:
  addi    sp,sp,-32
  sw      ra,28(sp)
  li      a5,2
  sw      a5,12(sp)
  li      a5,-5
  sw      a5,8(sp)
  lw      a0,12(sp)
  jal     .abs
  sw      a0,4(sp)
  lw      a0,8(sp)
  jal     .abs
  sw      a0,0(sp)
  lw      ra,28(sp)
  addi    sp,sp,32
  jr      ra
```

Asm

abs

```asm
.abs:
  addi    sp,sp,-32
  sw      a0,12(sp)
  lw      a5,12(sp)
  bge     a5,zero,.L2
  lw      a5,12(sp)
  neg     a5,a5
  sw      a5,28(sp)
  j       .L3
.L2:
  lw      a5,12(sp)
  sw      a5,28(sp)
.L3:
  lw      a5,28(sp)
  mv      a0,a5
  addi    sp,sp,32
  jr      ra
```

Asm

https://godbolt.org/ compiler output kann man schön zeigen auch, man sieht farblich welches c statement zu welchem RISC-V statement gehört

- Beispiel von oben nach und nach erweitern
- Pfeile von jal zu .abs (ra register)
- Pfeile von jr zurück (ra register)
- Grafik mit Stack neben abs
- RISC-V Code in Simulator ausführen

Function Calls in RISC-V

main

```asm
.mainProgram:
    addi    sp,sp,-32
    sw      ra,28(sp)
    li      a5,2
    sw      a5,12(sp)
    li      a5,-5
    sw      a5,8(sp)
    lw      a0,12(sp)
    jal     .abs
    sw      a0,4(sp)
    lw      a0,8(sp)
    jal     .abs
    sw      a0,0(sp)
    lw      ra,28(sp)
    addi    sp,sp,32
    jr      ra
```

Asm

abs

```asm
.abs:
    addi    sp,sp,-32
    sw      a0,12(sp)
    lw      a5,12(sp)
    bge     a5,zero,.L2
    lw      a5,12(sp)
    neg     a5,a5
    sw      a5,28(sp)
    j       .L3
.L2:
    lw      a5,12(sp)
    sw      a5,28(sp)
.L3:
    lw      a5,28(sp)
    mv      a0,a5
    addi    sp,sp,32
    jr      ra
```

Asm

TODO: Anchors break spacing

Function Calls in RISC-V

main

```asm
.mainProgram:
    addi    sp,sp,-32
    sw      ra,28(sp)
    li      a5,2
    sw      a5,12(sp)
    li      a5,-5
    sw      a5,8(sp)
    lw      a0,12(sp)
    jal     .abs
    sw      a0,4(sp)
    lw      a0,8(sp)
    jal     .abs
    sw      a0,0(sp)
    lw      ra,28(sp)
    addi    sp,sp,32
    jr      ra
```

Asm

abs

```asm
.abs:
    addi    sp,sp,-32
    sw      a0,12(sp)
    lw      a5,12(sp)
    bge     a5,zero,.L2
    lw      a5,12(sp)
    neg     a5,a5
    sw      a5,28(sp)
    j       .L3
.L2:
    lw      a5,12(sp)
    sw      a5,28(sp)
.L3:
    lw      a5,28(sp)
    mv      a0,a5
    addi    sp,sp,32
    jr      ra
```

Asm

TODO: Anchors break spacing

Function Calls in RISC-V

main

```asm
.mainProgram:
    addi    sp,sp,-32
    sw      ra,28(sp)
    li      a5,2
    sw      a5,12(sp)
    li      a5,-5
    sw      a5,8(sp)
    lw      a0,12(sp)
    jal     .abs
    sw      a0,4(sp)
    lw      a0,8(sp)
    jal     .abs
    sw      a0,0(sp)
    lw      ra,28(sp)
    addi    sp,sp,32
    jr      ra
```

Asm

abs

```asm
.abs:
    addi    sp,sp,-32
    sw      a0,12(sp)
    lw      a5,12(sp)
    bge     a5,zero,.L2
    lw      a5,12(sp)
    neg     a5,a5
    sw      a5,28(sp)
    j       .L3
.L2:
    lw      a5,12(sp)
    sw      a5,28(sp)
.L3:
    lw      a5,28(sp)
    mv      a0,a5
    addi    sp,sp,32
    jr      ra
```

Asm

TODO: Anchors break spacing

Function Calls in RISC-V

main

```asm
.mainProgram:
    addi    sp,sp,-32
    sw      ra,28(sp)
    li      a5,2
    sw      a5,12(sp)
    li      a5,-5
    sw      a5,8(sp)
    lw      a0,12(sp)
    jal     .abs
    sw      a0,4(sp)
    lw      a0,8(sp)
    jal     .abs
    sw      a0,0(sp)
    lw      ra,28(sp)
    addi    sp,sp,32
    jr      ra
```

Asm

abs

```asm
.abs:
    addi    sp,sp,-32
    sw      a0,12(sp)
    lw      a5,12(sp)
    bge     a5,zero,.L2
    lw      a5,12(sp)
    neg     a5,a5
    sw      a5,28(sp)
    j       .L3
.L2:
    lw      a5,12(sp)
    sw      a5,28(sp)
.L3:
    lw      a5,28(sp)
    mv      a0,a5
    addi    sp,sp,32
    jr      ra
```

Asm

TODO: Anchors break spacing

Function Calls in RISC-V

main

```asm
.mainProgram:
  addi    sp,sp,-32
  sw      ra,28(sp)
  li      a5,2
  sw      a5,12(sp)
  li      a5,-5
  sw      a5,8(sp)
  lw      a0,12(sp)
  jal     .abs
  sw      a0,4(sp)
  lw      a0,8(sp)
  jal     .abs
  sw      a0,0(sp)
  lw      ra,28(sp)
  addi    sp,sp,32
  jr      ra
```

Asm

Hier kommt n Stack hin und es wird der Stackpointer verschoben und nach und nach der Stack mit Inhalt gefüllt

Function Calls in RISC-V

abs

```asm
.abs:
  addi    sp,sp,-32
  sw      a0,12(sp)
  lw      a5,12(sp)
  bge     a5,zero,.L2
  lw      a5,12(sp)
  neg     a5,a5
  sw      a5,28(sp)
  j       .L3
.L2:
  lw      a5,12(sp)
  sw      a5,28(sp)
.L3:
  lw      a5,28(sp)
  mv      a0,a5
  addi    sp,sp,32
  jr      ra
```

Asm

Hier kommt n Stack hin und es wird der Stackpointer verschoben und nach und nach der Stack mit Inhalt gefüllt

Inhalt dieser Vorlesung

Einführung

Sprünge

Funktionen

Calling Conventions

TODO: Lukas, Simon K.
Ich (Lucas) kann bei Bedarf gerne weitere RISC-V Beispiele beisteuern.