CPU and Memory

Prof. Dr. Christian Becker

Universität Stuttgart, Institut für Parallele und Verteilte Systeme

15. Mai 2025

Inhalt dieser Vorlesung

Einführung

Datenpfad: CPU

Datenpfad: Memory

Pointer

Inhalt dieser Vorlesung

Einführung

Datenpfad: CPU

Datenpfad: Memory

Pointer

Wiederholung Lec. 2

Wiederholung der Datenrepräsentation.
Evtl. macht es Sinn Lecture 3 und 4 zu tauschen. Es fällt mir schwer das Konzept von memory mit assembly und compiler zu erklären ohne jeglichen Code
Hier wäre wichtig zu wissen, was genau Teil davon ist, weil ohne zu wissen was z.B. ein int ist wird der Teil schwierig.

Einführendes Beispiel

Schön wäre es sich an einem Beispiel wie diesem hier den Datenpfad aufzubauen. Wenn wir hier noch keinen C-Code haben wollen wäre auch Pseudocode eine Option, aber ich würde behaupten C-Code auf diesem Level ist fast Pseudocode.

```c
int x = 5;
int y = 3;
int z = x + y;
```

C

Inhalt dieser Vorlesung

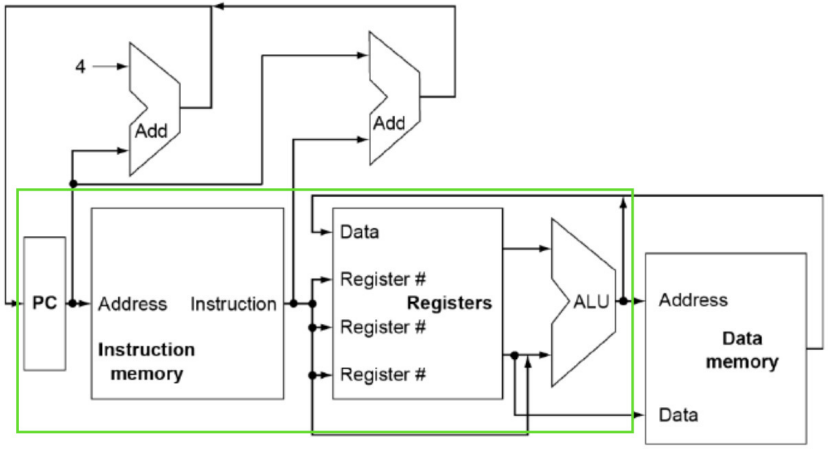
Einführung

Datenpfad: CPU

Datenpfad: Memory

Pointer

Datenpfad ohne Data memory, PC und den oberen Teil



Geklaut aus RO I, sollten wir selbst erstellen

- Komponenten erklären (PC, IM, Reg, ALU)

Einführendes Beispiel RISC-V

Stark vereinfachter RISC-V Code:

```asm
1  li      a4,5        # 5 in Register a4 laden (int x = 5)
2  li      a5,3        # 3 in Register a5 laden (int y = 3)
3  add     a6,a4,a5    # a6 = a4 + a5           (int z = x + y)
```

Asm

Lässt sich als showcase zeigen: `https://venus.kvakil.me/`
// TODO: Code formatting für riscv

- Asm Code Beispiel Schritt für Schritt in Datenpfad-Grafik einzeichnen und zeigen
- Code in Simulator ausführen

```c
int x = 5;
int y = 3;
// Sehr viel mehr Code
int z = x + y;
```

C

Begrenzte Anzahl an Registern
Oh nein, funktioniert nicht bei größeren Programmen
=> Wir brauchen Memory

Inhalt dieser Vorlesung

Einführung

Datenpfad: CPU

Datenpfad: Memory

Pointer

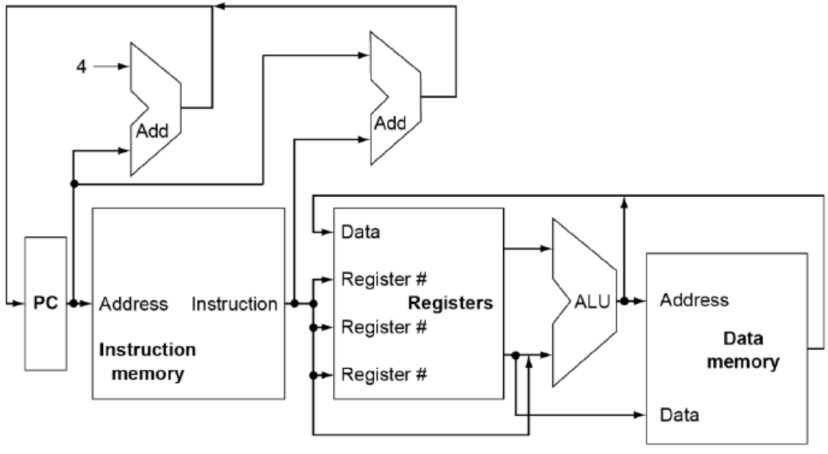
Memory

Aufbau Memory erklären Konzept von Adressen erklären

Address	+0	+1	+2	+3
0x00000030	00	00	00	00
0x0000002c	00	00	00	00
0x00000028	00	00	00	00
0x00000024	00	00	00	00
0x00000020	00	00	00	00
0x0000001c	23	2c	f1	fe
0x00000018	b3	07	f7	00
0x00000014	83	27	c1	ff
0x00000010	03	27	01	00
0x0000000c	23	2e	f1	fe
0x00000008	93	07	30	00
0x00000004	23	20	f1	00
0x00000000	93	07	50	00

Memory

Ohne die Adder oben, ich würde den PC hier als "magic"belassen und erst in Lecture 6 näher spezifizieren und zusammen mit Jumps einführen



- sw und lw erklären
- sp ist erstmal "magic-> Der Ort an dem wir unsere Daten hinspeichern können mit Offset, macht auch erst in Lecture 6 Sinn aber wir zeigen den wert von sp, damit klar ist, dass es sich dabei um eine Adresse handelt.

Beispiel RISC-V

Alter Code war "gelogen", echter Code eher so:
(Simplified output von `https://godbolt.org/` mit -fomit-frame-pointer)

```asm
# int x = 5;
li      a5,5          # 5 in Register a5 laden
sw      a5,12(sp)     # Wert von Register a5 in Adr. von x schreiben
# int y = 3;
li      a5,3          # 5 in Register a5 laden
sw      a5,8(sp)      # Wert von Register a5 in Adr. von y schreiben
# int z = x + y;
lw      a4,12(sp)     # Wert von x in a4 laden
lw      a5,8(sp)      # Wert von y in a5 laden
add     a5,a4,a5      # a5 = a4 + a5
sw      a5,4(sp)      # Wert von Register a5 an Adr. von z schreiben
```

Asm

Mehr Folien die den Register und Memory-Content in einer Grafik zeigen.
Lässt sich als showcase zeigen: `https://venus.kvakil.me/`

Inhalt dieser Vorlesung

Einführung

Datenpfad: CPU

Datenpfad: Memory

Pointer

Bisher weiß Compiler Memory-Adressen.
Gutes Beispiel finden, wo wir variable Adressen brauchen.
Pointer detaillierter erklären