Grundlagen der Rechnerarchitektur

Felix Leitl

25. Juli 2023

Inhaltsverzeichnis

Zahlensysteme

Präfixe

Kilo	10^3	Kibi	2^{10}
Mega	10^6	Mebi	2^{20}
Giga	10^9	Gibi	2^{30}
Tera	10^12	Tebi	2^{40}
Peta	10^15	Pebi	2^{50}

Multiplikation und Division mit Zweierpotenzen

Bei Multiplikation einen shift nach links, bei Division einen shift nach rechts:

$$0xAB \cdot 2^2 = 10101011 << 2 = 1010101100 = 0x2AC$$
$$0xAB/2^2 = 10101011 >> 2 = 00101010 = 0x2A$$

Rechnerarchitektur

Endo- vs. Befehlsarchitektur

Exteren Sicht(Befehlsarchitektur): Was muss nach außen hin sichtbar sein, damit man den Computer programmieren kann?

Interne Sicht(Endoarchitektur): Wie werden die Funktionalitäten intern realisiert?

Von-Neumann, URA und ISA

7 Eigenschaften des URAs/von-Neumann Architektur:

1. Rechner besteht aus 4 Werken:
   (a) Rechenwerk
   (b) Speicherwerk
   (c) Ein-/Ausgabewerk
   (d) Leitwerk
2. Rechner ist programmgestuerert
3. Programme und Daten im selben Speicher
4. Hauptspeicher ist in Zellen gleicher Größe aufgeteilt, jede Zelle hat eine Adresse
5. Programm ist eine Sequenz an Befehlen
6. Abweichung von sequentieller Ausführung durch Sprünge möglich
7. Rechner verwendet Binärdarstellung

Befehlszyklus

von-Neumann-Befehlszyklus:

1. Befehl holen
2. Befehl dekodieren
3. Operanden holen
4. Befehl ausführen
5. Ergebnis zurückschreiben
6. Nächsten Befehl addresieren

Assemblertheorie

RiscV-Befehlssatz

add[i]	x1	x2	x3	
and[i]	x1	x2	x3	
or[i]	x1	x2	x3	
xor[i]	x1	x2	x3	
sll[i]	x1	x2	x3	shift left
srl[i]	x1	x2	x3	shift right
mv	x1	x2		move
neg	x1	x2		logical negation
not	x1	x2		bitwise negation
sub	x1	x2	x3	
mul	x1	x2	x3	
div	x1	x2	x3	
rem	x1	x2	x3	remainder
li	x1	Imm		
la	x1	lable		
lb	x1	Imm(x2)		load byte
lh	x1	Imm(x2)		
lw	x1	Imm(x2)		
sb	x1	Imm(x2)		store byte
sh	x1	Imm(x2)		
sw	x1	Imm(x2)		
call	lable			
ret				

Speicherbereiche

		Schreibbar	Ausführbar	Dynamisch wachsend
Textsegment	Programmcode	Nein	Ja	Nein
Datensegment	Globale Variablen	Ja	Nein	Nein
Stack	Lokale Variablen	Ja	Nein	Ja
Heap	langlebige Variablen	Ja	Nein	ja

Stack

In RiscV wächst der Stack von oben nach unten. RiscV bietet ein spezielles Stackpointer-Register(sp Register), welches immer auf die Adresse des zuletzt hinzugefügten Elements zeigt

Lesen vom Stack

Letztes Element des Stacks lesen: `lw t0, (sp)`
Vorletztes Element des Stacks lesen: `lw t0, 4(sp)` (hier ein Integer (word))

Schreiben auf den Stack

Ein Element auf den Stack legen:

```
li t0, 10
addi sp, sp, -4
sw, t0, (sp)
```

Mehrere Elemente auf den Stack legen:

```
addi sp, sp,  -12
sw, t0, 8(sp)
sw, t1, 4(sp)
sw, t2, (sp)
```

Speicher freigeben

```
addi sp, sp, 4
```

Mikroprogrammierung

Befehlssatzarchitektur

Endianess and Alignment

Speicherhirachie

Cache

Arbeitsspeicher

Pipelining

Instruktionsparallelismus

Threadparallelismus

Grafikkarten

Speicherverwaltung