

## 13.2 Übergang zur realen Maschine

Bernd Becker – Technische Informatik II

### Unterschiede zwischen abstrakter und realer Maschine

1. Bei realer Maschine nur *ein Speicher M* für Daten und Befehle.  
M ist *endlich*.  
Für  $i \in \{ 0, \dots, 2^{32}-1 \}$  ist  $M(i)$  Inhalt der  $i$ -ten Speicherzelle.  
Speicherzellen können Elemente aus  $\{ 0,1 \}^{32}$  aufnehmen.

BB - TI II 13.2/2

## Unterschiede zwischen abstrakter und realer Maschine (ff)

2. CPU-Register PC, ACC, IN1, IN2 können nur Elemente  $w \in \{0,1\}^{32}$  aufnehmen.

$w$  heißt *Wort*.

Verschiedene Interpretationsmöglichkeiten von

Worten:

- Binärzahlen (z.B. bei Adressen aus M)
- 2er-Komplement (z.B. Zahlen bei arithmetischen Operationen)
- Bitstrings (z.B. bei logischen Operationen, die zusätzlich betrachtet werden)

BB - TI II 13.2/3

## Unterschiede zwischen abstrakter und realer Maschine (ff)

3. Befehle sind ebenfalls Worte aus  $\{0,1\}^{32}$ .

→ Parameter  $i \in \mathbf{N}_0$  bzw.  $i \in \mathbf{Z}$  werden kodiert durch  $m$ -stellige Binärzahlen bzw.

2er-Komplementzahlen mit  $m < 32$ .

→ *Maschinensprache* bzw. *Instruktionssatz*

BB - TI II 13.2/4

## Schreibweisen:

$$b^j = \underbrace{(b, \dots, b)}_{j \text{ mal}} \quad \text{für } b \in \{0, 1\}$$

$\langle A \rangle := B$  (A Register oder Speicherzelle,  
 $B \in \{0, \dots, 2^{32} - 1\}$ )

bedeute  $A := \text{bin}_{32}(B)$  (Beispiel:  $\langle \text{PC} \rangle := \langle \text{PC} \rangle + 1$ )

$[A] := B$  (A Register oder Speicherzelle,  
 $B \in \{-2^{31}, \dots, 2^{31} - 1\}$ )

bedeute  $A := \text{twoc}_{31}(B)$

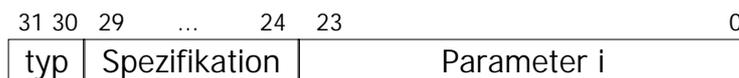
BB - TI II 13.2/5

## Instruktionsformate

Sei  $I = i_{31}, \dots, i_0 \in \{0, 1\}^{32}$ .

$I[y, x] := i_y, i_{y-1}, \dots, i_x$  für  $0 \leq x \leq y \leq 31$

Allgemeines Instruktionsformat:



BB - TI II 13.2/6

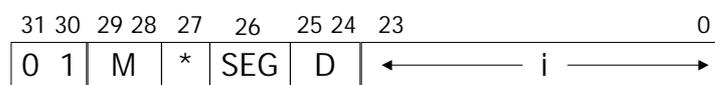
## Typ einer Instruktion:

I [31, 30]	Typ
0 0	Compute
0 1	Load
1 0	Store, Move
1 1	Jump

BB - TI II 13.2/7

## Load - Befehle

Prinzip:



Modus

vorläufig nicht relevant,  
siehe später

BB - TI II 13.2/8

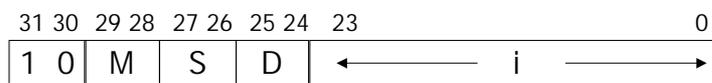
## Load – Befehle (ff)

Typ	Modus	SEG	Befehl	Wirkung	
0 1	0 0	0	LOAD i	ACC := M(<i>)	<PC> := <PC> + 1
0 1	0 1	0	LOADIN1 i	ACC := M(<IN1> + [i])	<PC> := <PC> + 1
0 1	1 0	0	LOADIN2 i	ACC := M(<IN2> + [i])	<PC> := <PC> + 1
0 1	1 1	0	LOADI i	ACC := 0 <sup>8</sup> i	<PC> := <PC> + 1

Durchführung von Rechnungen  $\langle x \rangle + [y]$  : siehe später

## Store, Move - Befehle

Prinzip:



Modus

Source

Drain

## Store, Move - Befehle (ff)

Typ	Modus	Befehl	Wirkung	
1 0	0 0	STORE i	M(<i>) := ACC	<PC> := <PC> + 1
1 0	0 1	STOREIN1 i	M(<IN1> + [i]) := ACC	<PC> := <PC> + 1
1 0	1 0	STOREIN2 i	M(<IN2> + [i]) := ACC	<PC> := <PC> + 1
1 0	1 1	MOVE S D	D := S	<PC> := <PC> + 1

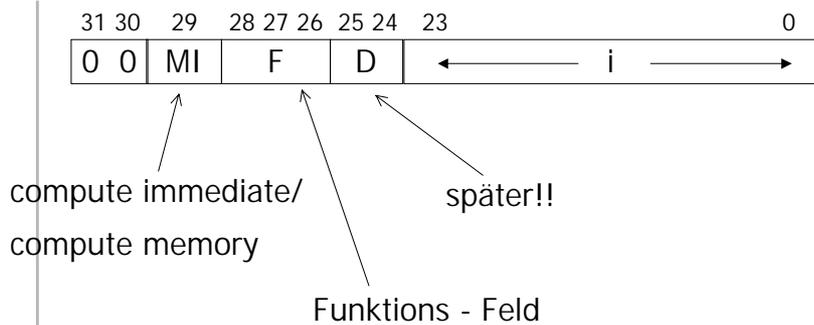
außer bei D = 0 0 (PC)

## Kodierung S, D

S, D	Register
0 0	PC
0 1	IN1
1 0	IN2
1 1	ACC

## Compute - Befehle

Prinzip:



BB - TI II 13.2/13

## Compute - Befehle (ff)

Typ	MI	F	Befehl	Wirkung	
00	0	010	SUBI i	$[ACC] := [ACC] - [i]$	$\langle PC \rangle := \langle PC \rangle + 1$
		011	ADDI i	$[ACC] := [ACC] + [i]$	$\langle PC \rangle := \langle PC \rangle + 1$
		100	OPLUSI i	$ACC := ACC \oplus 0^8 i$	$\langle PC \rangle := \langle PC \rangle + 1$
		101	ORI i	$ACC := ACC \vee 0^8 i$	$\langle PC \rangle := \langle PC \rangle + 1$
		110	ANDI i	$ACC := ACC \wedge 0^8 i$	$\langle PC \rangle := \langle PC \rangle + 1$
00	1	010	SUB i	$[ACC] := [ACC] - [M(\langle i \rangle)]$	$\langle PC \rangle := \langle PC \rangle + 1$
		011	ADD i	$[ACC] := [ACC] + [M(\langle i \rangle)]$	$\langle PC \rangle := \langle PC \rangle + 1$
		100	OPLUS i	$ACC := ACC \oplus M(\langle i \rangle)$	$\langle PC \rangle := \langle PC \rangle + 1$
		101	OR i	$ACC := ACC \vee M(\langle i \rangle)$	$\langle PC \rangle := \langle PC \rangle + 1$
		110	AND i	$ACC := ACC \wedge M(\langle i \rangle)$	$\langle PC \rangle := \langle PC \rangle + 1$

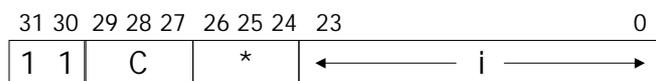
## Bemerkung zu OPLUS

$$\text{ACC} := \text{ACC} \oplus 0^8 i_{23} \dots i_0 \equiv$$
$$(\text{ACC}_{31} \oplus 0, \dots, \text{ACC}_{24} \oplus 0, \text{ACC}_{23} \oplus i_{23}, \dots, \text{ACC}_0 \oplus i_0)$$

BB - TI II 13.2/15

## Jump - Befehle

Prinzip:



Condition

BB - TI II 13.2/16

## Kodierung der Bedingung c:

C	Bedingung c
0 0 0	nie
0 0 1	>
0 1 0	=
0 1 1	≥
1 0 0	<
1 0 1	≠
1 1 0	≤
1 1 1	immer

BB - TI II 13.2/17

## Kodierung nach Schema:

nur I [29] = 1 ⇔ < wird abgefragt

nur I [28] = 1 ⇔ = wird abgefragt

nur I [27] = 1 ⇔ > wird abgefragt

Andere Abfragen durch Kombinationen, z.B.

C = 101 : < oder > , also ≠

BB - TI II 13.2/18

## Jump - Befehle (ff)

Typ	Befehl	Wirkung
1 1	JUMP <sub>C</sub> i	$\langle PC \rangle \leftarrow i$ , falls $\langle ACC \rangle \neq 0$ $\langle PC \rangle \leftarrow \langle PC \rangle + 1$ , sonst

*Unbedingte Sprünge* werden durch C = 111 ausgedrückt.

Bei C = 000 : *Keine* Wirkung des Befehls außer

Inkrementieren des Befehlszählers

→ NOP – Befehl (No Operation)

BB - TI II 13.2/19

## Andere Befehle

... sind durchaus sinnvoll und  
bei anderen Architekturen evtl. schon als  
Grundbefehl vorhanden.

Hier:

Zusammensetzen aus Grundbefehlen

mit Hilfe von *Unterprogrammen*.

BB - TI II 13.2/20

## Beispiel: Shift - Operationen

Linksshift: lsh:  $\mathbf{B}^n \rightarrow \mathbf{B}^n$ ,

$$(a_{n-1}, \dots, a_0) \rightarrow (a_{n-2}, \dots, a_0, 0)$$

zyklischer Linksshift: cls:  $\mathbf{B}^n \rightarrow \mathbf{B}^n$ ,

$$(a_{n-1}, \dots, a_0) \rightarrow (a_{n-2}, \dots, a_0, a_{n-1})$$

Rechtsshift: rsh:  $\mathbf{B}^n \rightarrow \mathbf{B}^n$ ,

$$(a_{n-1}, \dots, a_0) \rightarrow (0, a_{n-1}, \dots, a_1)$$

zyklischer Rechtsshift: crs:  $\mathbf{B}^n \rightarrow \mathbf{B}^n$ ,

$$(a_{n-1}, \dots, a_0) \rightarrow (a_0, a_{n-1}, \dots, a_1)$$

Realisierung mit vorhandenen Maschinenbefehlen: siehe Übung !

BB - TI II 13.2/21

## Addition und Sign Extension

Probleme bei Additionen:

1. Addition verschieden langer Zahlen

$$( \text{z.B. } [\text{ACC}] + [i] )$$

2. Addition von Binärdarstellungen und Zweierkomplementzahlen

$$( \text{z.B. } M(\underbrace{\langle \text{IN1} \rangle}_!) + [i] ) := \text{ACC}$$

BB - TI II 13.2/22

## Zu 1.

Lösung durch Sign Extension:

Sei  $y \in \{0,1\}^{24}$ .

Dann sei  $\text{sext}(y) := y_{23}^8 y$

$\text{sext}(y)$  heißt sign extension von  $y$

Es gilt:  $[y] = [\text{sext}(y)] \quad (\rightarrow \text{Übung})$

Damit wird  $[\text{ACC}] + [i]$  zurückgeführt auf  
 $[\text{ACC}] + [\text{sext}(i)]$

BB - TI II 13.2/23

## zu 2.: Lemma:

Sei  $x \in \mathbf{B}^{32}, y \in \mathbf{B}^{32}, 0 \leq \langle x \rangle + [y] < 2^{32}$

und es sei

$\langle x \rangle + \langle \text{sext}(y) \rangle = \langle c, s \rangle$  mit  $c \in \mathbf{B}, s \in \mathbf{B}^{32}$ .

Dann gilt:  $\langle x \rangle + [y] = \langle s \rangle$

D.h. es genügt,  $x$  und  $\text{sext}(y)$  als Binärzahl  
zu interpretieren und den Übertrag zu ignorieren  
(wenn keine Überlaufbehandlung nötig).

Beweis: Übung

BB - TI II 13.2/24