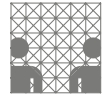


64-041 Übung Rechnerstrukturen



Aufgabenblatt 10 Ausgabe: 13.01., Abgabe: 20.01. 12:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 10.1 (Punkte 3+3+3+3+3)

Adressierung: Auf einer 1-Adress Maschine (Akkumulatormaschine) werden Ladebefehle mit unterschiedlichen Adressierungsmodi ausgeführt. Der Speicher enthält folgende Werte:

Adresse	Inhalt
20	40
30	60
40	50
50	30
60	80
70	10
80	20

Welcher Wert steht jeweils nach Ausführung der folgenden Befehle im Akkumulator?

- (a) LOAD IMMEDIATE 50
- (b) LOAD DIRECT 50
- (c) LOAD INDIRECT 50
- (d) LOAD DIRECT 60
- (e) LOAD INDIRECT 60

Aufgabe 10.2 (Punkte 15)

Befehlscodierung: Entwerfen Sie eine möglichst einfache und einheitliche Befehlscodierung, um alle der folgenden Befehle in 32-bit Befehlsworten unterzubringen:

- 15 Befehle mit einer 4-bit Registernummer und einer 24-bit Adresse
- 120 Befehle mit zwei 4-bit Registernummern und einem Adressoffset
Wie viele Bits stehen maximal für diesen Adressoffset zur Verfügung?
- 30 Befehle ohne Adressen oder Registerangaben

Skizzieren Sie für die drei Befehlsformate die Aufteilung der 32-bit Befehlsworte in die einzelnen Gruppen und begründen Sie Ihren Entwurf.

Aufgabe 10.3 (Punkte 4+4+4+4+4)

Darstellung von Immediate-Operanden: Um trotz eingeschränkter Wortlängen möglichst viele, häufig benötigte Werte als *immediate* Operand darzustellen, benutzen die Befehlsätze aktueller Prozessoren einige Tricks. Ein gutes Beispiel zeigt die für eingebettete Systeme und Mobilgeräte sehr beliebte 32-bit ARM-Architektur.

Unter anderem ist folgendes Befehlsformat mit Immediate-Operanden arithmetischen Befehle definiert:

cond	opcode	r_src	r_dest	rot	imm8
31 28 27	20 19	16 15	12 11	8 7	0

Dabei wird der in der Hardware vorhandene *Barrel-Shifter* benutzt, um 32-bit Immediate-Werte zu erzeugen:

$\langle imm8 \rangle$ 8-bit 0...255 beliebiger Ausgangswert

$\langle rot \rangle$ 4-bit 0...15 Distanz für eine *rotate-right* Operation ist $\langle rot \rangle \times 2$, Schrittweite 2

$imm32 = \langle imm8 \rangle \text{ rotate-right } (\langle rot \rangle \times 2)$

Überlegen Sie sich die jeweilige 12-bit Codierung der folgenden Immediate-Werte, oder begründen Sie, warum ein Wert nicht dargestellt werden kann.

- (a) 207
- (b) 398
- (c) 351
- (d) 2 684 354 560
- (e) 1 212 416

Aufgabe 10.4 (Punkte 4·10+10)

Befehlsformate: Vergleichen Sie 0-, 1-, 2- und 3-Adress Maschinen, indem Sie für jede Architektur ein Programm zur Berechnung des folgenden Ausdrucks schreiben:

$$W = (A * B - C) / (D + E * F)$$

Die verfügbaren Befehle der entsprechenden Maschinen sind unten angegeben. Dabei bedeuten M und K jeweils eine 16-bit Speicheradresse, und MEM[M] den Inhalt des Speichers an der Adresse M, während X,Y,Z für eine 4-bit Registernummer codieren.

0-Adress Maschine mit einen unbegrenzten Stack (TOS "top of stack")

Mnemonic	Bedeutung
PUSH M	push; TOS = MEM[M]
POP M	MEM[M] = TOS; pop
ADD	tmp = TOS; pop; TOS = tmp + TOS
SUB	tmp = TOS; pop; TOS = tmp - TOS
MUL	tmp = TOS; pop; TOS = tmp * TOS
DIV	tmp = TOS; pop; TOS = tmp / TOS

1-Adress Maschine: Akkumulatormaschine mit genau einem Register

Mnemonic	Bedeutung
LOAD M	Akku = MEM[M]
STORE M	MEM[M] = Akku
ADD M	Akku = Akku + MEM[M]
SUB M	Akku = Akku - MEM[M]
MUL M	Akku = Akku * MEM[M]
DIV M	Akku = Akku / MEM[M]

2-Adress Maschine: benutzt nur Speicheroperanden

Mnemonic	Bedeutung
MOV M, K	MEM[M] = MEM[K]
ADD M, K	MEM[M] = MEM[M] + MEM[K]
SUB M, K	MEM[M] = MEM[M] - MEM[K]
MUL M, K	MEM[M] = MEM[M] * MEM[K]
DIV M, K	MEM[M] = MEM[M] / MEM[K]

3-Adress Register-Maschine: *load-store* RISC-Architektur, 16 Universalregister

Mnemonic	Bedeutung
LOAD X, M	X = MEM[M]
STORE M, X	MEM[M] = X
MOV X, Y	X = Y
ADD X, Y, Z	X = Y + Z
SUB X, Y, Z	X = Y - Z
MUL X, Y, Z	X = Y * Z
DIV X, Y, Z	X = Y / Z

- (a) Schreiben Sie (möglichst kurze) Programme für die Berechnung von $W = (A * B - C) / (D + E * F)$ auf den vier Maschinen. Dabei bedeuten A..F und W die Speicheradressen für die Operanden bzw. den Resultatwert. Verwenden Sie falls nötig die Speicheradressen von H..V für Zwischenergebnisse.
- (b) Wenn die Befehlskodierung jeweils 8-bit für den Opcode verwendet (und natürlich 16-bit für eine Speicheradresse bzw. 4-bit für eine Registernummer), wie viele Bits werden dann für jedes der obigen vier Programme benötigt?

Welche Maschine hat also die kompakteste Codierung (gemessen an der Programmgröße in Bits) für dieses Programm?