64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 11 Ausgabe: 08.01., Abgabe: 15.01. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 11.1 (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	50
30	40
40	60
50	70

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

(a) `LOAD IMMEDIATE 20`

(b) `LOAD DIRECT 20`

(c) `LOAD INDIRECT 20`

(d) `LOAD DIRECT 30`

(e) `LOAD INDIRECT 30`

Aufgabe 11.2 (Punkte 15)

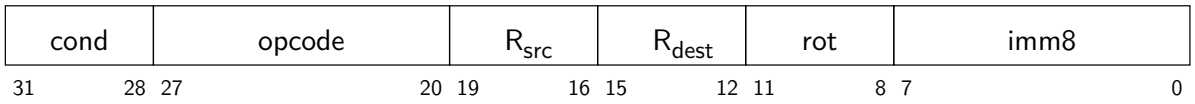
Befehlscodierung: Entwerfen Sie eine möglichst einfache und einheitliche Befehlscodierung, um alle der folgenden Befehle in 32-bit Befehlsworten unterzubringen:

- 7 Befehle mit einer 5-bit Registernummer und einer 24-bit Adresse
- 500 Befehle mit zwei 4-bit Registernummern und einem Adressoffset
  Wie viele Bits stehen maximal für diesen Adressoffset zur Verfügung?
- 50 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 11.3 (Punkte 4+4+4+4+4)

Darstellung von Immediate-Operanden: Um trotz eingeschränkter Wortlängen bei RISC-Befehlssätzen möglichst viele, häufig benötigte Werte als *Immediate* darzustellen, benutzen die Befehlssätze aktueller Prozessoren einige Tricks. Ein gutes Beispiel zeigt die für eingebettete Systeme und Mobilgeräte sehr beliebte 32-bit ARM-Architektur. Dort ist unter anderem für arithmetische Befehle folgendes Format mit Immediate-Operanden definiert:



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: $\langle rot \rangle \times 2$ (Schrittweite 2)
imm32 = $\langle imm8 \rangle$ *rotate-right* ($\langle rot \rangle \times 2$)

Überlegen Sie sich die jeweilige 12-bit Codierung ($\langle rot \rangle$ $\langle imm8 \rangle$) der folgenden Immediate-Werte oder begründen Sie, warum ein Wert nicht dargestellt werden kann.

(a) 167

(b) 267

(c) 573 440

(d) 1 233 125 376

(e) 1 342 177 280

Aufgabe 11.4 (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. Dabei gilt (natürlich) Punkt- vor Strichrechnung:

$$R = (A^2 + B)/(C - D * E)$$

Die verfügbaren Befehle der entsprechenden Maschinen sind unten angegeben. `M` und `N` stehen dabei für 16-bit Speicheradressen, während `X`, `Y` und `Z` eine 4-bit Registernummer codieren. `MEM[M]` sei der Inhalt des Speichers an der Adresse `M`.

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

Mnemonik	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

Mnemonik	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

Mnemonik	Bedeutung
`MOV M,N`	MEM[M] = MEM[N]
`ADD M,N`	MEM[M] = MEM[M] + MEM[N]
`SUB M,N`	MEM[M] = MEM[M] − MEM[N]
`MUL M,N`	MEM[M] = MEM[M] ∗ MEM[N]
`DIV M,N`	MEM[M] = MEM[M] / MEM[N]

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

Mnemonik	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 für alle vier Maschinen (möglichst kurze) Programme für die Berechnung von $R = (A^2 + B)/(C - D * E)$. Dabei stehen $A \ldots F$ und R für Speicheradressen der Operanden bzw. des Resultats. Zwischenergebnisse können (bei Bedarf) auf ungenutzten Speicheradressen ($G \ldots Q$) abgelegt werden.

(b) Wenn die Befehlscodierung 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?