64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 11 Ausgabe: 05.01., Abgabe: 12.01. 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	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 11.2 (Punkte 15)

Befehlscodierung: Entwerfen Sie eine möglichst einfache und einheitliche Befehlscodierung für eine 32-bit RISC Architektur. Dabei sind folgende Befehle in den 32-bit Befehlsworten unterzubringen:

- 7 Befehle mit einer 5-bit Registeradresse und einem 24-bit Wert
- 30 Befehle mit drei 5-bit Registeradressen
- 70 Befehle mit zwei 5-bit Registeradressen und einem Immediate-Operanden
  Wie viele Bits stehen maximal für den Operanden zur Verfügung?
- 30 Befehle ohne Adressen oder Registerangaben

Skizzieren Sie für die vier Befehlsformate die Aufteilung der 32-bit Worte in Bit-/Funktions-Gruppen und begründen Sie Ihren Entwurf. Nennen Sie Beispiele für jedes Befehlsformat.

Aufgabe 11.3 (Punkte 4·10+5)

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:

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

Die verfügbaren Befehle der entsprechenden Maschinen sind unten angegeben. `M` und `N` stehen dabei für 24-bit Speicheradressen, während `X`, `Y` und `Z` eine 5-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, 32 Universalregister

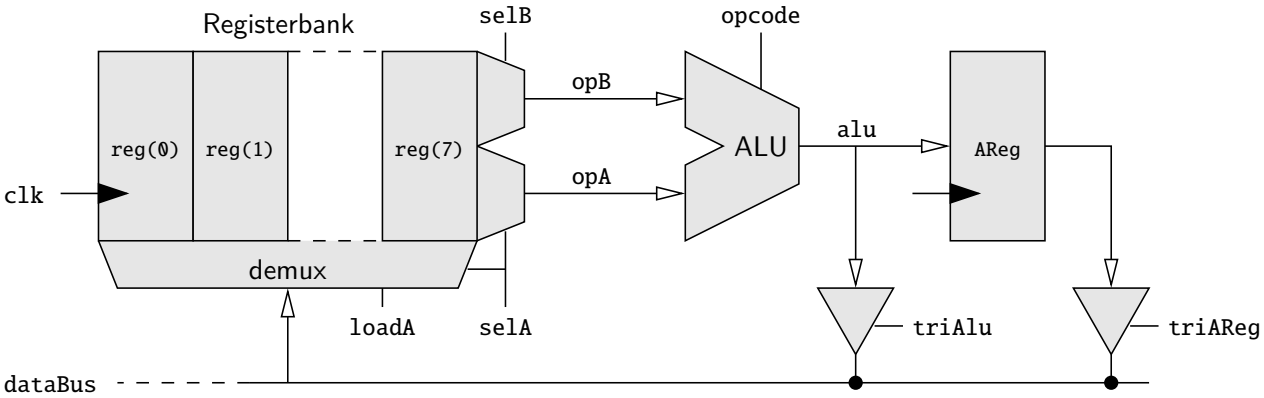
Mnemonik	Bedeutung
`LOAD X,M`	X = MEM[M]
`STORE M,X`	MEM[M] = X
`MOV X,Y`	X = Y
`ADD X,Y,Z`	X = Y + Z
`SUB X,Y,Z`	X = Y − Z
`MUL X,Y,Z`	X = Y ∗ Z
`DIV X,Y,Z`	X = Y / Z

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

(b) Wenn die Befehlscodierung jeweils 8-bit für den Opcode verwendet, 24-bit für eine Speicheradresse und 5-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?

Aufgabe 11.4 (Punkte 8+8+9)

Hardwarearchitektur: Bei dieser Aufgabe sollen Sie sich überlegen, wie auf einer vorgegebenen Hardwarearchitektur die Befehle eines 16-bit Mikrocontrollers abgearbeitet werden. Dem Paradigma der von-Neumann Architektur folgend, wird jeder Befehl in aufeinanderfolgenden Phasen ausgeführt: Befehl holen, decodieren, rechnen etc. Zur Ausführung der ALU-Operationen, sind im Rechenwerk die hier skizzierten Hardwareeinheiten vorhanden.



Zum Verständnis der Aufgabenteile wird die Funktionsweise der einzelnen Einheiten kurz erläutert:

Registerbank bestehend aus 8 16-bit Registern. Ist `loadA` aktiv, dann wird das mit `selA` selektierte Register mit dem Wert des Datenbusses geladen:

`dataBus` → `reg(selA)`

ALU mit einigen arithmetischen und logischen Befehlen, die als `opcode` spezifiziert werden. Die beiden Operanden `opA` und `opB` werden über Multiplexer (`selA`, bzw. `selB`) mit Werten aus der Registerbank versorgt.

Ausgangsregister (16-bit), der ALU nachgeschaltet.

Bidirektionales Bussystem `dataBus`, das im Datenpfad zwei Quellen besitzt: den direkten Ausgang der ALU oder das dahinter liegende Register. Beide sind über Tristate-Treiber, die über entsprechende Kontrollsignale (`triAlu` und `triAReg`) selektiert werden, an den Bus angeschlossen. Außerhalb des Mikrocontrollers würde dann auch noch der Speicher mit dem Bus verbunden sein.

(a) Wie würde auf der Hardware ein 2-Adress Befehl (beides Registeradressen) abgearbeitet werden? Beispielsweise die Addition: `ADD2 1,5`

Funktion: `reg(1) + reg(5)` → `reg(1)`

Schreiben Sie für die einzelnen Signale die jeweiligen Belegungen und die zugehörige Aktion auf. Wenn erforderlich über mehrere Takte:

Takt	`loadA`	`selA`	`selB`	`opcode`	`triAlu`	`triAReg`	Wirkung
1							
2							
...							

(b) Wie würde auf der Hardware ein 3-Adress Befehl (drei Registeradressen) abgearbeitet werden? Beispielsweise die Addition: `ADD 0,1,5`

Funktion: `reg(1) + reg(5)` → `reg(0)`

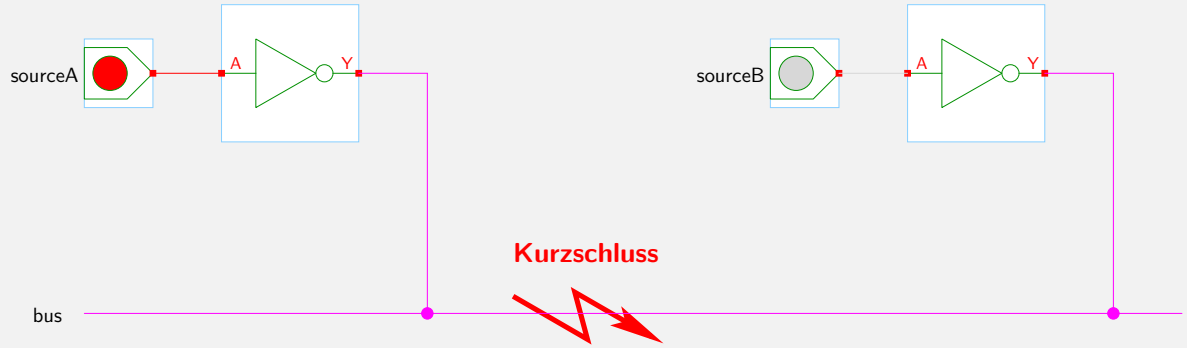
Schreiben Sie auch hier die Signalbelegungen und die zugehörigen Aktionen auf. Welche Verbesserungen in der Architektur wären demnach möglich?

(c) Angenommen man hätte neben dem Datenpfad weitere Hardware: ein Steuerwerk mit Kontrollautomaten, Befehlszähler, Instruktionsregister sowie die restliche Infrastruktur eines Mikrorechners. Trotzdem könnten noch keine Programme abgearbeitet werden, da in dieser Architektur noch etwas fehlt: Was muss zusätzlich vorhanden sein?

Tipp: Ein zentrales Konstrukt *jeder Art von Programmierung* ist noch nicht realisierbar. Welche „Arten" von Befehlen gibt es in einfachen (RISC-) Befehlssätzen? Können die Befehle alle auf diesem Datenpfad ausgeführt werden?

Tristate-Treiber

Bei Bussystemen, wo die Datenübertragung nicht nur in eine Richtung von Quelle (Sender) zu Senke (Empfänger) stattfindet, sondern mehrere Sender möglich sind, benötigt man auf Hardwareebene Tristate-Elemente. Als einfaches Beispiel kann man den Datenbus einer von-Neumann Architektur nehmen, wo ein Prozessor (Sender) Werte in den Speicher (Empfänger) schreibt, wo der Transfer über den Bus aber auch in die andere Richtung stattfindet, wenn Befehle oder Daten aus dem Speicher gelesen werden. Auf elektrischer Ebene hat man mehrere Gatterausgänge als Treiber auf einer Leitung „zusammengeschaltet“. Wie das folgende HADES-Beispiel mit zwei Quellen zeigt, kommt es zum Kurzschluss wenn Ausgang A eine 0 und Ausgang B eine 1 auf den Bus legt.



Tristate-Treiber ermöglichen als „dritten Zustand“, daher der Name, eine Datenquelle elektrisch komplett vom Bussystem zu trennen. Solange höchstens ein Treiber aktiv ist, was Aufgabe des Automaten ist, der die Steuersignale (`triA`, `triB`) erzeugt, funktioniert das Bussystem ohne Probleme.