64-041 Übung Rechnerstrukturen

Aufgabenblatt 11 Ausgabe: 11.01., Abgabe: 18.01. 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	60
30	40
40	20
50	40
60	20
70	80
80	60

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

(a) `LOAD IMMEDIATE 40`

(b) `LOAD DIRECT 70`

(c) `LOAD INDIRECT 60`

(d) `LOAD DIRECT 80`

(e) `LOAD INDIRECT 30`

Aufgabe 11.2 (Punkte 4·8+8)

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. `M` und `K` 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,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

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 $W = (A * B - C)/(D + E * F)$. Dabei stehen $A..F$ und W für Speicheradressen der Operanden bzw. des Results. Falls nötig, können Sie ungenutzte Speicheradressen ($H..V$) für Zwischenergebnisse verwenden.

(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?

Aufgabe 11.3 (Punkte 7*3 [davon 6 Bonus])

Flags: Viele Prozessoren (z.B. x86-Architektur) haben ein Carry- und ein Overflow-Flag. Kreuzen Sie an, welches Flag bei Ausführung der folgenden Operationen gesetzt werden würde.

Operation	Inhalt %eax	Inhalt %ebx	CF	OF
addl %eax, %ebx	0x00000001	0x00000002		
addl %eax, %ebx	0x00000001	0xFFFFFFFF		
addl %eax, %ebx	0x00000002	0x7FFFFFFF		
addl %eax, %ebx	0x80000000	0xFFFFFFFE		
subl %eax, %ebx	0x00000002	0x00000001		
subl %eax, %ebx	0x00000001	0x00000002		
subl %eax, %ebx	0x00000001	0x80000000		

Zur Erinnerung: das Carry-Flag wird bei einer arithmetischen Operation gesetzt, wenn sich in der höchstwertigsten Stelle ein Übertrag ergibt. Das Overflow-Flag wird gesetzt, wenn das Ergebnis der Operation das „falsche“ Vorzeichen hat, beispielsweise wenn die Addition zweier positiver Zahlen ein negatives Ergebnis liefert.

Aufgabe 11.4 (Punkte 5+5+5+5+5+5)

x86-Adressierung: Angenommen, die folgenden Werte sind in den angegebenen Registern bzw. Speicheradressen gespeichert:

Register	Wert
%eax	0x00000100
%ecx	0x0000000C
%edx	0x00000004

Adresse	Wert
0x100	0x0000BEEF
0x104	0x000000CB
0x108	0x00012300
0x10C	0x00098700

Überlegen Sie sich, welche Speicheradressen bzw. Register als Ziel der folgenden Befehle ausgewählt werden und welche Resultatwerte sich aus den Befehlen ergeben:

(a) `addl %edx, (%eax)`

(b) `subl %ecx, 4(%eax)`

(c) `imull $32, (%eax,%edx,2)`

(d) `incl 12(%eax)`

(e) `decl %ecx`

(f) `subl %edx, %eax`

Sie können die Befehle natürlich gerne auch im Assembler und Debugger direkt ausprobieren. Mit einigen Befehlen lassen sich die oben angegebenen Werte in den Speicher schreiben, und die Resultate lassen sich dann direkt ablesen. Geben Sie in diesem Fall Ihr Assemblerprogramm bitte mit ab.

Zur Erinnerung: für den gnu-Assembler gilt

- der Zieloperand steht rechts
- Registerzugriffe werden direkt ausgedrückt
- eine runde Klammer um ein Register bedeutet einen Speicherzugriff, ggf. mit Immediate-Offset und Index: $\langle imm \rangle$`(`$\langle Rb \rangle$`,`$\langle Ri \rangle$`,`$\langle s \rangle$`)` $\rightarrow$ `MEM[`$\langle Rb \rangle$`+`$\langle s \rangle$`*`$\langle Ri \rangle$`+`$\langle imm \rangle$`]`

$\Rightarrow$ zum Beispiel bewirkt der Befehl: `addl %edx, 8(%eax)`
die Operation: `MEM[0x00000108] = 0x00012304`