## Aufgabe 9

a) do ρ(ρ(γ)+100):=α;

```
lw $1, 100($1) # γ':=ρ(γ)+100
               # ich gehe davon aus, dass die Adressierung bereits
               # umgerechnet wurde, sonst müsste hier 25($1) stehen
sw $0, 0($1)   # ρ(γ'):=α
```

b) do α:=ρ(ρ(γ)+50);

```
lw $1, 50($1)  # γ':=ρ(γ)+50, zur Adressierung siehe Aufgabe a)
lw $0, 0($1)   # α:=ρ(γ')
```

c) if α=γ goto j;

```
beq $0, $1, j
```

d) do α:=α+γ;

```
add $0, $0, $1 # ich gehe davon aus, dass der Rechner in der Lage ist,
               # in einem Takt α auszulesen, neu zu berechnen und dann
               # zu beschreiben - falls nicht, müsste man ein drittes,
               # temporäres Register benutzen
```

## Aufgabe 10

a) In Pseudocode formuliert sich die Gleichung um als (Umrechnung 8-Bit-Adressen in 32-Bit-Adressen bereits beachtet):
ρ(168+$9) = ρ(512+$9) - ($18 - ρ(40+$9))
Der dazugehörige Assemblercode sieht dann folgendermaßen aus:

```
lw  $7, 40($9)       # lade A[10] in Register $7
sub $8, $18, $7      # in Register $8 steht nun h-A[10]
lw  $7, 512($9)      # lade A[128] in Register $7
sub $8, $7, $8       # $8 enthält das Resultat obiger Gleichung
sw  $8, 168($9)      # $8 in A[42] abspeichern
```

Etwas kritisch ist für mich der Befehl in Zeile 4, da er Register $8 als Source und als Destination benutzt. Man könnte die Klammer in obiger Gleichung auflösen und dann
ρ(168+$9) = ρ(512+$9) - $18 + ρ(40+$9)
berechnen, wodurch dieses Register-Problem entfällt, da man sehr schön von links nach rechts die Gleichung ausrechnen lassen kann.

b) Im folgenden liste ich die einzelnen Belegungen der Bestandteile der Befehle in Dezimalschreibweise auf, in der zweiten Tabelle steht dann der komplette Befehl als Binärcode:

| Befehl | op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|---|
| `lw $7, 40($9)` | 35 | 9 | 7 | 40 | | |
| `sub $8, $18, $7` | 0 | 18 | 7 | 8 | 0 | 34 |
| `lw $7, 128($9)` | 35 | 9 | 7 | 512 | | |
| `sub $8, $7, $8` | 0 | 7 | 8 | 8 | 0 | 34 |
| `sw $8, 168($9)` | 43 | 9 | 8 | 168 | | |

| Befehl | Binärcode |
|---|---|
| `lw $7, 40($9)` | `100011 01001 00111 0000000000101000` |
| `sub $8, $18, $7` | `000000 10010 00111 01000 00000 100010` |
| `lw $7, 128($9)` | `100011 01001 00111 0000001000000000` |
| `sub $8, $7, $8` | `000000 00111 01000 01000 00000 100010` |
| `sw $8, 168($9)` | `101011 01001 01000 0000000010101000` |

| | |
|---|---|
| op | Operation |
| rs | 1. Registersource |
| rt | 2. Registersource |
| rd | Registerdestination |
| shamt | shift |
| funct | Funktion bzgl. op |

**Aufgabe 11**

a) Computerintern werden alle Zahlen im dualen Zahlensystem dargestellt, d.h. in der Form $2^n a_n + 2^{n-1}a_{n-1} + ... + 2^1 a_1 + 2^0 a_0$. Die Multiplikation mit 2 ergibt: $2^{n+1}a_n + 2^n a_{n-1} + ... + 2^2 a_1 + 2^1 a_0$, d.h. die Koeffizienten $a_0$ bis $a_n$ sind jeweils um eine Zweierpotenz verschoben worden, was auf Bitebene einer Linksverschiebung um 1 Bit entspricht. Ähnlich funktioniert die Division durch 2, hier wird um 1 Bit nach rechts geschoben.
Zusammenfassend lässt sich sagen, dass die Multiplikation bzw. Division mit/durch 2 durch die Zurückführung auf einfachste Bitverschiebungen ausgedrückt werden kann, was eine sehr einfache (und damit kostengünstige) Implementation in Hardware zulässt.

b) Die allgemeine Formel für die Ermittlung der MIPS-Zahl (million instructions per second) eines Computers sieht wie folgt aus: $MIPS = \frac{Befehlsanzahl}{Ausführungszeit * 10^6}$. Da in der Aufgabenstellung gefordert wird, dass beide Rechner für das gleiche Programm auch ungefähr die gleiche Zeit benötigen, hängen Unterschiede in der MIPS-Zahl im wesentlichen nur von der Befehlsanzahl ab.
Die CISC-CPU stellt einen umfangreichen Befehlssatz bereit, der in der Lage ist, Dinge mit einem Befehl zu erledigen, wofür die RISC-CPU unter Umständen mehrere Befehle benötigt. Somit kann man davon ausgehen, dass das Programm auf einer CISC-CPU aus weniger Befehlen besteht, somit auch die MIPS-Zahl niedriger als die der RISC ist.

**Aufgabe 12**

a) Die Anzahl der benötigten Bits ist:
$\log_2(10^{13}+1) \approx 43{,}185$ bits
Diese Zahl muss auf die nächstgrößere natürliche Zahl aufgerundet werden, d.h. es sind min. 44 Bits notwendig. Da der Computer mit 8 Bit breiten Bytes arbeitet, müssen $6*8 = 48$ Bits, also 6 Bytes breite Register verwendet werden.

b) Durch geschickte Verteilung der einzelnen Bits lässt sich auch eine Zahl auf mehrere Register verteilen. Dabei muss dann aber die unterschiedliche Wertigkeit der einzelnen Stellen beachtet werden. Die Benutzung mehrerer Register für eine einzelne Zahl ist z.B. üblich bei der Multiplikation zweier Register, da das Produkt eventuell doppelt soviele Bits wie die Faktoren benötigt.

c) 1 Byte entspricht $2^3$ Bits
1 MByte entsprechen $2^{20}$ Byte
128 MByte entsprechen $2^7$ Mbyte
Dieser Computer hat demzufolge einen Hauptspeicher in der Größe von $2^3 * 2^{20} * 2^7 = 2^{30}$ Bits. Da jedes Bit wiederum 2 Zustände annehmen kann (0 bzw. 1), sind $2^{2^{30}}$ Zustände möglich. Diese Zahl entspricht etwa $10^{3*10^8}$. Angesichts der Vermutung, dass unser bekanntes Universum nur aus ca. $10^{88}$ Elektronen besteht und die größte Zahl mit Namen $10^{140}$ ist (sie heißt im buddhistischen *Asankhyeya*), kann man postulieren, dass es unmöglich ist, alle Zustände dieses (Mittelklasse-) Computers genauestens durchzuarbeiten, um z.B. alle Bugs in der darauf laufenden Software zu finden.
[alle Zahlenbeispiele dem Guinessbuch entnommen]