Luentomonistetta ”Tietokoneen toiminta, Auvo Häkkinen, 1998” mukaillen.

# Yksinkertaisia pikkuohjelmia

## *Yksinkertaisen aritmeettisen lausekkeen laskeminen*

Lasketaan lauseke (mem[100]+mem[101])/mem[102] ja talletetaan tulos mem[100]:aan.

| | |
|---|---|
| **LOAD R1, 100** | Ladataan muistipaikan 100 (mem[100]) sisältö rekisteriin R1. |
| **ADD R1, 101** | Lisätään rekisterin R1 arvoon muistipaikan 101 sisältö ja talletetaan tulos rekisteriin R1, eli R1 = R1 + mem[101] |
| **DIV R1, 102** | Jaetaan rekisterin R1 arvo muistipaikan 102 sisällöllä ja talletetaan tulos rekisteriin R1, eli R1 = R1/ mem[102] |
| **STORE R1, 100** | Talletetaan rekisterissä R1 oleva arvo muistipaikkaan 100 (mem[100]). |

## *Peräkkäisten muistipaikkojen nollaus*

Nollataan peräkkäiset muistipaikat. Ensimmäisen muistipaikan osoite on R2:ssa, viimeisen R3:ssa.

| | | |
|---|---|---|
| | **LOAD R1, =0** | ”Nollaaja”. STORE-käskylle ei voida antaa lukua, sen täytyy olla rekisteri. |
| **Loop** | **STORE R1, 0(R2)** | Talletetaan rekisterin R1 arvo, eli nolla, muistiosoitteeseen, joka löytyy R2:sta. |
| | **ADD R2, =1** | Lisätään R2:een yksi, jolloin saadaan seuraavan muistipaikan osoite. |
| | **COMP R2, R3** | Ollaanko ohitettu viimeisen muistipaikan osoite? Sehän oli talletettu R3:een. Huomioi tässä järjestys, ensin nollataan, sitten siirrytään yksi muistipaikka eteenpäin, joten pitää verrata siihen onko R2 suurempi kuin R3, yhtäsuuruus ei kelpaa. Muuten viimeinen paikka jää nollaamatta. |
| | **JNGRE Loop** | Jos ei, hypätään takaisin silmukan alkuun. Jos ollaan ohitettu viimeinen nollattava muistipaikka, jatketaan seuraavaan käskyyn (jota ei tässä esimerkissä näy). |

Luentomonistetta ”Tietokoneen toiminta, Auvo Häkkinen, 1998” mukaillen.

## *Summan laskeminen*

Lasketaan muistipaikoissa 100-200 olevien lukujen summa rekisteriin R1, jonka jälkeen lopetetaan. Hyödynnetään tässä silmukkaa. Toki voisimme kirjoittaa ADD R1, mem[x] sata kertaa, mutta se olisi aika työlästä. ;-)

| | | |
|---|---|---|
| | **LOAD R2, =100** | Käytetään R2:sta indeksirekisterinä, ja lasketaan summa aloittaen muistipaikasta 200. Toki voitaisiin myös aloittaa muistipaikasta 100, kasvattaa R2:sen arvoa, ja verrata sitä sataan (100 + 100 = 200) joka kierroksella. |
| | **LOAD R1, =0** | Alustetaan R1 nollaksi ennen kuin aletaan laskea summaa. |
| **Alku** | **JNEG R2, Loppu** | Jos R2 on negatiivinen, hypätään kohtaan ”Loppu”. Kuten mainittua, R2 laskee sadasta nollaan. |
| | **ADD R1, 100(R2)** | Lisätään R1:n arvoon muistipaikan mem[100+R2] arvo ja talletetaan summa rekisteriin R1. Varsinainen laskenta siis tapahtuu tässä askeleessa. |
| | **SUB R2, =1** | Vähennetään R2:stä yksi, eli siirrytään yksi muistipaikka ”taaksepäin”. |
| | **JUMP Alku** | Ja aloitetaan uudestaan silmukan alusta. |
| **Loppu** | **SVC SP, =HALT** | Loppu. Kutsutaan käyttöjärjestelmän lopetuskäskyä. |

Luentomonistetta ”Tietokoneen toiminta, Auvo Häkkinen, 1998” mukaillen.

## *Monimutkaisemman summan laskeminen*

Lasketaan summa $\sum_{i=1}^{20} i^2$ . Javan tyyliin lasku sujuisi seuraavasti:

```
s = 0;
for (i = 1; i < 21; i++) s = s + i* i;
```

Matkitaan ylläolevaa koodia symbolisella konekielellä.

| | | |
|---|---|---|
| | **s DC 0** | Alustetaan muuttuja s. |
| | **i DC 1** | Alustetaan muuttuja i. |
| | | |
| **Taas** | **LOAD R1, i** | Ladataan i:n arvo rekisteriin R1. |
| | **MUL R1, R1** | R1 = i * i |
| | **ADD R1, s** | R1 = s + i * i |
| | **STORE R1, s** | Talletetaan rekisterin R1 arvo muuttujaan s, eli s = s + i * i |
| | **LOAD R1, i** | Ladataan muuttujan i arvo rekisteriin R1. |
| | **ADD R1, =1** | Lisätään arvoon 1... |
| | **STORE R1, i** | ja talletetaan takaisin muuttujaan i. |
| | **COMP R1, =21** | Verrataan i:n arvoa (joka siis sijaitsee rekisterissä R1) lukuun 21. |
| | **JLES Taas** | Jos se on vähemmän, palataan silmukan alkuun. |
| | **SVC SP, =HALT** | Loppu. |

Sama ohjelma olisi voitu tehdä käyttäen pelkästään rekistereitä. Tällöin olisi vältytty muutamalta LOAD- ja STORE-käskyltä. Esimerkistä käy ilmi optimoimattoman kääntäjän tyyli generoida koodia.