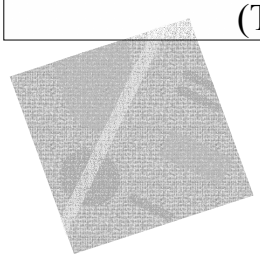
Jakso 3
Konekielinen ohjelmointi (TTK-91, KOKSI)



Muuttujat
Tietorakenteet
Kontrolli
Optimointi
Tarkistukset

Tiedon sijainti suoritusaikana

- Muistissa (=keskusmuisti)
  - iso
  - hidas

  Esim. 256 MB tai 64 milj. 32 bitin sanaa

  Esim. 10 ns
- Rekisterissä
  - pieni
  - nopea

  Esim. 256 B tai 64 kpl 32 bitin sanaa

  Esim. 1 ns

  TTK-91: 8 kpl +PC + ..
- Ongelma: milloin X:n arvo pidetään muistissa, milloin rekisterissä?
  - Missä päin muistia? Miten siihen viitataan?

Miten tietoon viitataan?

- Tieto muistissa
  - Muistiosoitteen (esim. 0x6F123456 tai 3459321) avulla
  - Symbolin (esim. HenkTunn tai X) avulla symbolista konekieltä käytettäessä – symbolin arvo on muistiosoite
    - HenkTunn = 0x6F123456, X = 3459321
- Tieto on välimuistissa
  - Samalla tavalla kuin olisi muistissa
  - Viittaushetkellä ei tiedetä, kummasta paikasta tieto lopulta löytyy tai kauanko viittaamiseen kuluu aikaa!
- Tieto on rekisterissä
  - Rekisterin osoitteen (esim. 6 tai 18) avulla
- Tieto on konekäskyssä (vakiona)
  - Oletusarvoisesti, käskyssä on vain yksi paikka tiedolle

Tieto ja sen osoite (3)

```
X  DC    12
   LOAD  R1, =X
   LOAD  R2, X
```

Muuttujan x osoite on symbolin X arvo

symbolin X arvo

muuttujan X arvo

int x =12;

	230
	12345
	12556
	128765
	12222
X=230:	12
	12998
	muisti

- Muuttujan X osoite on 230
- Muuttujan X arvo on 12
- Symbolin X arvo on 230
  - symbolit ovat yleensä olemassa vain käännösaikana!
  - Virheilmoituksia varten symbolitaulua pidetään joskus yllä myös suoritusaikana

Tieto ja sen osoite (5)

```
Xptr  DC   0
X     DC   12
      LOAD  R1, =X
      STORE R1, Xptr
      LOAD  R2, X
      LOAD  R3, @Xptr
```

Xptr=225:

X=230:

230
12345
12556
128765
12222
12
12998

- Muuttujan X osoite on 230
- Muuttujan X arvo on 12
- Osoitinmuuttujan (pointterin) Xptr <u>osoite</u> on 225
- Osoitinmuuttujan Xptr <u>arvo</u> on 230
- Osoitinmuuttujan Xptr osoittaman kokonaisluvun arvo on 12

Osoitinmuuttujat

- Muuttujia samalla tavoin kuin kokonaislukuarvoiset muuttujatkin
  - muistissa olevalla osoitinmuuttujalla on osoite
- Arvo on jonkun tiedon <u>osoite muistissa</u>
  - globaalin yksi- tai monisanaisen tiedon osoite
    - muuttuja, taulukko, tietue, olio
  - keosta (heap) dynaamisesti (suoritusaikana) varatun tiedon osoite
    - Pascalin tai Javan ”new”-operaatio palauttaa varatun muistialueen osoitteen (tai virhekoodin, jos operaatiota ei voi toteuttaa)
  - aliohjelman tai metodin osoite
    - osoite ohjelmakoodiin!

Globaali, kaikkialla näkyvä data

- Globaalit muuttujat ja muut globaalit tietorakenteet sijaitsevat TTK-91-koneen muistissa ohjelmakoodin jälkeen
  - muuttujat

```
int  X = 25;
short Y;
float  Ft;
```

```
char Ch;
char Str[] = ”Pekka”;
boolean fBig;
```

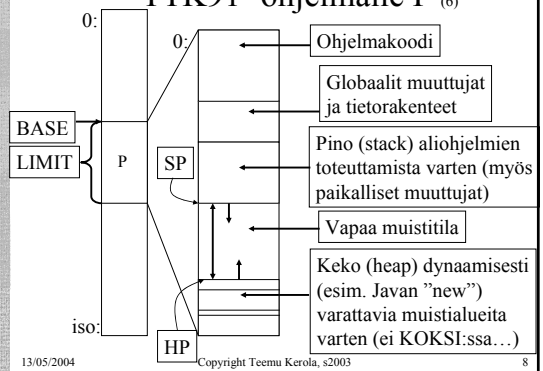
  - tilan varaus

```
X       DC    25  ; alkuarvo = 25
Y       DC    0
fBig    DC    1 ; 1=true, 0=false
```

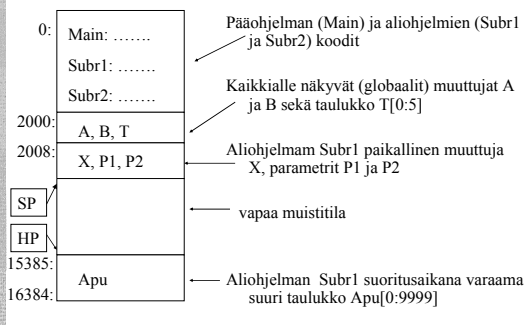
  - viittaaminen

```
        LOAD    R1, X
        STORE   R2, Y
```


Muistitilan käyttö yhdelle TTK91- ohjelmalle P (6)



Esimerkki: ohjelman muistin (osoiteavaruuden) käyttö aliohjelman Subr1 suorituksen aikana



Muistissa oleva data

- Globaali data — Int X; function Print();
  - varataan ohjelman latauksen yhteydessä
  - kaikkialla viitattavissa nimen (osoitteen) avulla
- Dynaaminen data — Mach M = new Mach();
  - varataan tarvittaessa keosta suorituksen aikana
  - vapautetaan, kun ei enää tarvita — Ei KOKSIssa!
  - viittaus varauksen jälkeen osoitteen avulla
- Aliohjelmien paikallinen data
  - varataan pinosta kutsuhetkellä — Parametrit, paik. muuttujat
  - vapautetaan rutiinista poistuttaessa
  - viittaus aliohjelman sisällä osoitteen avulla

Tiedon sijainti suoritusaikana

- Rekisteri (nopein)
  - kääntäjä päättää, milloin muuttujan arvo on rekisterissä
- Välimuisti (nopea)
  - laitteisto hoitaa automaattisesti
- Muisti (hidas)
  - Kääntäjä/lataaja valitsee sijaintipaikan
    - globaali data ohjelman latauksen yhteydessä
    - vakiot konekäskyssä
  - ohjelma sijoittaa suoritusaikana
    - aliohjelmien paikalliset muuttujat, parametrit
  - Käyttöjärjestelmä sijoittaa suorituksen aikana
    - dynaaminen data keosta suorituksen aikana
- Levy, levypalvelin (liian hidas, ei mahdollista)
  - Vaatii käyttöjärjestelmän varusohjelmien apua

Ohjelmoinnin peruskäsitteet

- Aritmeettinen lauseke
  - miten tehdä laskutoimitukset?
- Yksinkertaiset tietorakenteet
  - yksiulotteiset taulukot, tietueet
- Kontrolli - mistä seuraava käsky?
  - valinta: if-then-else, case
  - toisto: for-silmukka, while-silmukka
  - aliohjelmat, virhetilanteet
- Monimutkaiset tietorakenteet
  - listat, moniulotteiset taulukot

Aritmeettinen lauseke (2)

tilan varaus

```
A   DC   0
B   DC   0
C   DC   0
```

```
int  a, b, c;
…
b = 34;

a = b + 5 * c;
```

koodi

```
LOAD   R1, =34
STORE  R1, B
……
LOAD   R1, B
LOAD   R2, C
MUL    R2, =5
ADD    R1, R2
STORE  R1, A
```

tai:

```
LOAD R1,=5
MUL R1, C
ADD R1, B
STORE R1, A
```


Globaalin taulukon tilan varaus ja käyttö (2)

```
int X, Y;
intTaulu [30];
…
X = 5;
Y = Taulu[X];
```

```
X      DC   0
Y      DC   0
Taulu  DS   30
       …
       LOAD   R1, =5
       STORE  R1, X
       LOAD   R1, X
       LOAD   R2, Taulu(R1)
       STORE  R2, Y
```

X:
Y:
Taulu:

Optimoiva kääntäjä osaisi jättää pois jälkimmäisen "LOAD R1,X" käskyn

Globaalien tietueiden tilan varaus ja käyttö (3)

```
int X;
struct Tauno{
    int Pituus,
    int Paino;
   }
…
X = Tauno.Paino
```

Kentän "Paino" suhteellinen osoite tietueen Tauno sisällä

```
X       DC   0
Tauno   DS   2
Pituus  EQU  0
Paino   EQU  1
…
LOAD R1,=Tauno
LOAD R2, Paino(R1)
STORE R2, X
```

Tietueen osoite on sen ensimmäisen sanan osoite

X:
Tauno: pituus
paino

Kontrolli - valinta konekielellä (4)

- Ehdoton hyppy
  - JUMP, CALL ja EXIT, SVC ja IRET
- Hyppy perustuen laiterekisterin arvoon (verrataan 0:aan)
  - JZER, JPOS, ...
- Hyppy perustuen aikaisemmin asetetun tilarekisterin arvoon
  - COMP
  - JEQU, JGRE, …
  - Ongelma vai etu: ttk-91:ssä kaikki ALU-käskyt asettavat tilarekisterin
    - ADD, SUB, MUL, DIV, NOT, AND, OR, XOR, SHL, SHR

```
COMP  R2, LIMIT
JEQU  LOOP
```


If-then-else -valinta

```
if (a<b)
    x = 5;
else
    x = y;
```

```
        LOAD   R1, A
        COMP   R1, B
        JNLE   Else
        LOAD   R1, =5
        STORE  R1, X
        JUMP   Done
Else    LOAD   R1, Y
        STORE  R1, X
Done    NOP
```

?

```
        LOAD  R2, Y
        LOAD  R1, A
        COMP  R1, B
        JNLES Else
        LOAD  R2, =5
Else    STORE R2,X
```

Vai olisiko tämä parempi?

Case lauseke (1)

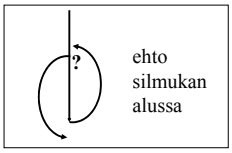
```
Switch (lkm) {
 case 4:  x = 11;
          break;

 case 0:  break;

 default: x = 0;
          break;
}
```

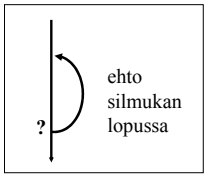
```
Swi    LOAD R1, Lkm

Vrt4   COMP  R1,=4
       JNEQ  Vrt0
       LOAD  R2, =11
       STORE R2, X
       JUMP  Cont

Vrt0   COMP  R1, =0
       JNEQ  Def
       JUMP  Cont

Def    LOAD  R2,=0
       STORE R2, X

Cont   NOP
```

Onko case-tapausten järjestyksellä väliä?

Toistolausekkeet (2)

- For-step-until -silmukka
- Do-until -silmukka
- Do-while -silmukka
- While-do -silmukka
- …





For lauseke (1)

```
for (int i=20; i < 50; ++i)
        T[i] = 0;
```

```
I       DC      0
        …
        LOAD  R1, =20
        STORE R1, I
Loop    LOAD  R2, =0
        LOAD  R1, I
        STORE R2, T(R1)
        LOAD  R1, I
        ADD   R1, =1
        STORE R1, I
        LOAD  R3, I
        COMP  R3, =50
        JLES  Loop
```

Olisiko parempi pitää i:n arvo rekisterissä? Miksi? Milloin?

Mikä on i:n arvo lopussa? Onko sitä olemassa?

While-do -lauseke (1)

```
X = 14325;
Xlog = 1;
Y = 10;
while (Y < X) {
        Xlog++;
        Y = 10*Y
        }
```

```
        LOAD   R1, =14325
        STORE R1, X
        LOAD  R1, =1   ; R1=Xlog
        LOAD  R2, =10 ; R2=Y
While   COMP  R2, X
        JNLES  Done
        ADD    R1, =1
        MUL    R2, =10
        JUMP   While
Done    STORE  R1, Xlog ; talleta tulos
        STORE  R2, Y
```

Mitä kannattaa pitää muistissa?

Mitä kannattaa pitää rekisterissä ja milloin?

Koodin generointi (9)

- Kääntäjän viimeinen vaihe
  - voi olla 50% käännösajasta
- Tavallinen koodin generointi
  - alustukset, lausekkeet, kontrollirakenteet
- Optimoidun koodin generointi
  - käännös kestää (paljon) kauemmin
  - suoritus tapahtuu (paljon) nopeammin
  - Milloin globaalin muuttujan X arvo kannattaa pitää rekisterissä ja milloin ei?
  - Missä rekisterissä X:n arvo kannattaa pitää?

Optimoitu For- lauseke (3)

```
for (int i=20; i < 50; ++i)
        T[i] = 0;
```

```
        LOAD  R1, =20 ; i
        LOAD  R2, =0  ; 0
Loop    STORE R2, T(R1)
        ADD   R1, =1
        COMP  R1, =50
        JLES  Loop
```

Mitä eroja? Onko tämä OK?

6 vs 11 konekäskyä (koodin koko)
122 vs. 272 suoritettua käskyä!
152 vs. 452 muistiviitettä!
Muuttujan i arvo lopussa?

Alkuperäinen koodi

```
I       DC      0
        …
        LOAD  R1, =20
        STORE R1, I
Loop    LOAD  R2, =0
        LOAD  R1, I
        STORE R2, T(R1)
        LOAD  R1, I
        ADD   R1, =1
        STORE R1, I
        LOAD  R3, I
        COMP  R3, =50
        JLES  Loop
```


Virhetilanteisiin varautuminen (3)

- Suoritin tarkistaa käskyn suoritusaikana
  - ”automaattinen”
  - integer overflow, divide by zero, ...

```
ADD   R1, R2  ; overflow??
DIV   R4, =0  ; divide-by-zero
```

- Generoidut konekäskyt tarkistavat ja explisiittisesti aiheuttavat keskeytyksen tai käyttöjärjestelmän palvelupyynnön tarvittaessa
  - ”manuaalinen”
  - index out of bounds, bad method, bad operand, ihan mitä vain haluat testata!

```
        COMP  R1, Tsize    ; indeksin rajatarkistus
        JLES  IndexOK
        SVC   SP, =BadIndex  ; käyttöjärj. huolehtii
IndexOK ADD   R2, Taulu(R1)  ; R1 = 12 345 000 ??
```

Taulukon indeksitarkistus

```
for (int i=20; i < 50; ++i)
        T[i] = 0;
```

```
I       DC    0

T       DS    50
Tsize   DC    50  ;koko
        …
```

```
        LOAD  R1, =20
        STORE R1, I
Loop    LOAD  R2, =0
        LOAD  R1, I
        JNNEG R1, ok1
        SVC   SP,=BadIndex
ok1     COMP  R1,Tsize
        JLES  ok2
        SVC   SP,=BadIndex
ok2     STORE R2, T(R1)
        LOAD  R1, I
        ADD   R1, =1
        STORE R1, I ; 50 OK!
        LOAD  R3, I
        COMP  R3, =50
        JLES  Loop
```

Voisiko loopin kontrollia ja indeksin tarkistusta yhdistää? Optimoiva kääntäjä osaa!

Taulukon alaindeksi ei ala nollasta (ei animoitu)

```
for (int i=20; i < 50; ++i)
        T[i] = 0;
```

```
I       DC    0

T       DS    30 ; 30 alkiota
Tlow    DC    20 ;alaraja
Thigh   DC    50 ;yläraja+1
        …
```

T: T[20]
T+1: T[21]
T+29: T[49]

indeksitarkistukset...

Taulukon alaindeksi ei ala nollasta (2)

```
for (int i=20; i < 50; ++i)
        T[i] = 0;
```

```
I       DC    0

T       DS    30 ; 30 alkiota
Tlow    DC    20 ;alaraja
Thigh   DC    50 ;yläraja+1
        …
```

indeksitarkistukset...

```
        LOAD  R1, =20
        STORE R1, I

Loop    LOAD  R2, =0
        LOAD  R1, I
        SUB   R1, Tlow
        STORE R2, T(R1)

        LOAD  R1, I
        ADD   R1, =1
        STORE R1, I

        LOAD  R3, I
        COMP  R3, =50
        JLES  Loop
```


Moniulotteiset taulukot

- Ohjelmointikieli voi tukea suoraan moniulotteisia taulukoita

```
X= Tbl[i,j];
```

```
Y = Arr[k][6][y+2];
```

- Toteutus konekielitasolla aina (useimmissa arkkitehtuureissa) yksiulotteinen taulukko
  - konekäskyissä vain yksi indeksirekisteri!
- Moniosainen toteutus
  - laske alkion osoite yksiulotteisessa taulukossa ja käytä indeksoitua osoitusmoodia tiedon viittaukseen
  - TAI: laske alkion osoite muistissa ja käytä epäsuoraa tiedonosoitusmoodia

2-ulotteiset taulukot (6)

```
int[][]  T = new int[4] [3];
…
Y = T[i][j];
```

```
T       DS    12
Trows   DC    4
Tcols   DC    3
        …
        LOAD  R1, I
R1      MULT  R1, Tcols
        ADD   R1, J
R1
        LOAD  R2, T(R1)
        STORE R2, Y
```

Esimerkki I: 1, J: 2

T:

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2
3,0	3,1	3,2

looginen

T: T[0][0], T[0][1], T[0][2], T[1][0], T[1][1], T[1][2], T[2][0], T[...][...]

fyysinen

Tarkistukset…. ?

Moniulotteiset taulukot (3)

- Talletus riveittäin
  - C, Pascal, Java?
- Talletus sarakkeittain
  - Fortran
- 3- tai useampi ulotteiset
  - samalla tavalla!

T: T[0][0], T[0][1], T[0][2], T[1][0], T[1][1], T[1][2], T[2][0], T[...][...]

T: T[0][0], T[1][0], T[2][0], T[3][0], T[0][1], T[1][1], T[2][1], T[...][...]

Linkitetty lista (6)

R1: -1
R2: 132

First → Data Next → Data Next → Data Next

2-sanainen tietue

First=200: 211

R1 → 211: 33
R2: 0 255

R1 → 222: 55
R2: 77 -1

R1 → 255: 44
R2: 33 222

list_sum.k91

```
Data    EQU   0 ; suht. osoite
Next    EQU   1
Sum     DC    0
Main    LOAD  R1, First ; ptrRec
        JNEG  R1, Done
        LOAD  R2,=0  ; sum
Loop    ADD   R2, Data(R1)
        LOAD  R1, Next(R1)
        JNNEG R1, Loop
Done    STORE R2, Sum
        SVC   SP, =HALT
```


Monimutkaiset tietorakenteet

- 2-ulotteinen taulukko T, jonka jokainen alkio on tietue, jossa neljä kenttää:
  - pituus
  - ikä
  - viime vuoden palkka kunakin kuukautena
  - viime vuoden töissäolopäivien lukumäärä kunakin kuukautena
- Talletustapa?
- Viitteet?
- Tarkistukset?

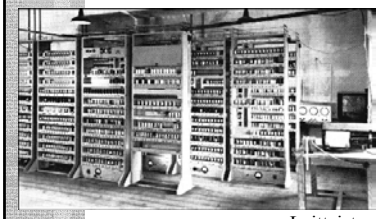
```
X = T[yliopNum][opNum].palkka[kk];
```


EDSAC

(Electronic Delay Storage Automatic Computer)

- Ensimmäinen toimiva ”todellinen” tietokone
  - ohjelma ja data samassa muistissa
  - Maurice Wilkes, Cambridge University
  - 1949
  - 256 sanan muisti
    - elohopeasäiliöteknologia
  - 35-bitin sanat

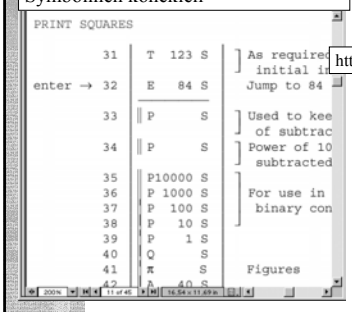
EDSAC

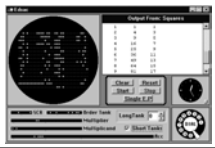


Laitteisto Muisti

EDSAC Simulator

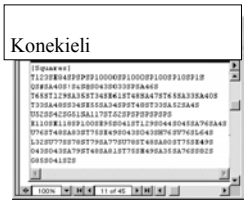
Symbolinen konekieli





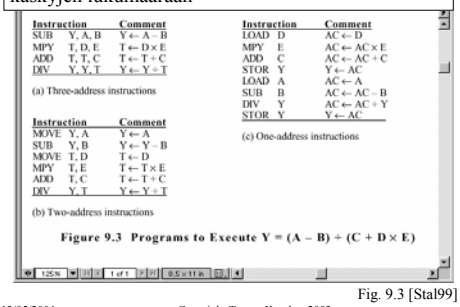
http://www.dcs.warwick.ac.uk/~edsac/

Konekieli



-- Jakson 3 loppu --

Konekielen operandien lukumäärän vaikutus käskyjen lukumäärään



Instruction	Comment
SUB Y, A, B	Y ← A − B
MPY T, D, E	T ← D × E
ADD T, T, C	T ← T + C
DIV Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction	Comment
MOVE Y, A	Y ← A
SUB Y, B	Y ← Y − B
MOVE T, D	T ← D
MPY T, E	T ← T × E
ADD T, C	T ← T + C
DIV Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction	Comment
LOAD D	AC ← D
MPY E	AC ← AC × E
ADD C	AC ← AC + C
STOR Y	Y ← AC
LOAD A	AC ← A
SUB B	AC ← AC − B
DIV Y	AC ← AC ÷ Y
STOR Y	Y ← AC

(c) One-address instructions

Figure 9.3 Programs to Execute Y = (A − B) ÷ (C + D × E)

Fig. 9.3 [Stal99]