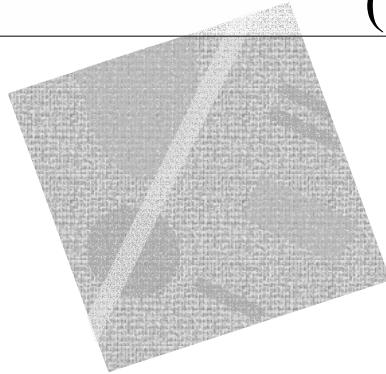


Jakso 3

Konekielinen ohjelmointi (TTK-91, KOKSI)



Muuttujat
Tietorakenteet
Kontrolli
Optimointi
Tarkistukset

Tiedon sijainti suoritusaikana

- Muistissa (=keskusmuisti)
 - iso

Esim. 256 MB tai 64 milj. 32 bitin sanaa
--
 - hidas

Esim. 10 ns

- Rekisterissä
 - pieni

Esim. 256 B tai 64 kpl 32 bitin sanaa

 - nopea

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

13/05/2004

Copyright Teemu Kerola, s2003

3

Muuttujan x osoite on symbolin X arvo

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

Tieto ja sen osoite ⁽³⁾

symbolin X arvo

muuttujan X arvo

230
12345
12556
128765
12222
12
12998

```
int x =12;
```

- Muuttujan X osoite on 230
- Muuttujan X arvo on 12
- Symbolin X arvo on 230 X=230:
 - symbolit ovat yleensä olemassa vain käännoa aikana!
 - Virheilmoituksia varten symbolitaulua pidetään joskus yllä myös suoritus aikana

muisti

13/05/2004

Copyright Teemu Kerola, s2003

4

```

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

```

Tieto ja sen osoite ⁽⁵⁾

Xptr=225:	230
	12345
	12556
	128765
	12222
X=230:	12
	12998

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

13/05/2004

Copyright Teemu Kerola, s2003

5

Osoitinmuuttujat

- Muuttujia samalla tavoin kuin kokonaislukuarvoiset muuttujatkin
 - muistissa olevalla osoitinmuuttujalla on osoite
- Arvo on jonkun tiedon osoite muistissa
 - 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!

13/05/2004

Copyright Teemu Kerola, s2003

6

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

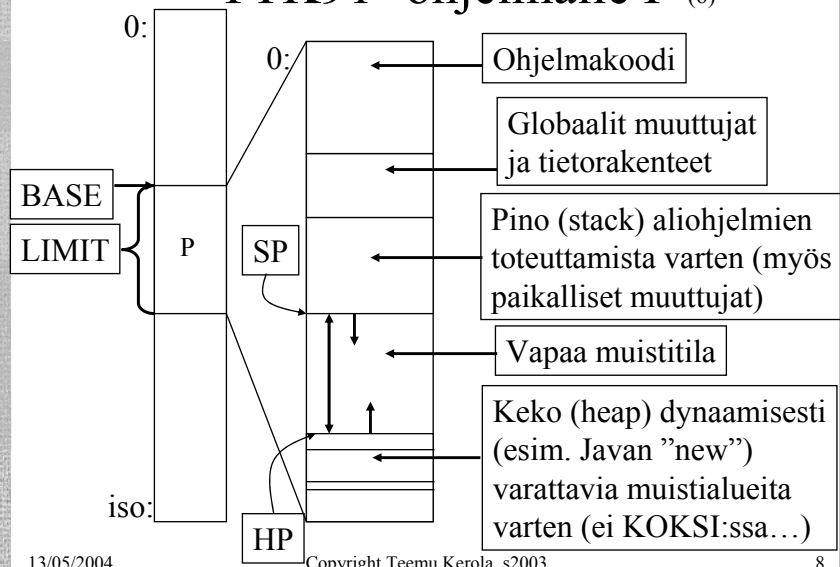
```
LOAD    R1, X  
STORE   R2, Y
```

13/05/2004

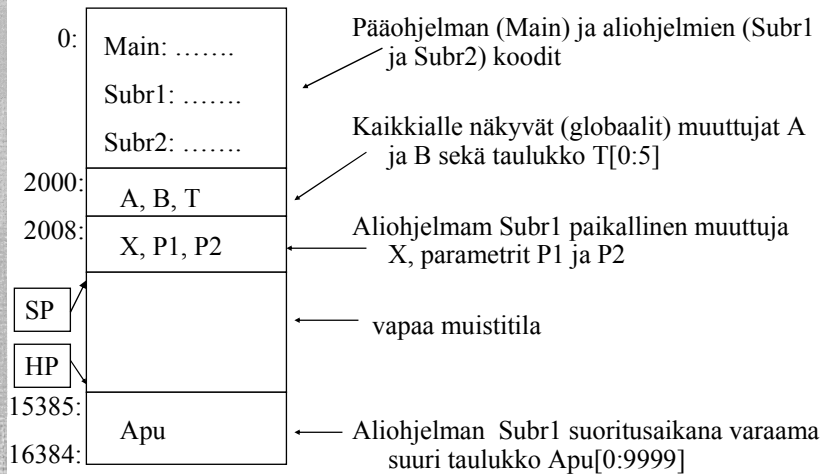
Copyright Teemu Kerola, s2003

7

Muistitilan käyttö yhdelle TTK91- ohjelmalle P ⁽⁶⁾



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



13/05/2004

Copyright Teemu Kerola, s2003

9

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** `Parametrit, paik. muuttujat`
 - varataan pinosta kutsuhetkellä
 - vapautetaan rutiinista poistuttaessa
 - viittaus aliohjelman sisällä osoitteen avulla

13/05/2004

Copyright Teemu Kerola, s2003

10

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

13/05/2004

Copyright Teemu Kerola, s2003

11

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

13/05/2004

Copyright Teemu Kerola, s2003

12

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
```

13/05/2004

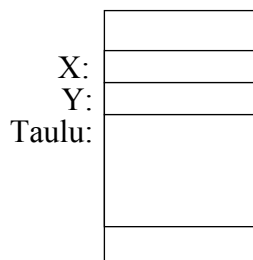
Copyright Teemu Kerola, s2003

13

Globaalin taulukon tilan varaus ja käyttö (2)

```
int X, Y;
int Taulu [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
```

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

13/05/2004

Copyright Teemu Kerola, s2003

14

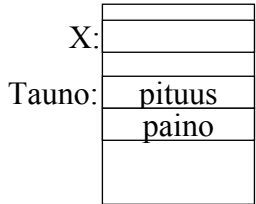
Globaalien tietueiden tilan varaus ja käyttö (3)

```
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



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;
```

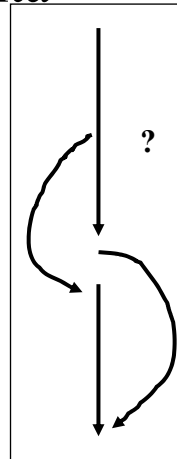
```
LOAD R1, A
COMP R1, B
JNLE Else

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
STORE R2, X

Else
```

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
```

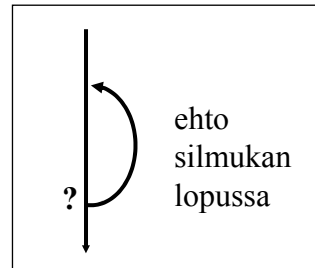
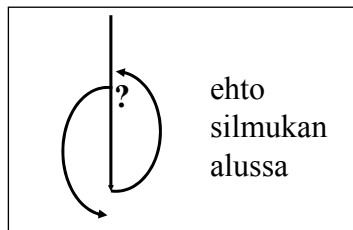
```
Def   LOAD R2, =0
      STORE R2, X
```

```
Cont  NOP
```

Onko case-tapausten järjestyksellä väliä?

Toistolausekkeet (2)

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

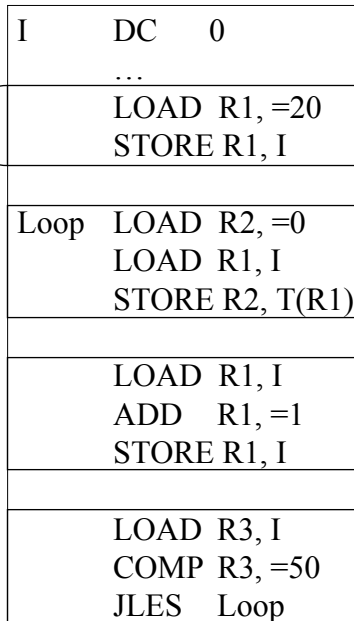


For lauseke (1)

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

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

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



While-do -lauseke (1)

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

Mitä kannattaa pitää muistissa?

Mitä kannattaa pitää rekisterissä ja milloin?

	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

13/05/2004

Copyright Teemu Kerola, s2003

21

Koodin generointi (9)

- Kääntäjän viimeinen vaihe
 - voi olla 50% käänösajasta
- Tavallinen koodin generointi
 - alustukset, lausekkeet, kontrollirakenteet
- Optimoidun koodin generointi
 - käänös 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?

13/05/2004

Copyright Teemu Kerola, s2003

22

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
```

Kerola, s2003

23

Virhetilanteisiin varautuminen ⁽³⁾

- Suoritin tarkistaa käskyn suoritusaikana
 - ”automaattinen”
 - integer overflow,
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 ??
```

13/05/2004

Copyright Teemu Kerola, s2003

24

Taulukon indeksitarkistus

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

```
I      DC    0

T      DS   50
Tsize  DC   50 ;koko
...

```

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

13/05/2004

Copyright Teemu Kerola, s2003

25

```

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
    
```

Taulukon alaindeksi ei alasta 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...

13/05/2004

Copyright Teemu Kerola, s2003

26

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

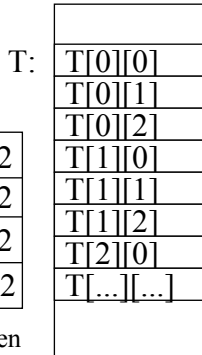
2-ulotteiset taulukot (6)

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

```
T      DS      12
Trows  DC      4
Tcols  DC      3
```

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



```
...
LOAD R1, I
R1 MULT R1, Tcols
R1 ADD R1, J
LOAD R2, T(R1)
STORE R2, Y
```

looginen

Tarkistukset.... ?

13/05/2004

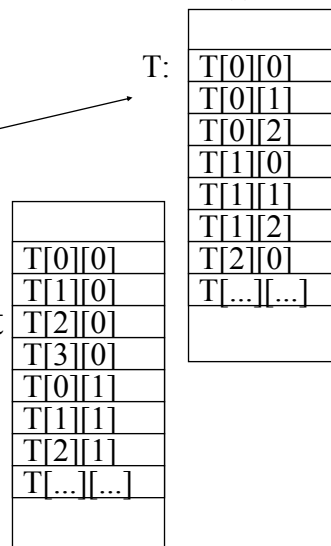
fyysinen

Copyright Teemu Kerola, s2003

29

Moniulotteiset taulukot (3)

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



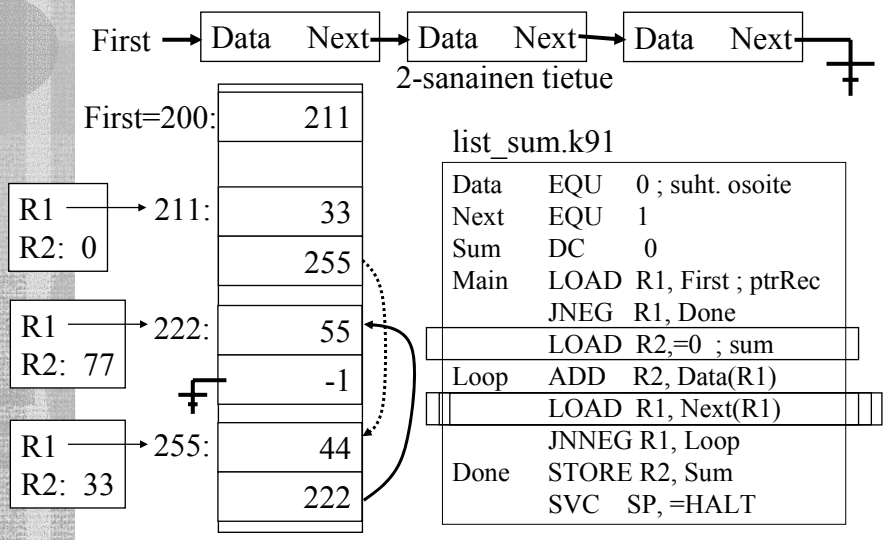
13/05/2004

Copyright Teemu Kerola, s2003

30

R1: -1
R2: 132

Linkitetty lista (6)



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? $X = T[\text{yliopNum}][\text{opNum}].\text{palkka}[\text{kk}];$
- Tarkistukset?

EDSAC

(Electronic Delay Storage Automatic Computer)

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

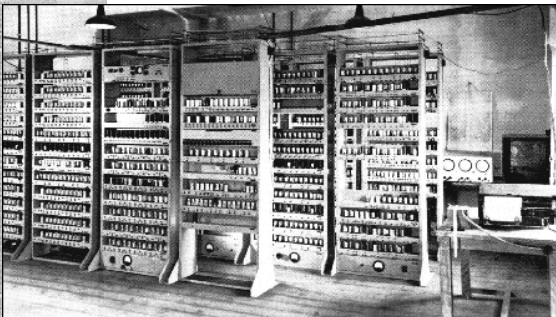


13/05/2004

Copyright Teemu Kerola, s2003

33

EDSAC



Laitteisto



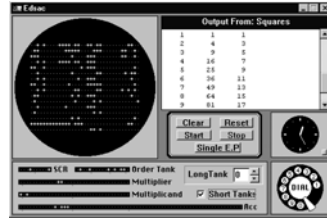
(b) Mercury delay lines or "long tanks" for the main memory, with M. V. Wilkes looking on. The battery of 16 tanks shown here had a capacity of 512 words - the equivalent of a little over 1 Kilobyte.

13/05/2004

Copyright Teemu Kerola, s2003

34

EDSAC Simulator



Symbolinen konekieli

```

PRINT SQUARES
31 | T 123 S ] As required
enter → 32 | E 84 S ] initial in
33 | || P S ] Used to kee
34 | || P S ] of subtrac
35 | P10000 S ] Power of 10
36 | P 1000 S ] subtracted
37 | P 100 S ] For use in
38 | P 10 S ] binary con
39 | P 1 S ]
40 | Q S ]
41 | π S ] Figures
42 | A 40 S ]
    
```

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

Konekieli

```

[ Squares ]
T123SE84SPSP1000SP1000SP100SP10SP1SP
Q98SA40S19438S043S033P5A46S
T65ST129SA35ST34SE1ST48SA47ST65SA33SA40S
T93SA48S834SE58A34SPST48ST93SA52SA4S
U52S842SG51SA1178T52SPSPSPSPSP
E110SE118SP100SE95S041ST129S044S048SA76SA4S
U76ST48SA83ST75SA49S043S043SH76SV76SL64S
L32SU778S78ST79SA77SU78T48SA80ST75SA49S
O43S043SA79ST48SA81ST75SE49SA35SA76S8S2S
G8S8041S2S
    
```

13/05/2004

Copyright Teemu Kerola, s2003

35

-- Jakson 3 loppu --

Konekielen operandien lukumäärän vaikutus käskyjen lukumäärään

Instruction	Comment
SUB Y, A, B	$Y \leftarrow A - B$
MPY T, D, E	$T \leftarrow D \times E$
ADD T, T, C	$T \leftarrow T + C$
DIV Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Instruction	Comment
MOVE Y, A	$Y \leftarrow A$
SUB Y, B	$Y \leftarrow Y - B$
MOVE T, D	$T \leftarrow D$
MPY T, E	$T \leftarrow T \times E$
ADD T, C	$T \leftarrow T + C$
DIV Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

Instruction	Comment
LOAD D	$AC \leftarrow D$
MPY E	$AC \leftarrow AC \times E$
ADD C	$AC \leftarrow AC + C$
STOR Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
SUB B	$AC \leftarrow AC - B$
DIV Y	$AC \leftarrow AC \div Y$
STOR Y	$Y \leftarrow AC$

(c) One-address instructions

Figure 9.3 Programs to Execute $Y = (A - B) \div (C + D \times E)$

Fig. 9.3 [Stal99]

13/05/2004

Copyright Teemu Kerola, s2003

36