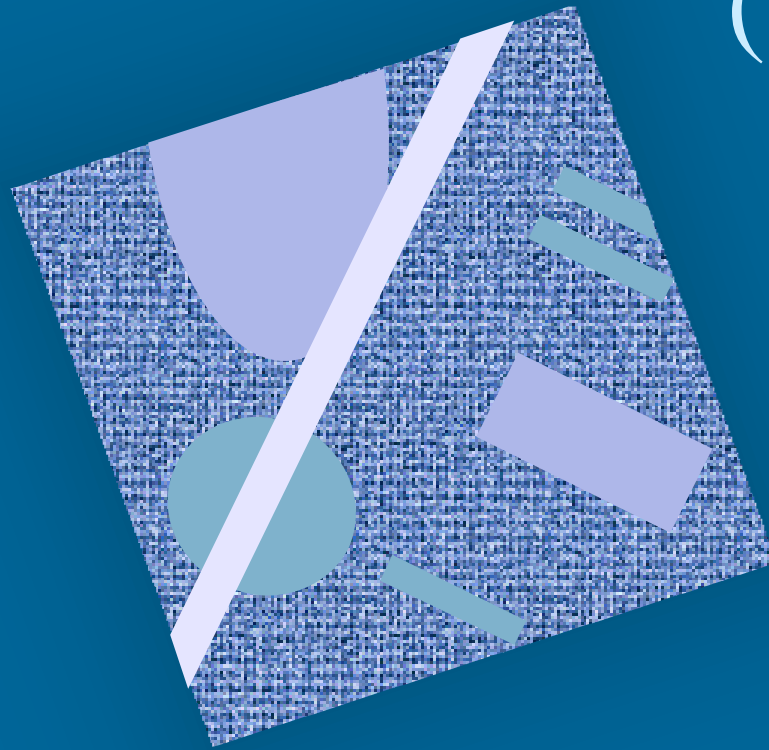


Jakso 3

Konekielinen ohjelmointi (TTK-91, KOKSI)



Muuttujat
Tietorakenteet
Kontrolli
Optimointi
Tarkistukset

Tieto ja sen osoite ⁽³⁾

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

```
int x =12;
```

symbolin X arvo

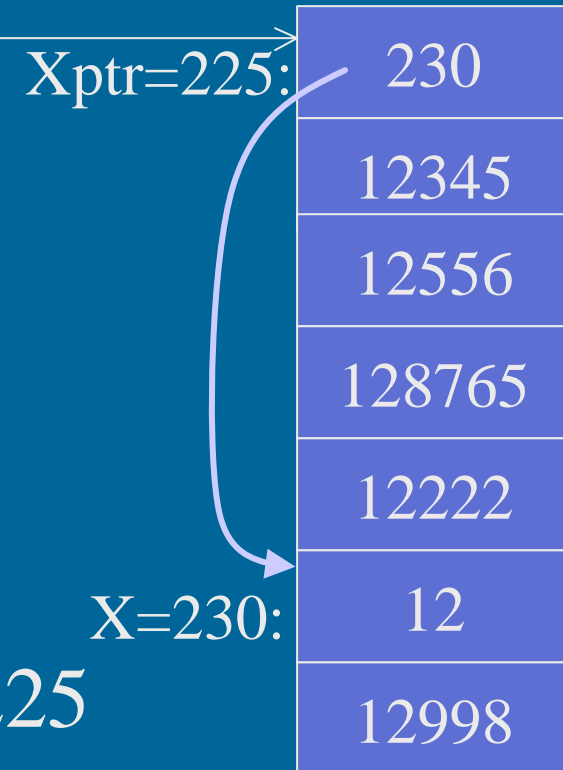
muuttujan X arvo

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

230
12345
12556
128765
12222
12
12998

Tieto ja sen osoite (5)

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



- 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

Osoitinmuuttujat

- Muuttujia samalla tavoin kuin kokonaislukuarvoiset muuttujatkin
- Arvo on jonkun tiedon osoite muistissa
 - globaalin monisanaisen tiedon osoite
 - taulukot, tietueet, oliot
 - kasasta (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 data ⁽²⁾

- Globaalit muuttujat ja muut tietorakenteet sijaitsevat 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

Aritmeettinen lauseke (2)

```
int a, b, c;
```

```
...
```

```
b = 34;
```

```
a = b + 5 * c;
```

tilan varaus

A	DC	0
B	DC	0
C	DC	0

koodi

LOAD	R1, =34
STORE	R1, B
<hr/>	
LOAD	R1, B
LOAD	R2, C
MUL	R2, =5
ADD	R1, R2
STORE	R1, A

Globaalin taulukon tilan varaus ja käyttö (2)

```
int X, Y;  
int [] Taulu = new int [30];  
...  
X = 5;  
Y = Taulu[X];
```

X:
Y:
Taulu:



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
```

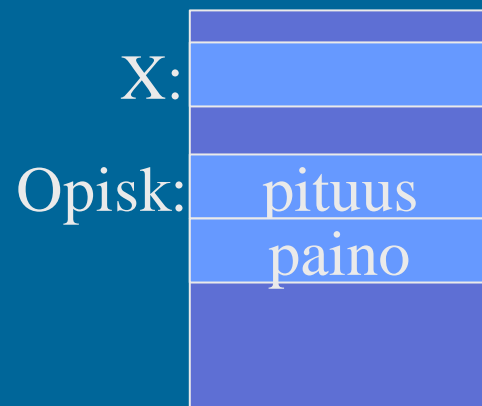
Globaalien tietueiden tilan varaus ja käyttö ⁽³⁾

```
int X;  
public class Opisk  
    { public int Pituus, Paino; }  
...  
Opisk Tauno = new Opisk ();  
...  
X = Tauno.Paino
```

Kentän "Paino"
suhteellinen osoite
tietueen Opisk sisällä

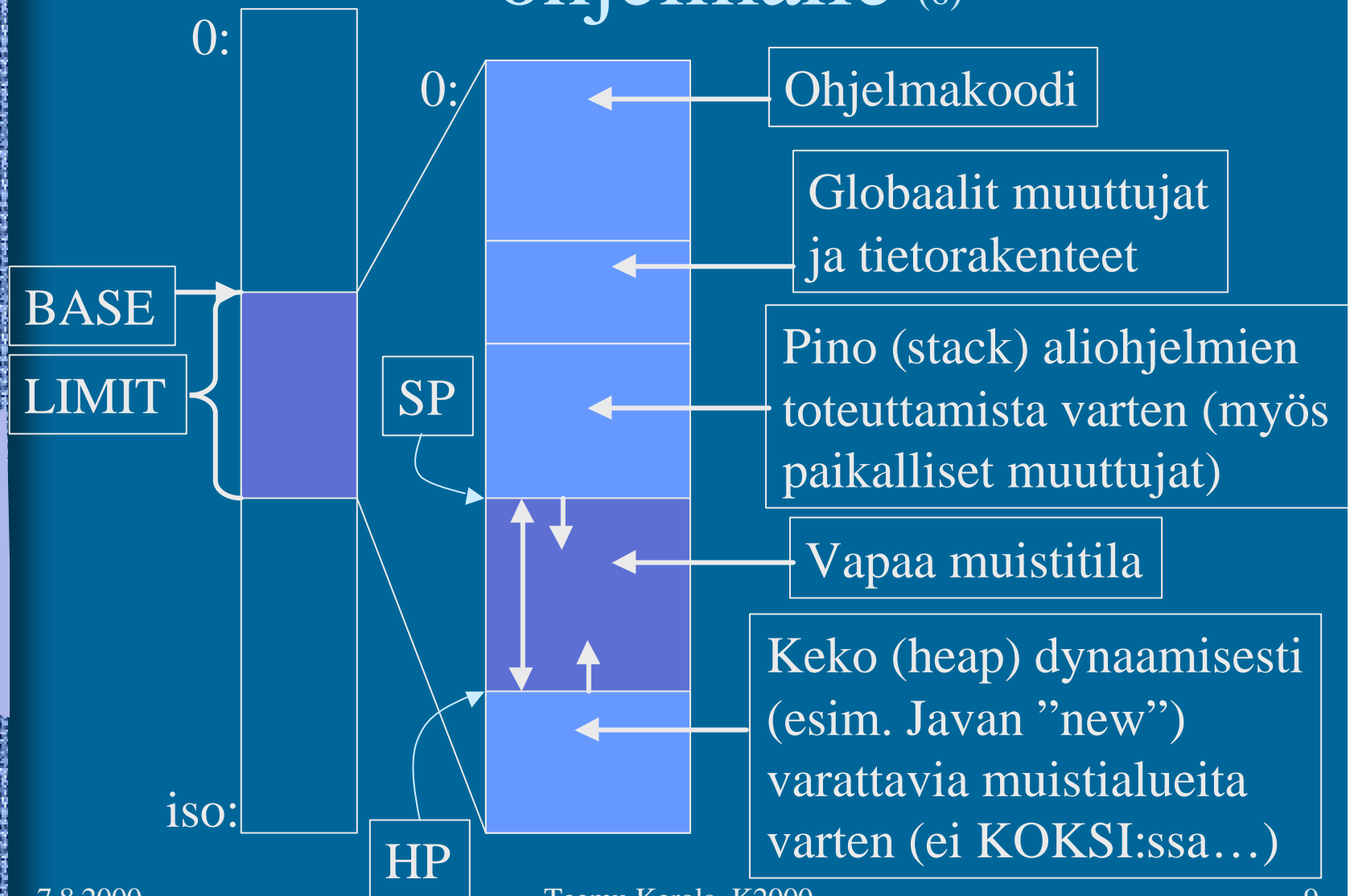
X	DC	0
Opisk	DS	2
Pituus	EQU	0
Paino	EQU	1

Tietueen
osoite
on sen
ensimmäisen
sanan osoite



...
LOAD R1,=Opisk
LOAD R2, Paino(R1)
STORE R2, X

Muistitilan käyttö yhdelle ohjelmalle (6)



Kontrolli - valinta konekielellä (4)

- Ehdoton hyppy
 - JUMP, CALL, EXIT, SVC
- Hyppy perustuen laiterekisterin arvoon
 - 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
```

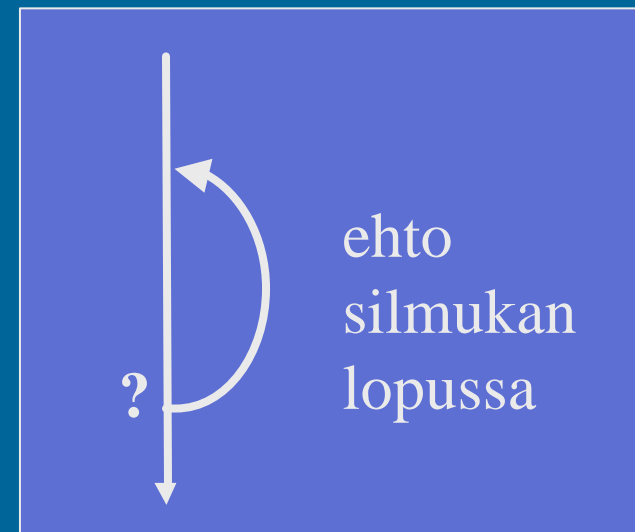
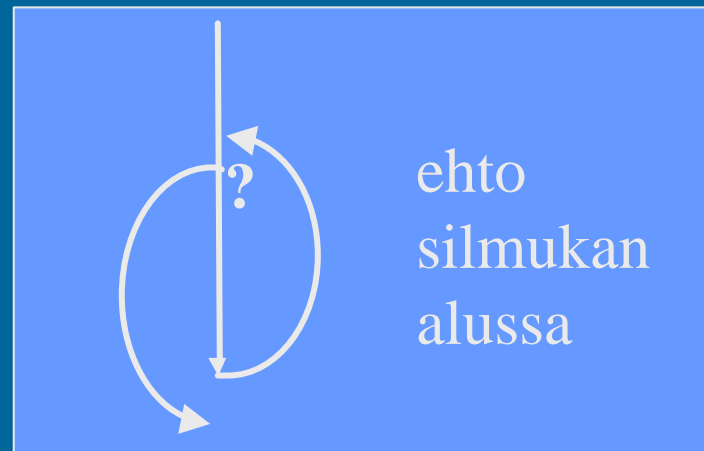
If-then-else -valinta (1)

```
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	

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
```

While-do -lauseke (1)

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

```
LOAD R1, =14325  
STORE R1, X  
LOAD R3, =1 ; R3=Xlog  
LOAD R2, =10 ; R2=Y
```

```
While COMP R2, R1  
JNLES Done
```

```
ADD R3, =1  
MUL R2, =10
```

```
JUMP While
```

```
Done STORE R3, Xlog ; talleta tulos  
STORE R2, Y
```

Case lauseke (1)

```
Switch (lkm) {  
  case 4: x = 11;  
         break;  
  
  case 0: break;  
  
  default: x = 0;  
         break;  
}
```

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

Koodin generointi ⁽⁹⁾

- Kääntäjän viimeinen vaihe
 - voi olla 50% käänösajasta
- Tavallinen koodin generointi
 - alustukset, lausekkeet, kontrollirakenteet
- Optimoidun koodin generointi
 - käänös kestää kauemmin
 - suoritus tapahtuu 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
```

122 vs. 272
suoritettua käskyä!

Mitä eroja? Onko tämä OK?

```
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

- 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

- index out of bounds, bad method,
bad operand, ihan mitä vain haluat testata!

```
LOAD R3, Tsize      ; tarkista  
COMP R1, R3  
JLES IndexOK  
SVC  SP, =BadIndex
```

```
Index OKADD      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
...
```

```
Loop   LOAD R1, =20
        STORE R1, I
        LOAD R2, =0
        LOAD R1, I
ok1    JNNEG R1, ok1
        SVC    SP(=BadIndex)
        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 ala nolasta ⁽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...

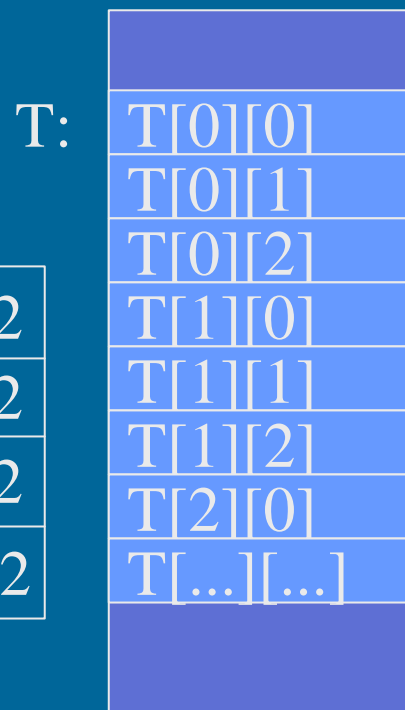
```
LOAD R1, =20  
STORE R1, I  
  
Loop  LOAD R2, =0  
      LOAD R1, I  
      SUB  R1, Tlow  
      STORE R2, T(R1)  
  
      LOAD R1, I  
      ADD  R1, =1  
      STORE R1, I  
  
      LOAD R3, I  
      COMP R3, =50  
      JLES Loop
```

2-ulotteiset taulukot (6)

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

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

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

Tarkistukset.... ?

Moni-ulotteiset taulukot ⁽³⁾

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

T:

T[0][0]

T[1][0]

T[2][0]

T[3][0]

T[0][1]

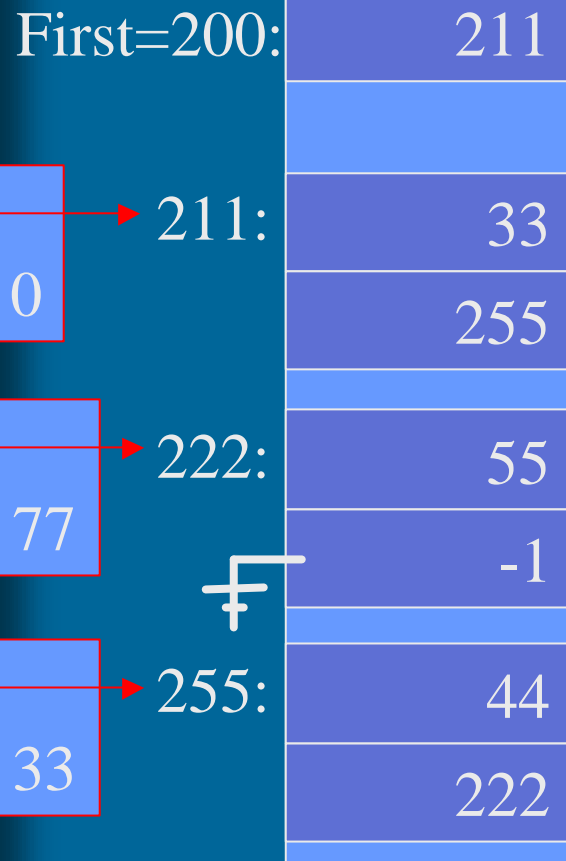
T[1][1]

T[2][1]

T[...][...]

Linkitetty lista (6)

R1: -1
R2: 132



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

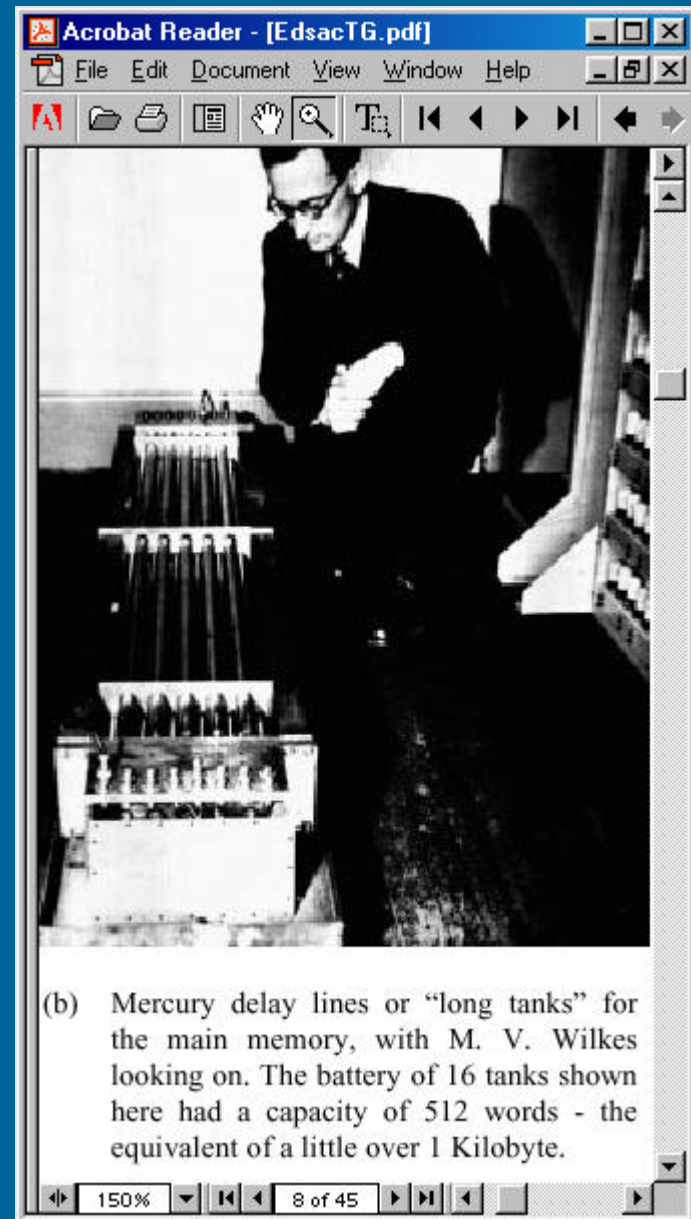
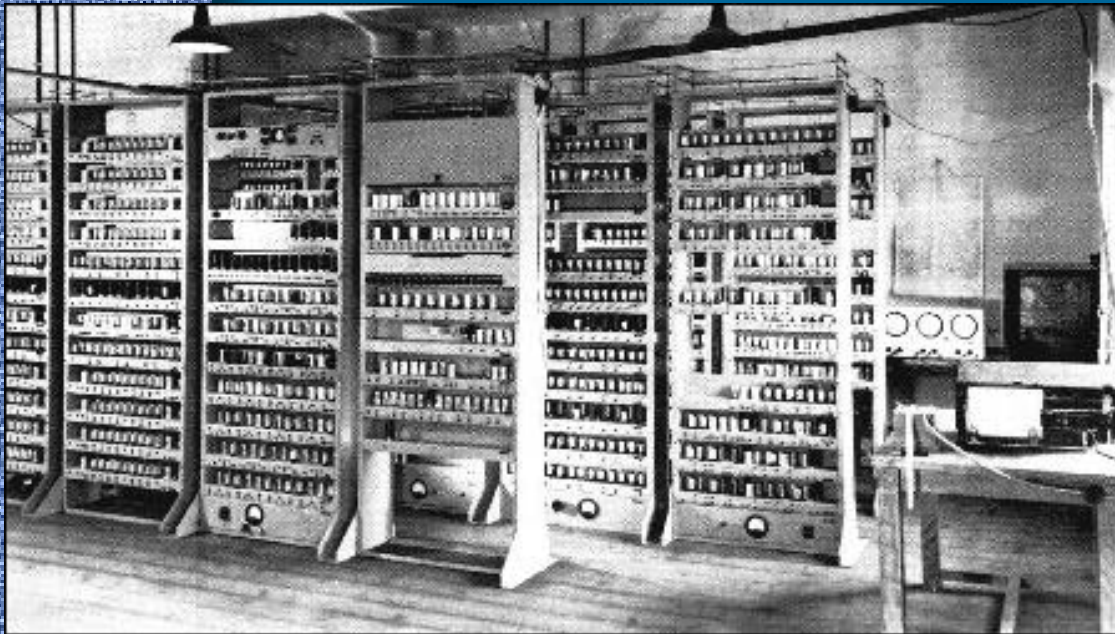
EDSAC

(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



(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.

Laitteisto

Muisti

EDSAC Simulator

Symbolinen konekieli

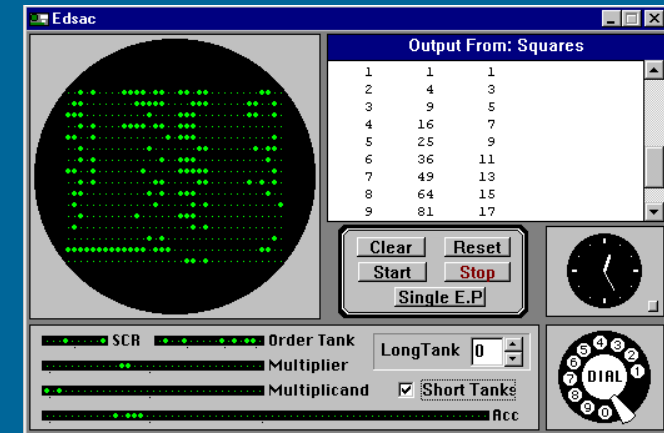
```

PRINT SQUARES

31 T 123 S ] As required
              initial in
enter → 32 E 84 S ] Jump to 84

33 || P S ] Used to kee
              of subtrac
34 || P S ] Power of 10
              subtracted

35 || P10000 S ]
36 || P 1000 S ] For use in
37 || P 100 S ] binary con
38 || P 10 S ]
39 || P 1 S ]
40 || Q S ]
41 || π S ] Figures
42 || A 40 S ]
    
```



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

Konekieli

```

[Squares]
T123SE84SPSPSP10000SP1000SP100SP10SP1S
QS#SA40S!S&S@S043S033SPSA46S
T65ST129SA35ST34SE61ST48SA47ST65SA33SA40S
T33SA48S34SE55SA34SPST48ST33SA52SA4S
U52SS42SG51SA117ST52SPSPSPSPSPS
E110SE118SP100SE95S041ST129S044S045SA76SA4S
U76ST48SA83ST75SE49S043S043SH76SV76SL64S
L32SU77SS78ST79SA77SU78ST48SA80ST75SE49S
043S043SA79ST48SA81ST75SE49SA35SA76SS82S
G85S041S2S
    
```

-- Jakson 3 loppu --

Konekielen operandien lukumäärän vaikutus käskyjen lukumäärään

<u>Instruction</u>		<u>Comment</u>		<u>Instruction</u>	<u>Comment</u>
SUB	Y, A, B	$Y \leftarrow A - B$		LOAD	$AC \leftarrow D$
MPY	T, D, E	$T \leftarrow D \times E$		MPY	$AC \leftarrow AC \times E$
ADD	T, T, C	$T \leftarrow T + C$		ADD	$AC \leftarrow AC + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$		STOR	$Y \leftarrow AC$

(a) Three-address instructions

<u>Instruction</u>		<u>Comment</u>		<u>Instruction</u>	<u>Comment</u>
MOVE	Y, A	$Y \leftarrow A$		LOAD	$AC \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$		SUB	$AC \leftarrow AC - B$
MOVE	T, D	$T \leftarrow D$		DIV	$AC \leftarrow AC \div Y$
MPY	T, E	$T \leftarrow T \times E$		STOR	$Y \leftarrow AC$
ADD	T, C	$T \leftarrow T + C$			
DIV	Y, T	$Y \leftarrow Y \div T$			

(b) Two-address instructions

(c) One-address instructions

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

Fig. 9.3 [Stal99]