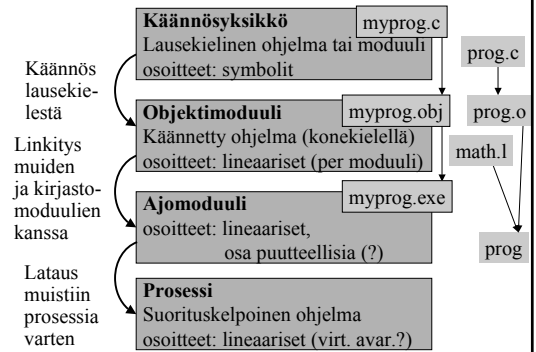


# Luento 10

## Käännös, linkitys ja lataus

Käännös  
Linkitys  
Dynaaminen linkitys  
Lataus

## Lausekielestä suoritukseen (3)

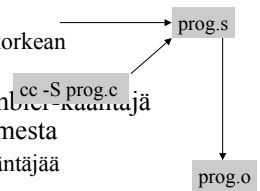


## Käännösyksikkö (4)

- Jollain ohjelmointikielellä kuvattu eheä kokonaisuus, joka halutaan aina kääntää yhdessä
  - kaikki yhteen liittyvät aliohjelmat
  - olioperustainen luokka
- Liian suuri kokonaisuus?
  - turhaa aikaa kääntämiseen joka muutoksen jälkeen
- Liian pieni kokonaisuus?
  - turhaa aikaa liitoksien suunnitteluun ja toteutukseen muiden moduulien kanssa
- Käännösyksikön ohjelmointikieli ei ole tärkeä
  - niiden sitominen yhteen tapahtuu objektimoduulien tasolla

## Assembler-kielinen käännösyksikkö (2)

- Käännösyksikkö voi olla myös suoraan k.o. koneen symbolisella konekielellä kirjoitettu
  - suoraan käsin
  - kääntäjän generoimana korkean tason kielestä
- Käännöksen tekee assembler-kääntäjä tavallisen kääntäjän asemesta
  - yleensä osa tavallista kääntäjää



## Objektimoduuli (8)

- Konekielinen koodi
  - moduulin sisäiset viitteet paikallaan (linearisessa muistiavaruudessa)
  - moduulin ulkopuoliset viitteet merkitty
- Linkitystä varten:
  - tiedot niiden osoitteiden sijainneista, jotka täytyy paikoittaa RELOCATION TABLE
  - osoiteavaruus yhdistetään jonkin toisen moduulin osoiteavaruuden kanssa linkityksessä
  - tiedot viittauksista moduulin ulkopuolelle
  - tiedot kohdista, joista tähän moduuliin saavutetaan ulkopuolelta IMPORT
  - symbolitaulu EXPORT

SYMBOL TABLE

## Symbolitaulu

- Kääntäjä generoi
- Ylläpidetään linkityksen aikana
- Joskus ylläpidetään myös latauksen jälkeen virheilmoitusten tekemistä varten
  - ohjelmien kehitysympäristöt ylläpitävät symbolitaulua koko ajan
- Jätetään pois valmiista ohjelmasta
  - vie turhaa tilaa

## Lähdekieli vs. konekieli <sup>(3)</sup>

- Pascal lauseke:  $N := I+J;$
- C lauseke:  $N = I+J$
- Java lauseke:  $N = I+J;$

TTK-91 symbolinen  
konekieli:

I	DC	3
J	DC	4
N	DC	0
FORMULA	LOAD R1, I	
	ADD R1, J	
	STORE R1, N	

Pentium II, Motorola 680x0 ja  
SPARC symbolinen konekieli:

ks. Fig. 7-2 [Tane99]

04/04/2002

Teemu Kerola, Copyright 2002

7

## (Assembler) kääntäjän ohjauskäskyt <sup>(4)</sup>

- Eivät varsinaista koodia  
– niistä ei tule konekäskyjä
- Ohjaavat käännöstä

TTK-91: DC  
DS  
EQU

Pentium II:

ks. Fig. 7-3 [Tane99]

04/04/2002

Teemu Kerola, Copyright 2002

8

## Makrot <sup>(6)</sup>

- Helpottavat ohjelmointia
- Usein toistuville koodisarjoille annetaan nimi  
⇒ makro
- Makroilla voi olla parametreja  
– useimmiten nimiparametreja (call-by-name)
- Makrot käsitellään ennen kääntämistä  
– eivät kuulu konekieleen  
– makron ”kutsu” (käyttö) korvataan makron rungolla
- Esimerkkejä  
– swap  
– aliohjelmien prologi ja epilogi  
– itse tehdyt, kääntäjän käyttämät
- Erot aliohjelmiin

ks. Fig. 7-4 ja 7-6 [Tane99]

ks. Fig. 7-5 [Tane99]

04/04/2002

Teemu Kerola, Copyright 2002

9

## Literaali <sup>(5)</sup>

- Vakioita
- Niin suuria, että eivät mahdu konekäskyn vakio-osaan ...  
ttk-91: käskyn vakiot 2-tavuisia, arvoalue: -32767 ... 32767
- ... tai muuten vain halutaan pitää datan joukossa eikä käskyjen yhteyteen talletettuna  
Pi DC 3.14159265 ; (!!!?)  
One DC 1 vrt. One EQU 1  
OneMeg DC 1024576
- Niitä ei saisi muuttaa  
LOAD R1, One  
ADD R1,=1  
STORE R1, One ; ask for trouble

04/04/2002

Teemu Kerola, Copyright 2002

10

## Literaali <sup>(2)</sup>

- Korkean tason kielissä kaikki isot vakiot ovat literaaleja  $N := 35000;$   $\text{var myStr} = \text{”literal”}$   
– kääntäjän pitäisi estää literaalien muuttamisen  
FortranX: 5 = 6; LOAD R1, six  
STORE R1, five ???
- literaalia ei saisi välittää viiteparametrina  
• aliohjelma voisi muuttaa sen arvoa? Java string?
- Myös joissakin assemblerkielissä literaalien implisiittinen (automaattinen) määrittely  
– helpommin luettavaa koodia Load R14, =F’234567’  
– literaalien 234567 tilanvaraus automaattisesti

04/04/2002

Teemu Kerola, Copyright 2002

11

## Assembler käännös <sup>(10)</sup>

- 1. vaihe:
  - laske käskyjen tilanvaraukset
    - ttk-91 helppoa, koska kaikki käskyt 4 tavua!
  - generoi symbolitaulu  
ks. Kuva 6.2 [Häkk98]
    - arvot, arvon vaatima tavumaara
    - uudelleensijoitustiedot (omana tauluna?)
  - generoi tai käytä muita tauluja
    - literaalitaulu (tilanvaraus lopuksi)
    - kääntäjän ohjauskäskytaulu
    - operaatiokooditaulu

04/04/2002

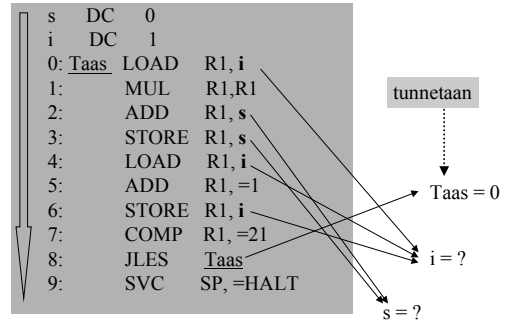
Teemu Kerola, Copyright 2002

12

# Assembler käänös (8)

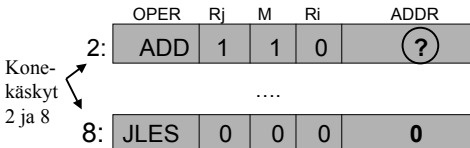
- 2. vaihe
  - generoi lopullinen objektimoduuli ks. Kuva 6.3 [Häkk98]
  - tulosta symbolinen konekielinen listaus ks. Fig. 7-16 [Tane99]
  - generoi taulut linkitystä varten
    - osana objektimoduulia
  - anna virheilmoitukset
- 3. vaihe
  - koodin optimointi
  - voi olla oikeasti ennen 2. vaihetta tai sen yhteydessä

# TTK-91 Assembler käänös, 1. vaihe

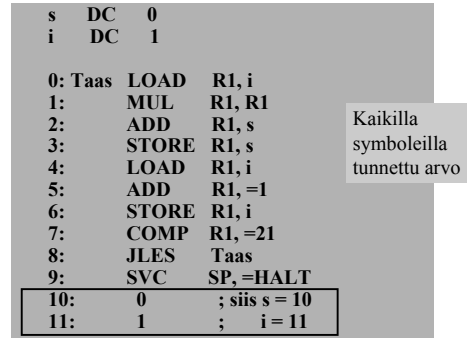


Symbolitaulu 1. vaiheen aikana ks. kalvo 14

Symboli	tyyppi	arvo	uud. sij.tietoa
s	data	?	2, 3
i	data	?	0, 4, 6
Taas	viite	0	8
HALT	vakio	11	



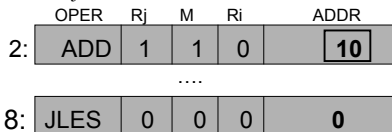
Koodi & data 1. vaiheen jälkeen



Symbolitaulu 1. vaiheen jälkeen: ks. kalvo 16

Symboli	tyyppi	arvo	uud. sij.tietoa
s	data	10	2, 3
i	data	11	0, 4, 6
Taas	viite	0	8
HALT	vakio	11	

Koodi 2. vaiheen jälkeen:



# TTK-91 objektimoduuli



( Kuva 6.3 [Häkk98] )

## TTK-91 objektimoduuli

- Moduulin otsakeosa
  - moduulin nimi
  - linkittäjän tarvitsemia tietoja
    - objektimoduulin osien pituudet
    - käännös päivämäärä
    - kääntäjän nimi ja versio
    - ensimmäisen suoritettavan käskyn osoite
      - ellei aina 0

04/04/2002

Teemu Kerola, Copyright 2002

19

## TTK-91 objektimoduuli

- EXPORT-hakemisto
  - tunnukset, joihin voidaan viitata muista moduuleista
    - rutiinit, aliohjelmat, (oliot, metodit)
    - yhteiskäyttöinen data
  - tunnuksen osoite (= symbolin arvo)
  - mahdollinen käyttöoikeus
    - R/W/E/RW

04/04/2002

Teemu Kerola, Copyright 2002

20

## TTK-91 objektimoduuli

- IMPORT-hakemisto
  - muissa moduuleissa määritellyt tunnukset
    - tunnus
    - niiden käskyjen osoitteet, jossa tunnus esiintyy
- Koodi ja alustettu data
  - alustamattomille muuttujille ei tarvitse varata (vielä) tilaa, mutta ne on otettava huomioon data-alueen koossa

04/04/2002

Teemu Kerola, Copyright 2002

21

## TTK-91 objektimoduuli

- Uudelleensijoitushakemisto
  - niiden käskyjen osoitteet, joiden osoiteosaa on korjattava, kun siirrytään moduulin yhteiseen osoiteavaruuteen
    - suoraviivainen lisäys (joka käskyyn) ei toimi, sillä käskyn osoiteosa voi olla vakio, jota ei saa muuttaa
    - erikseen (a) paikalliseen dataan viittaavat ja (b) hyppykäskyt, sillä linkittäessä yhdistetään erikseen data- ja koodialueet

04/04/2002

Teemu Kerola, Copyright 2002

22

## Korkean tason kielen käännös <sup>(7)</sup>

- Enemmän vaiheita
  - Syntaktisten alkiodien etsintä (front end)
    - Syntaksipuun generointi ja jäsenys
  - Lauseiden tunnistaminen syntaksipuun avulla
  - Välikielen (välikoodin) generointi (ei aina)
    - Välikieliesitys ja symbolitaulut
  - Koodin generointi (back end)
    - ei (yleensä) Java-ohjelmille

Lisää tietoa?



Kääntäjien ja ohj. kielten kurssit

ks. syntaksipuun jäsenyspuu esimerkit

04/04/2002

Teemu Kerola, Copyright 2002

23

## Linkitys

- Uudelleensijoitusongelma (relocation problem)
  - jokaisen objektimoduulin osoitteet alkavat 0:sta
  - tulosmoduulissa kaikki yhdessä lineaarisessa osoiteavaruudessa
  - useimpien moduulien kaikkia osoitteita täytyy muuttaa
    - käskyjen osoitteet
    - datan osoitteet

04/04/2002

Teemu Kerola, Copyright 2002

24

## Linkitys esimerkki <sup>(4)</sup>

- Neljä moduulia: A, B, C ja D ks. Fig. 7-14 [Tane99]
- Laske joka moduulille uudelleen-sijoitusvakio (moduulin alkuosoite) (relocation constant)
- Lisää k.o. vakio kunkin moduulin sisäisiin viitteisiin
- Etsi kaikki moduulien väliset viitteet, ks. Fig. 7-15 (a) [Tane99]  
ja aseta kyseisten viitteiden osoitteet oikein ks. Fig. 7-15 (b) [Tane99]

04/04/2002

Teemu Kerola, Copyright 2002

25

## Muuttujan X viittausten päivitys <sup>(3)</sup>

- Miten löytää linkityksen aikana kaikki kohdat, jossa muuttujaan X viitataan?
- Vastaus 1: taulukko, jossa kaikki kohdat listattu
  - taulukko voi olla hyvin iso
  - kaikille muuttujille tilaa maksimitarpeen verran?
- Vastaus 2: Muuttujan X viittaukset on linkitetty keskenään linkitetyksi listaksi objektimoduulissa
  - vain linkitetyn listan alkuosoite taulukossa (tarvitaan vain yksi osoite per muuttuja)
  - X:n osoitteen paikalla aluksi linkki seuraavaan käskyyn, missä X:ään viitataan
  - listan voi käyttää vain yhden kerran?

04/04/2002

Teemu Kerola, Copyright 2002

26

## Muuttujan X viittaukset linkitettyinä listana <sup>(1)</sup>

symbolitaulu, moduuli ABC			
Symb	sij	1. viittaus	
X	700	23	

lähdekoodi, moduuli ABC			
23:	Load	R1, X	
...			
34	Store	R3, X(R1)	
...			
555	Add	R4, X	
...			
700	DC	0 ; X	

objektimoduuli			
23:	Load	1 0	34
...			
34:	Store	3 1	555
...			
555:	Add	4 0	-1

Diagram showing arrows from the 'objektimoduuli' table to the 'symbolitaulu' table. An arrow points from the '1. viittaus' column (23) to the 'Symb' column (X). Another arrow points from the 'objektimoduuli' table to the 'lähdekoodi' table, specifically from the '34' and '555' rows to the 'Store' and 'Add' instructions respectively.

04/04/2002

Teemu Kerola, Copyright 2002

27

## Staattinen linkitys <sup>(5)</sup>

- Tavallinen (staattinen) linkitys vaatii, että kaikki ohjelmakoodissa viitatut moduulit ja kirjastorutiinit on linkitetty ennen suoritusta
- Ajomoduulista tulee hyvin iso
  - mukana myös paljon moduuleja, joihin ei yhdeällä suorituskerralla tule lainkaan viittauksia
    - esim: kääntäjässä koodin optimointikoodi, vaikka koodin optimointia ei suoriteta joka kerta
    - esim: pelissä tasojen 8-22 moduulit, kun aloittelija ei pääse tasoa 3 ylemmäksi vielä kuukausiin

04/04/2002

Teemu Kerola, Copyright 2002

28

## Dynaaminen linkitys <sup>(3)</sup>

- Jätetään linkityksessä kutsukohdat muihin moduuleihin auki
- Pienempi ajomoduuli, mutta hitaampi suorittaa
- Viittaus ”ratkaisemattomaan” (eli ei-linkitettyyn) moduuliin ratkotaan suoritusaikana
  - suoritus keskeytyy ja puuttuva moduuli linkitetään paikalleen (kaikki viittaukset siihen korjataan kuntoon)

04/04/2002

Teemu Kerola, Copyright 2002

29

## Windows DLL <sup>(4)</sup>

- DLL - Dynamically Linked Library
  - koodia, dataa tai molempia .dll yleinen tapaus
- Säästää tilaa myös yhteiskäytön vuoksi .drv driver  
.fon font
- Helpompi korjata virheitä ks. Fig. 7.19 [Tane99]
  - ei tarvita uutta käännöstä eikä lähdekielistä koodia!
  - riittää kun DLL vaihdetaan uuteen
  - seuraavassa suorituksessa uusi versio käyttöön
- Ajomoduuli kootaan kuten tavallinen objektimoduuli
  - DLL moduulit ja DLL viittaukset merkitty erikoislipukkeella (huomioidaan linkityksen yhteydessä)

04/04/2002

Teemu Kerola, Copyright 2002

30

## Windows DLL:n linkityksen kaksi tapaa <sup>(3)</sup>

- Epäsuora dynaaminen linkitys
- Suora dynaaminen linkitys
- DLL:ssä oleva koodi suoritetaan osana kutsuvaa prosessia käyttäen sen omaa aktivointitietuepinoa

## DLL:n epäsuora dynaaminen linkitys (implicit linking)

- Kaikki viitatus moduulit ladataan (lataus aloitetaan) virtuaalimuistiin ja niihin viitataan staattisesti linkitetyn pienemmän liitospalikan (import library) avulla

```
...  
call stubS  
...  
stubS: "wait until load S done"  
call S  
exit
```

## DLL:n suora dynaaminen linkitys

(explicit linking)

- Koodiin generoidaan suoraan viitepaikalle käskyt, joiden avulla linkitys tapahtuu tarvittaessa
- DLL ladataan vain jos siihen tulee viittaus

```
...  
"link S"  
"load S"  
call S  
...
```

## Nimien sidonta <sup>(2)</sup>

(name binding)

- Milloin symbolin L suoritusaikainen muistiosoite tai muu lopullinen arvo sidotaan (lasketaan valmiiksi)?
  - ohjelman kirjoitusaikana?
  - käännoaikana?
  - linkityksessä?
  - latauksessa?
  - kantarekisterin asetuksen aikana?
  - osoitteen sisältämän konekäskyn suoritusaikana?
- Jos muuttujan sijaintipaikka siirretään sitomisen jälkeen, mennään metsään ...

≠  
virtuaaliosoite

## Sijainnista riippumaton koodi <sup>(3)</sup>

(position independent code)

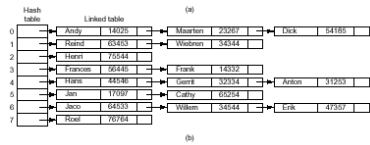
- Jos koodi siirretään toiseen paikkaan, niin mitään osoitetta ei tarvitse päivittää
- Kaikki muistiviittaukset ovat
  - absoluuttisia (esim. keskeytys käsittelijän osoite),
  - suhteessa PC:hen, tai
  - pinossa
- Siellä ei ole viittauksia mihinkään koodiin tai tietorakenteeseen suorien (fysisten) muistiosoitteiden avulla (tähän koodisegmenttiin)

## Lataus <sup>(4)</sup>

- Ajomoduulista luodaan suorituskelpoinen prosessi (rakennetaan PCB ja sen viitteet kuntoon)
- Prosessin koodialueet (tai ainakin sen pääohjelma) ja tarvittava data-alue ladataan muistiin, prosessi siirretään R-to-R jonoon
- Sitten kun prosessi saa suoritusvuoron suorittimella, MMU ja laiterekisterit ladataan PCB:n avulla tämän prosessin tiedoilla
  - virtuaalimuistia käytettäessä joidenkin nimien sidonta tehdään viime hetkellä (konekäskyn suoritusaikana) MMU:n avulla

# -- Luennon 10 loppu --

Andy	14026	0
Arto	3123	43
Cathy	05264	5
Dick	24185	0
Erik	47357	0
Francis	96446	7
Frank	14332	3
Gert	32334	4
Hari	65428	43
Helen	75544	2
Jan	17097	6
Jaco	65533	6
Marion	23287	0
Nancy	64125	1
Rosal	76764	7
William	35544	6
Wendy	34344	1



**Figure 7-12.** Hash coding. (a) Symbols, values, and the hash codes derived from the symbols. (b) Eight-entry hash table with linked lists of symbols and values.

[Tane99]