Kodegenerering del 2: tilleggsnotat , INF5110 – v2006

Arne Maus,
Ifi UiO

Pensumoversikt - kodegenerering

8.1 Bruk av mellomkode

8.2 Basale teknikker for kode-generering

8.3 Kode for referanser til datastrukturer (ikke 8.3.2)

8.4 Kode for generereing for kontroll-setninger og logiske uttrykk

(8.5 Kode-generering for prosedyrer og kall)

---------(resten ikke pensum)—

8.6 Kode produsert av to kommersielle kompilatorer

8.7 TM: En enkel maskin

8.8 Kode-gen for Tiny på TM

8.9 og 8.10 Optimalisering

+Pensum:

En del fra utdelt kap 9 fra Aho, Sethi og Ullmann ' s kompilatorbok ("Drage-boka") :

9.2, 9.4, 9.5 og 9.6

Maskinen det oversettes til

- To-adress-intruksjoner:

op source dest ← *angir et register eller en lagercelle (med ett unntak)*

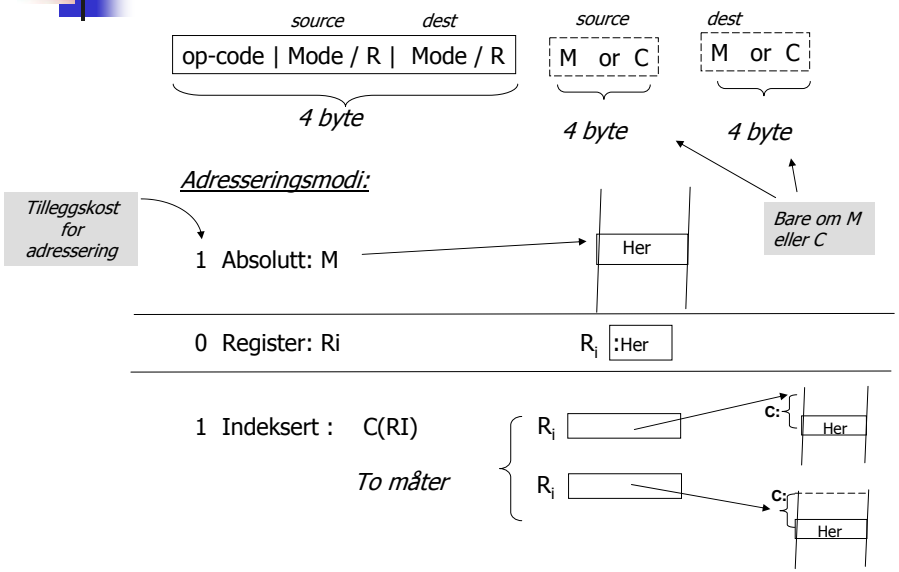
hva

mange forskjellige ulike former, se neste foil

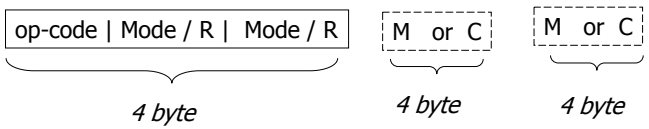
ADD a b
SUB a b
MUL a b
……..
GOTO l
+ Betingete hopp
+ Prosedyrekall
++

Mer er en CISC-maskin enn en RISC-maskin

Instruksjonsformat og adressemodi – del 1

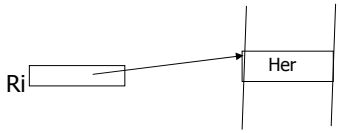


Instruksjonsformat og adressemodi – del 2

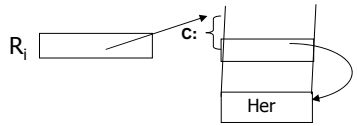


Adresseringsmodi:

0 Indirekte Register *R



1 Indirekte *C(Ri)



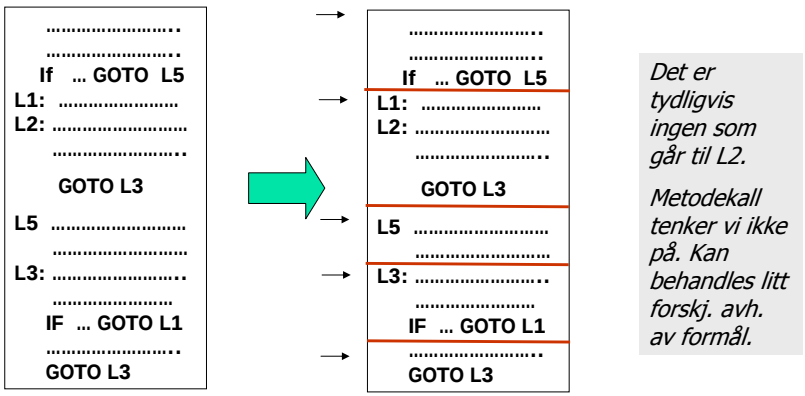
1 Literal #M bare for source

9.4 Basale blokker og Flyt-grafer

- En Flyt-graf er en graf der nodene er sekvenser av treadresse-setninger
- Nodene i grafen er kalt Basale Blokker
  - En basal blokk representer en sekvens av treadresse
  - Programkontrollen kommer alltid inn i første setning og går sekvensielt gjennom hver setning uten stopp eller sidesprang. Eneste unntak er at siste setning kan være et hopp – muligenens betinget.
- Kantene i grafen representerer de mulige veier programflyten-kontrollen kan ta
- Innen en Basal Blokk er det lett å holde oversikt over hvor ting er etc,. og lett å gjøre "abstrakt interpretasjon".

Eksempel på oppdeling i basale blokker

- Basale blokker: Fra og med en leder fram til neste
- Algoritme:
  - Første setning er leder
  - en "goto i", gjør setning "i" til en leder
  - setninger etter "goto .." er ledere



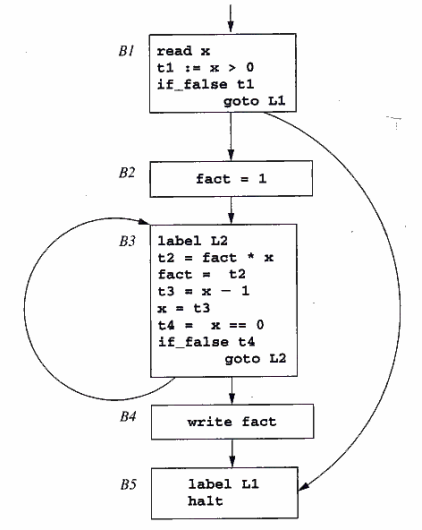
Det er tydligvis ingen som går til L2.

Metodekall tenker vi ikke på. Kan behandles litt forskj. avh. av formål.

Flyt-graf

- Nodene i flyt-grafen er de basale blokkene med en initiell node (den første basale blokken i programmet).
- Det er en rettet kant fra blokk B_1 til B_2 hvis B_2 kan følge direkte etter B_1 i en eller annen utførelse. Det vil si at det enten er:
  - En betinget eller ubetinget goto fra siste setning i B_1til første setning i B_2 eller
  - B_2 følger direkte etter B_1 i programmet, og B_1 ender ikke i en ubetinget goto.
- Vi sier at B_1 er en forgjenger til B_2 og B_2 er en etterfølger til B_1

Flytgraf fra Louden 8.9
oftest: En flytgraf for hver metode for hver metode



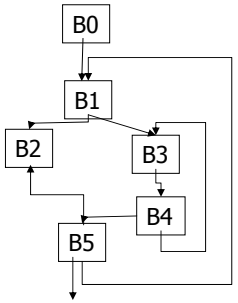
En goto-setning eller if-goto-setning vil alltid være siste setning setning i sin basale blokk (men ikke alle basale blokker slutter slik!)

- Metodekall kan plasseres litt forskjellig avh. av grafens bruk
  - Danne egne basale blokker
  - Kan ligge først i en basal blokk?

Løkker i flyt-grafer

- Bruk, for eksempel :
  - Kan vi flytte beregninger ut av løkka?
    while (i<n) { i= i + 2*n; A[i] = n/2; }
  - Kan vi holde ofte brukte variable i registre mens vi er i løkka?

Anta følgende graf av Basale Blokker:



En løkke er et utplukk L av noder slik at:

1) Dersom Bx ∈L og By ∈L, så går det en rettet vei fra Bx til By av lengde ≥ 1.

2) L har en "inngang": Det finnes bare én B ∈L slik at Bn → B og Bn∉L.

{B3,B4} og {B1,B2,B3,B4,B5} er løkker

{B1,B2,B5} er ikke løkke (!?)

Grunnen for 2): Praktisk: Ett sted å initialisere løkke & Ett sted om vi skal flytte noe "ut av løkka"

Hva er "liveness" ("i live") ?

- Er uavhengig av basale blokker
- Defineres i 9.4 og brukes i 9.5
  - Terminologi:

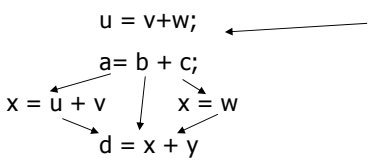
```
a := x + y;
if (x<a) goto L;
```

Her "defineres" a, og her "brukes" x og a.

Intuitiv def

En variabel x er "levende"/"i live" på et gitt sted i programmet dersom den verdien den der har har kan bli brukt i en eller annen utførelse.

Def. som kan avgjøre om x er levende



Stedet "i"
Er "x" i live her ?

Svaret er "ja", om det finnes en TA-setning "j" slik at det er minst én eksekveringsvei fra "i" til "j" uten noen titordning eller definisjon ti "x".

Def: En variabel som ikke er "i live" på et gitt punkt, sies å være "død"

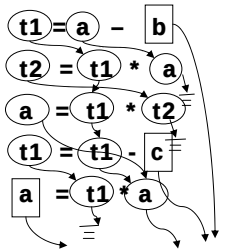
kap 9.5 "Neste bruk" og "i live" innen én basal blokk

- Før man gjør kodegenerering for en basal blokk er det lurt å skaffe seg oversikt over bruk av variable:
  - En variabelforekomst har en "neste-bruk" :
    - Den verdien den har blir brukt senere i den basale blokka.
    - Da kan det være lurt å la den bli værende i et register
  - En variable-forekomst har ingen "neste-bruk", men er "i live":
    - Verdien i variabelen blir verken brukt eller strøket senere i blokken
    - Men den kan bli/blir brukt i senere blokker
    - Vi må passe på å bevare verdien
    - En variabel-forekomst er død

Eksempel på informasjon om "neste bruk" og "i live"

: Angir at den har "neste bruk", og hvor.
Disse er "i live"

: Angir at den er "i live", uten noen "neste bruk"



a er "død" her siden den får ny verdi innen denne blokka senere

: Er "død"

t1 er død, siden den bare brukes innen denne blokka

Altså : I live etter blokka: ***a,b,c***
Dvs., programvariablene.

Kommentarer:
1. Temporære brukes også ofte på tvers av basale blokker (for eksempel flytting)
2. Global "dataflyt-analyse" finner de variable som faktisk er i live etter en basal blokk.

Algoritme for å finne informasjon om "neste bruk" og "i live"

Har en tabell T over alle variable i blokka, der hver variabel kan merkes som:

1. "i live", og har gitt en "neste bruk" i blokka
2. "i live", men uten "neste bruk" i blokka
3. "død" (og dermed ingen "neste bruk")

Initialisering:

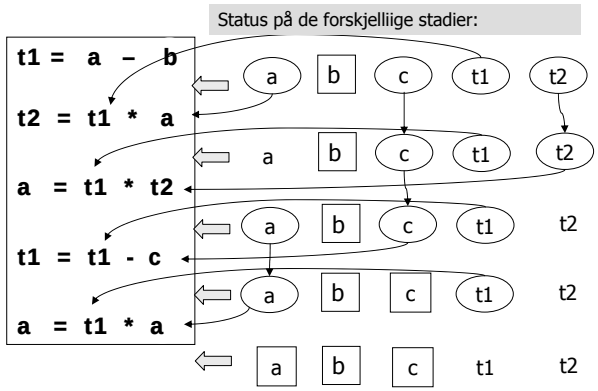
De variablene som er "i live" ved slutten av blokka (oftest: brukervariable) merkes med 2 i T, resten merkes 3

1. Merk x i S slik x er merket i T
2. Forandre x sitt merke i T til 3
3. Merk y og z i S slik de er merket i T
4. Forandre i T merkene for y og z til 1, med "neste bruk" satt til h.h.v y og z i S.

Må skille mellom 1. og 3. for at `a = a + b;` skal bli riktig
(Trykkfeil som er rettet i det utdelte)

Algoritme for "neste bruk" og "i live"

- Gå bakfra og hold greie på "status til alle variable:
  (N.B. Trykkfeil i bokas algoritme – rettet i det utdelte)



Døde variable er tegnet uten ring rundt.

I siste linje antar at vi at bare progr. variable overlever fra en blokk til en annen.

Kodegenererings-algoritme

- Enkel algoritme: Lager maskinkode for én og én Basal Blokk
- Algoritmen holder (innefor hver Basale Blokk) beregnede verdier i registre så langt det er mulig (Spesielt viktig om de har "neste bruk")
- Når programkontrollen går mellom de Basale Blokkene så ligger samtlige variabel-verdier i sine respektive hukommelses-plasser (antar at temporærer variable ikke "er i live")
- Kodegenerering for hver Basal Blokk blir da:
  - Det genereres kode for én og én treadresse-setning av gangen, i tur og orden fra første til siste setning
  - Etter siste setning legges verdier fra registre tilbake til sine respektive hukommelses-plasser
- Noen mangler ved algoritmen
  - Variable som kun blir lest innenfor en Basal Blokk blir ikke lagt i registre, selv om det er mange referanser til variabelen
  - I enkleste utgave sjekker den ikke på komutativitet

Register- og adresse-deskriptorer

- Kodegenerator-algoritmen bruker deskriptorer for å holde greie på hva som er i registre og i adresser for variable:
  - En Register-deskriptor for hvert register holder greie på hva som for tiden er i et register. Ved starten skal alle register-deskriptorer angi at registeret er tomt. Generelt angir register-deskriptoren at registeret inneholder verdien "null" eller flere variable (navn).
  - En Adresse-deskriptor holder greie på hvor verdien av en variabel finnes i øyeblikket. Det kan være i et register, en lokasjon på stakken, en hukommelses-adresse eller i en mengde av slike.
  - Disse opprettes etter hvert som det blir "snakk om" variablene.
    At det ikke er noen adresse-deskriptor for 'x' betyr:
    - x er programvariabel: Verdien ligger (bare) i variabel-lokasjon
    - x er temporær variabel: Variabelen er ikke i bruk
  - Informasjonen er redundant – dvs. vi har begge deskriptor-typene (adresse og register) "bare" for å få raske oppslag. Kunne greid oss med én av dem.

Kodegenerering for: X = Y op Z

1. Finn et register for å holde resultatet:
   1. L = getreg(X = Y op Z)
2. Sjekk adresse-deskriptor for Y:
   - Y' = "beste lokasjon" der Y finnes
   - Hvis Y' ≠ L: **MOV Y', L**
3. Sjekk adresse-deskriptoren for Z:
   Z' = "beste" lokasjon der Z finns
   - **OP Z' , L**
4. Y og/eller Z:
   - har "ingen ny bruk" og
   - er "ikke i live"
   - er i et register

   } AltsÅ. er "død"

   Oppdater i så fall deskriptoren for registrene:
   Registrene inneholder nå ikke Y og/eller Z
5. Oppdaterer adresse-deskriptor for X: X.loc = {L}.
   Hvis L er et register så oppdater også register-deskriptorer for L: L.contain = {X}

Getreg (X = Y op Z)

1. Hvis Y ikke er "i live" etter "X = Y op Z" og Y er alene i R_i:
   - Oppdater (R_i) else
2. hvis det finnes et tomt register R_i : Return (R_i) else
3. hvis X har en "neste bruk" eller X er lik Z eller operatoren ellers krever et register:
   - Velg et (okkupert) register R
   - Hvis verdien ikke ligger i hukommelsen:
     - **MOV R, MEM**
       Oppdater adresse-deskriptor for **MEM**
   - return (R) else
4. Return (X), altså lever hukommelses-plassen til X (må kanskje opprettes om X er en temp-variabel

Opprinnelig verdi av X ødelegges →

NB: For at X = Y + X skal funke, måtte 3 modifiseres, ellers ville vi fått:

~~MOV Y X~~

~~Add X X~~

Eksempel på kode-generering

Setninger	Generert kode	Reg. deskriptorer	Adr. deskriptorer
		Alle Reg er tomme	
t = a - b	MOV a, R0 SUB b, R0	R0 inneholder t	t i R0
u = a – c	MOV a, R1 SUB c, R1	R0 inneholder t R1 inneholder u	t i R0 u i R1
v = t + u	ADD R1, R0	R0 inneholder v R1 inneholder u	v i R0 u i R1
d = v + u	ADD R1, R0 MOV R0, d	R0 inneholder d	t i R0 d i R0 og hukommelsen

Eksamen INF5110
våren 2004

Oppgave 1
Vi ser på følgende sekvens av treadresse-setninger:

(1) a := b - c
(2) b := d - a
(3) t1:= a - b
(4) t2:= a - c
(5) t3:= t1 + t2
(6) d := t3 - t2
(7) if d > 0 goto 10
(8) a := c - d
(9) goto 11
(10) a := c + b
(11) c := a - d
(12) if c < a goto 8
(13) c := a + d
(14) goto 3

Spørsmål 1.a
Del opp sekvensen over i basale blokker. Angi svaret ved bare å liste opp linjenumrene på startlinjen i hver blokk.

Spørsmål 1.b
Gi de basale blokkene nedover i programbiten navnene B1, B2, osv.Tegn så en frittstående utgave av flyt-grafen til programmet, der nodene bare er merket med sitt navn (uten noe kode inni).

Spørsmål 1.c
Angi kort kravene som i vårt pensum settes for at noe skal være en ``løkke" i en flytgraf?

Spørsmål 1.d
Angi de løkkene som finnes i flytgrafen fra 1.b

Spørsmål 1.e
Forklar kort hva det vil si at en variabel er ``i live" (``live")på et angitt sted i programmet.

Spørsmål 1.f
Avgjør følgende, og gi korte forklaringer:
- Er variablen a i live etter linje (4) i programbiten over?