Mer kodegenerering: Tilleggsnotat fra AHU
INF5110 – 6. mai 2008

Stein Krogdahl,
Ifi UiO

Program framover:

Torsdag 8. mai: **Ikke** forelesning (altså **ikke** som tidligere annonsert)

Tirsdag 13. mai: Avsluttende vanlig forelesning

Torsdag 22. mai: Gjennomgåelse av fjorårets eksamen

Avsluttende om kodegenerering

Avsluttende pensum:

En del fra utdelt kap 9 fra Aho, Sethi og Ullmann ′ s kompilatorbok ("Drage-boka**") :** Kap. 9.4, 9.5 og 9.6

Vi innfører en del begreper:

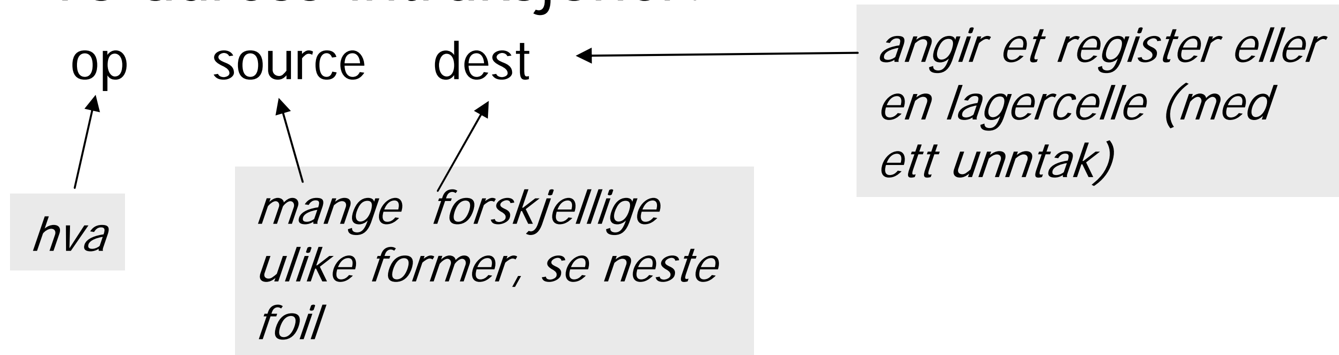
som vi ikke skal bruke i fullt monn. De er imidlertid viktige om man vil gå videre med litt optimalisering, og vi skal forsøke å gi begrepene en intuitiv begrunnelse

Som bakgrunnsmateriale:

kan det være fint også å lese kap. 8.9 i Louden

Maskinen det oversettes til

- To-adress-intruksjoner:



ADD a b

SUB a b Merk: Beregner b – a og legger svaret i b

MUL a b

....

GOTO l

\+ **Betingete hopp**

\+ **Prosedyrekall**

\+ +

Complex

Reduced

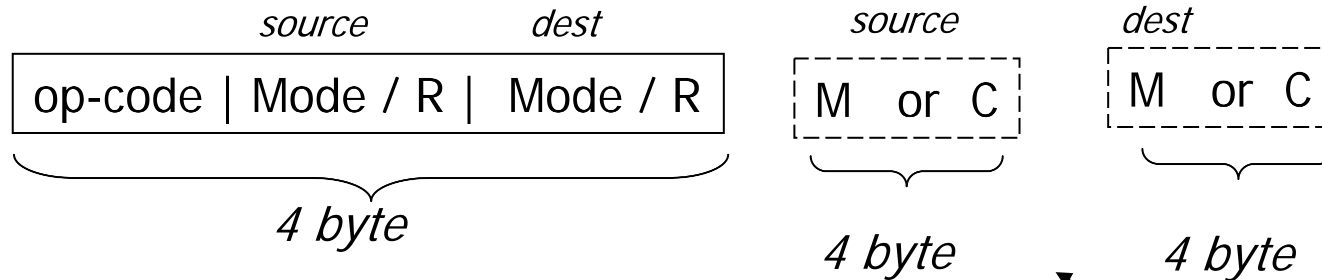
Mer er en CISC-maskin enn en RISC-maskin

Instruksjonsformat og adressemodi – del 1

Vi bruker instruksjonens lengde som "kost" av en instruksjon

Grunnkost for en fire-bytes instruksjon er 1

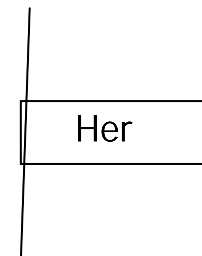
Tilleggskost for hver adressering



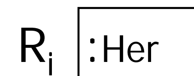
Bare om M eller C

<u>*Adresseringsmodi:*</u>

1 Absolutt: M



0 Register: Ri

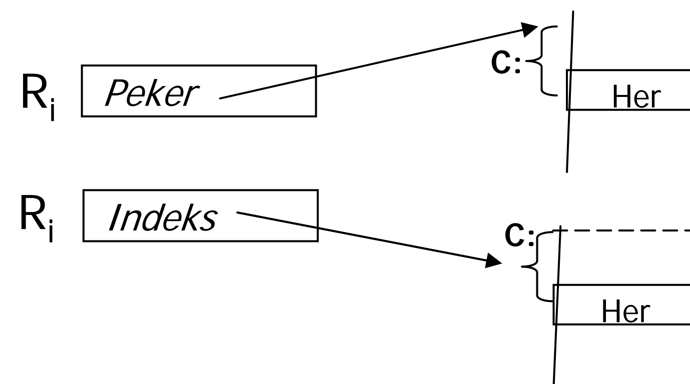


1 Indeksert : C(RI)

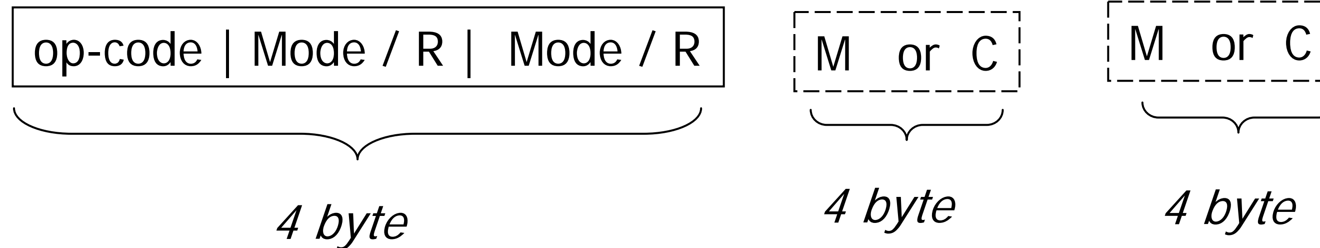
Grei på to måter:

(1) Til å la R inneholde en objekt-peker og la C være en kompilatorkjent relativadresse i objektet

(2) Til å la C være peker til en fastliggende array, og la R ha en indeks inn i denne

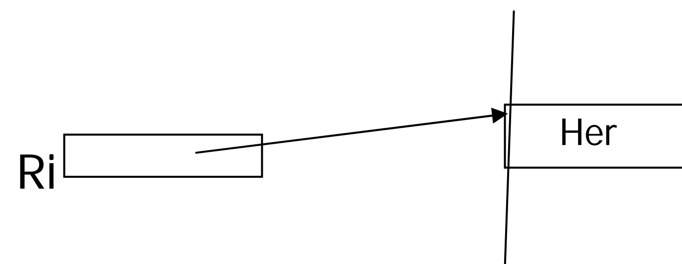


Instruksjonsformat og adressemodi – del 2



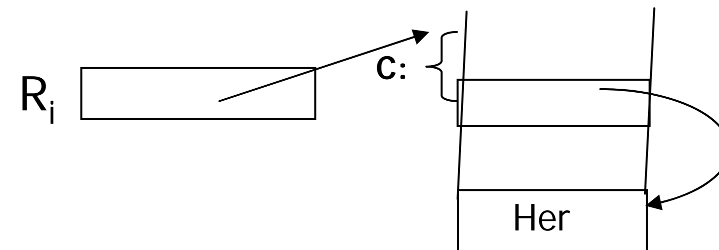
Adresseringsmodi:

0 Indirekte Register *R



1 Indirekte *C(Ri)

Kan brukes til å hoppe to steg av gangen langs en linket liste, f.eks ved følging av "lang" access link



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 (eller en annen type lavnivå kode)

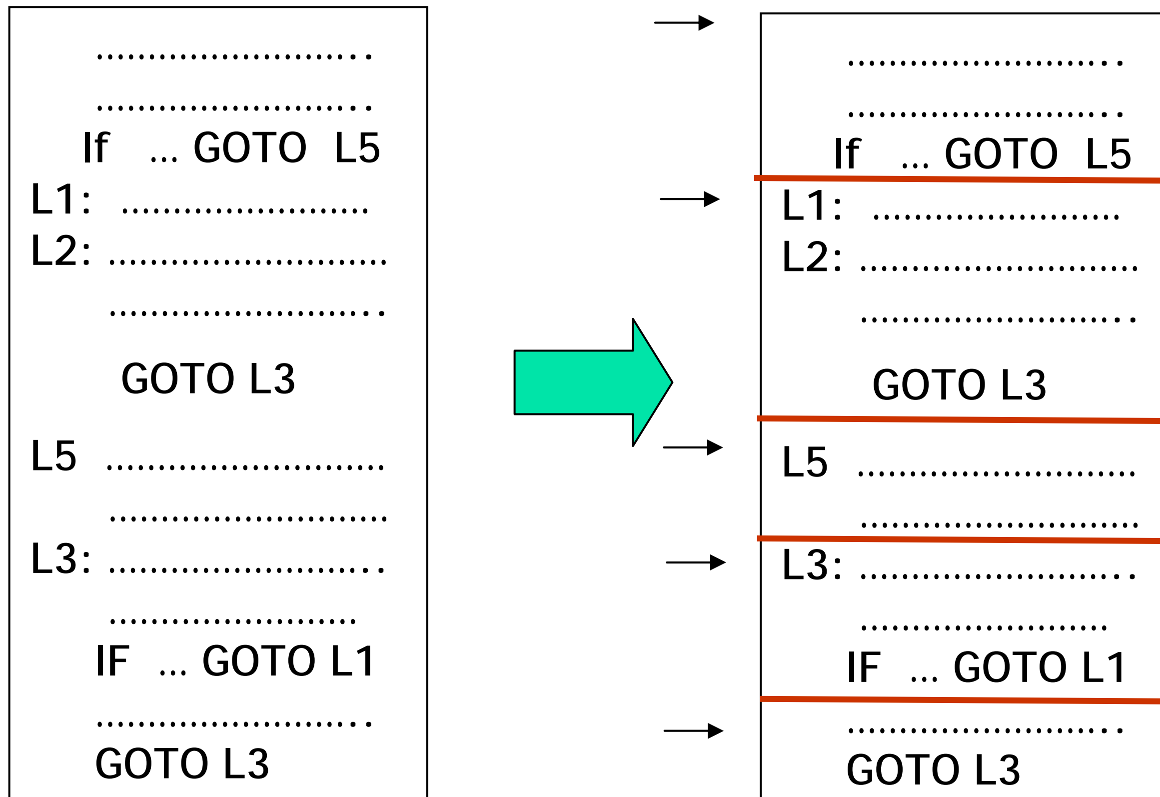
- Nodene i grafen er kalt Basale Blokker
  - En basal blokk representer en sekvens av treadresse
  - Programkontrollen kommer alltid inn i første setning og går sekvensielt gjennom hver setning uten stopp eller sidesprang. Eneste unntak er at siste setning kan være et hopp – også et betinget hopp.

- 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" = "statisk simulerig"

Eksempel på oppdeling i basale blokker

- Basale blokker: Fra og med en "leder" fram til neste, eller slutt
- Algoritme for å finne alle ledere:
  - 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.

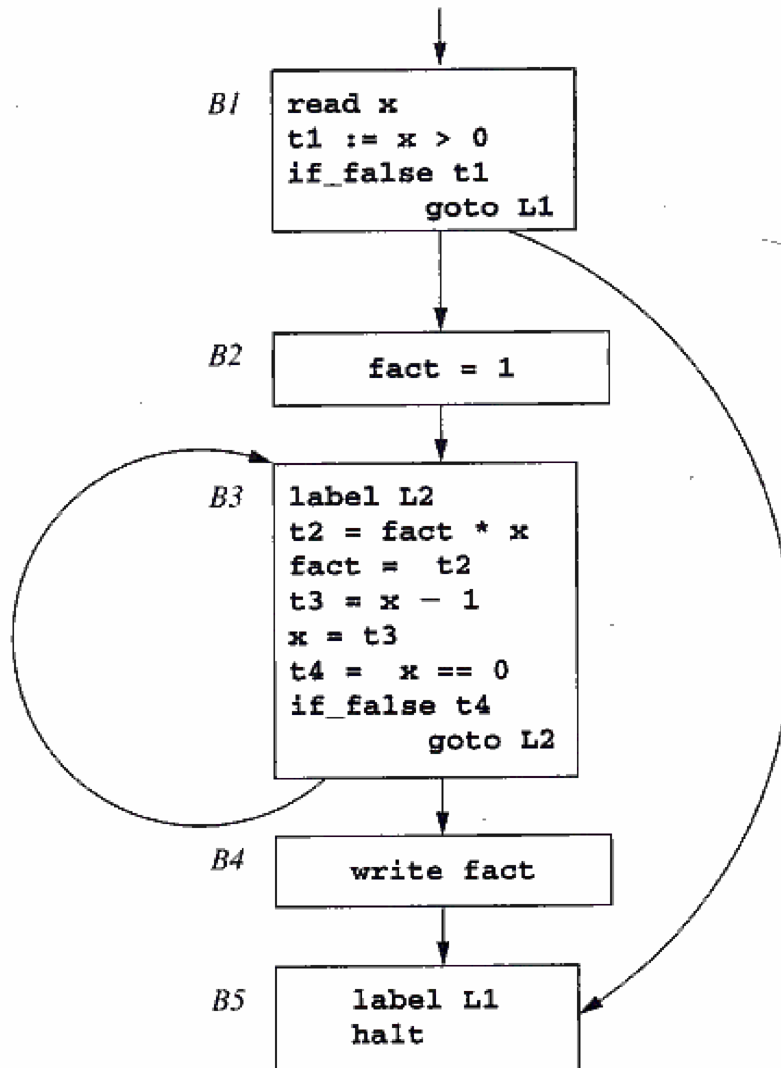
Flyt-graf

- 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



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

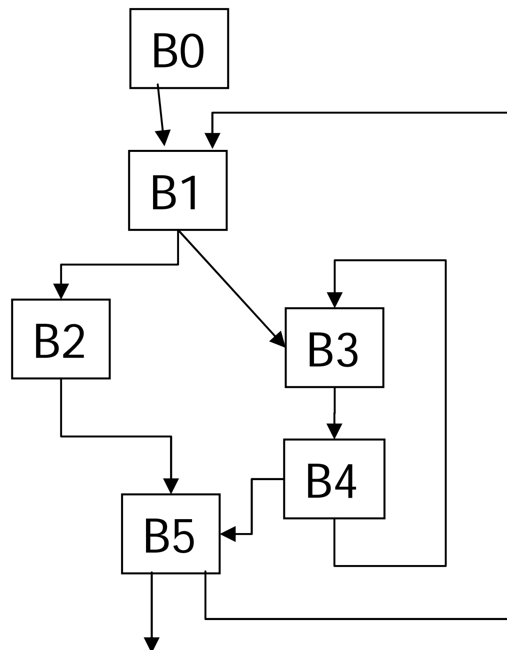
- Metodekall kan plasseres litt forskjellig avh. av grafens bruk
  - Danne egne basale blokker
  - Kan ligge først i en basal blokk?

Kode kan analyseres/arbeides med (f.eks. til optimalisering) på tre nivåer:

1. Inne i én basal blokk
2. Flytgrafen for én metode
3. Alle flytgrafene for hele programmet

Løkker i flyt-grafer

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



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 *(også om Bx og By er samme node!)*

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

Begrunnelse rent praktisk: Ett sted å initialisere løkka & Ett sted om vi skal flytte noe "ut av løkka"

Eksempler:

{B3,B4} og {B1,B2,B3,B4,B5} er løkker

{B1,B2,B5} er <u>ikke</u> løkke (!?)

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

- Begrepet 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 "brukes" x og y

Her "brukes" x og a.

Intuitiv definisjon:

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

Definisjon som kan avgjøre om x er levende

```
x = v+w;
...
a= b + c;
x = u + v        x = w
d = x + y
```

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

Svaret er "ja", fordi det finnes en TA-setning "j" slik at det er minst én eksekveringsvei fra "i" til "j" uten noen tilordning til (eller definisjon av) "x".

Definisjon: En variabel som ikke er "i live" på et gitt punkt, sies å være "død" på dette punktet (og verdien kan da "kastes")

Andre def. som kan være interessant for optimalisering

Kalles global dataflyt-analyse. Eksempler:

- Gitt en TA-instruksjon der x brukes:
  - Finn alle de tilordninger (definisjoner) der denne verdien på x kan være satt
- Gitt en tilordning der x blir satt:
  - Finn alle de steder der denne verdien av x kan bli brukt

Disse og liknende sammenhenger kan "lett" bergenes ved en iterasjons-algoritme på de basale blokkene.

ASU, kap 9.5: Vi generer kode for én og én basal blokk

- Alle foilene til dette kapittelet er rettet litt, og ligger på starten av foilene til neste forelesning (13/5)