Oppgaver til kodegenerering etc.
INF-5110, 16. mai, 2014

Oppgave 1: Vi skal se på koden generert av TA-instruksjonene til høyre i figur 9.10
i det utdelte notatet, side 539

a) Se på detaljene i hvorfor maskinkoden ble slik den ble ut fra algoritmen.

b) Påvis at det finnes en bedre kodesekvens for de samme TA-setningene enn den angitte, som er generert av notatets algoritme.

c) Diskuter hvordan vi kunne forandre denne kodegenererings-algoritmen, slik at den gir bedre kode i dette og andre tilfeller.

Kodegenerering for: **X = Y op Z**

(Med rettelser som også er angitt i notatet)

1. Finn et register for å holde resultatet:
   - L = getreg("X = Y op Z") // Helst et sted Y allerede er
2. Sørg for at verdien av Y faktisk er i L:
   - Hvis Y er i L, oppdater adressediskr. til Y: Y ikke lenger i L **else**
   - Y' := "beste lokasjon" der verdien av Y finnes
   - OG: generer: **MOV Y', L**
3. Sjekk adresse-deskriptoren for Z:

   Z' := "beste" lokasjon der verdien til Z ligger // Helst et register
   - Generer så "hovedinstruksjonen": **OP Z' , L**
4. For hver av Y og Z: Om den er død og er i et register

   Oppdater i så fall register-deskriptoren:

   Registrene inneholder nå ikke lenger Y og/eller Z
5. Oppdaterer deskriptorer i forhold X:
   - X sin adr.deskr. := {L}, og X er ingen andre steder.
6. Hvis L er et register så oppdater register-deskr. for L:
   - L sin reg.deskr. := {X}

Getreg ("X = Y op Z")

Instruksjonen som utfører operasjonen vi få Y som target-adresse

1. Hvis Y ikke er "i live" etter "X = Y op Z", og Y er alene i R_i:
   - Return(R_i)   (punkt 1 kan lett forfines en god del)   **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 i R ikke også ligger "hjemme" i hukommelsen:
     - Generer **MOV R, mem**   // mem er lagerlokasjonen for R-verdien
     - 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)

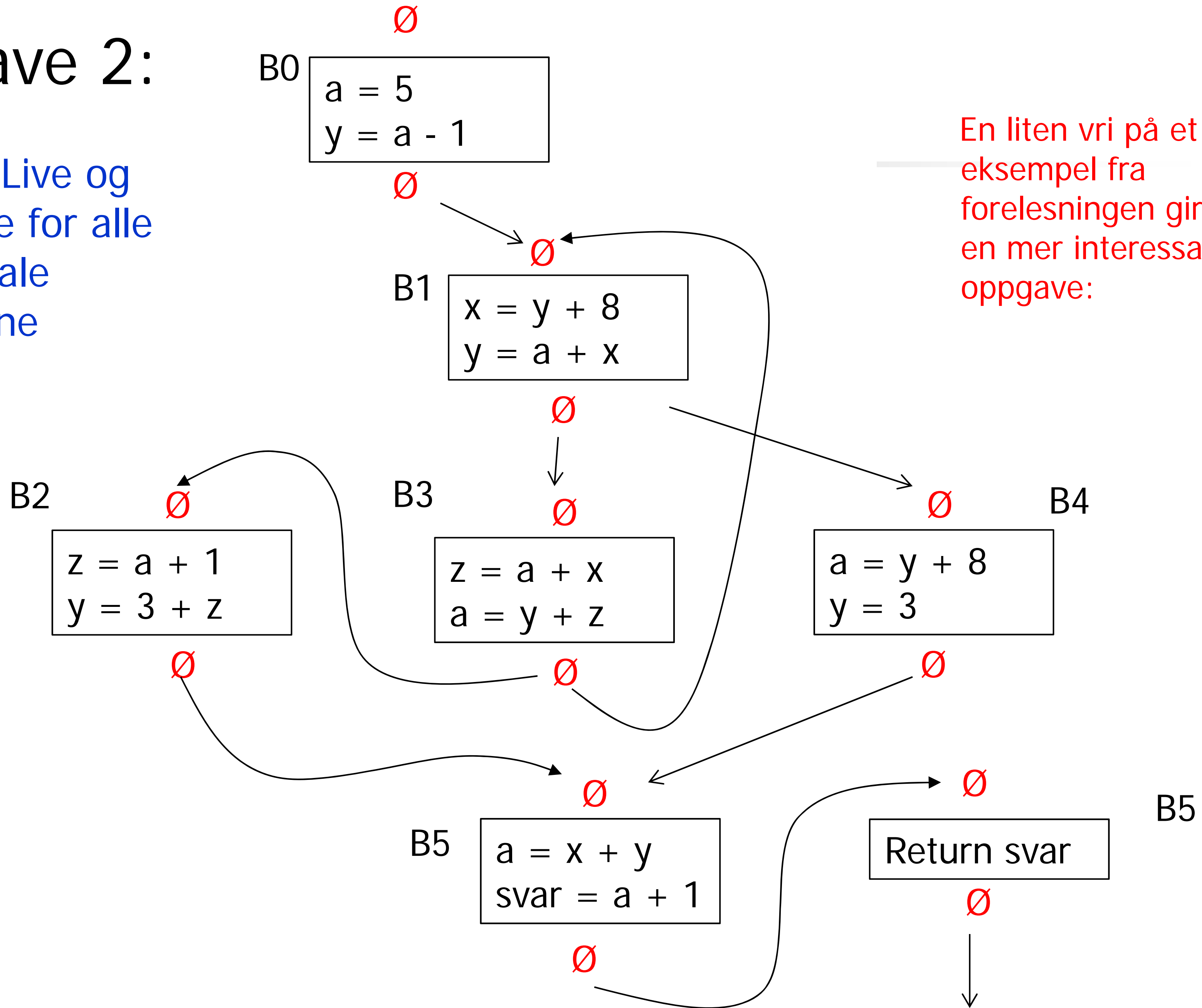
NB: For at X = Y + X skal funke, måtte pnk. 3 modifiseres, ellers ville vi fått:

MOV Y X
ADD X X

Opprinnelig verdi av X ødelegges

Oppgave 2:

Finn InLive og OutLive for alle de basale blokkene



En liten vri på et eksempel fra forelesningen gir en mer interessant oppgave:

Oppgave 3 (Eksamen 2007, opg 4a)

Gitt følgende program, der alle setningene er tre-addresse-instruksjoner, bortsett fra at vi også tillater if- og while-setninger på vanlig måte. Instruksjonene "x = input" og "output x" regnes som vanlige tre-addresse-instruksjoner, med den opplagte betydning. Vi antar at ingen variable er i live ved slutten av programmet.

```
1:   a = input
2:   b = input
3:   d = a + b
4:   c = a * d
5:   if ( b < 5 ) {
6:       while ( b < 0 ) {
7:           a = b + 2
8:           b = b + 1
9:       }
10:      d = 2 * b
11: } else {
12:      d = b * 3
13:      a = d - b
14: }
15: output a
16: output d
```

Angi for hver av variablene *a, b, c* og *d* om de er i live eller ikke *umiddelbart etter* linje 4. Gi en kort forklaring for hver av variablene.

Oppgave 4 (Eksamen 2009, opg 4a, 4b, 4c og 4d)

Gitt følgende sekvens av treadresse-instruksjoner (TA-instruksjoner):

```
1     a  =  input
2     b  =  input
3     t1 =  a + b
4     t2 =  a * 2
5     c  =  t1 + t2
6     if  a < c goto 8
7     t2 =  a + b
8     b  =  25
9     c  =  b + c
10    d  =  a – b
11    if  t2 = 0 goto 17
12    d  =  a + b
13    t1 =  b – c
14    c  =  d – t1
15    if  c < d  goto 3
16    c  =  a + b
17    output  c
18    output  d
```

Oppgave 4 (Eksamen 2009, opg 4a og 4b)

4a Angi (som en sortert liste av tall) ved hvilke linje-numre det starter en ny basal blokk ("basic block").

4b Gi hver basal blokk nedover i programmet navn B1, B2, ... osv, og tegn opp flytgrafen (uten koden, men bare med navnet inni hver node).

Oppgave 4 (Eksamen 2009, opg 4c)

Den som produserer TA-kode påstår at den er slik at temporære variable alltid er døde på slutten av hver basal blokk, og ved starten av programmet, selv om de samme temporærvariable altså blir brukt i flere basale blokker, slik som over.

Formuler en generell regel som skal brukes lokalt på alle basale blokker for å sjekke om det produsenten av TA-kode sier er riktig, for et gitt TA-program. Vi antar her at alle variable er døde når programmet stopper.

Oppgave 4 (Eksamen 2009, opg 4d)

Bruk reglen du fant under **4c** til å undersøke hvordan dette forholder seg i den TA-koden som er angitt over. De temporære variablene heter t1, t2.

Oppgave 5 (Eksamen 2010, opg 4d)

Arne har sett på kodegenerings-algoritemen på slutten av det utdelte heftet (fra kap. 9 i ASU). Han mener da at for de to treadresse-instruksjonene:

```
t1 = a – b
t2 = b – c
```

så vil algoritmen produsere instruksjonene under. Han har antatt at det er to registre, og at begge er tomme ved starten

```
MOV a, R0
MOV b, R1
SUB  R1, R0
SUB  c, R1
```

Ellen er uenig. Hvem har rett? Forklar.

Oppgave 6 (Eksamen 2011, opg 4c)

Vi vil oversette vår P-kode til maskinkode for en maskin der alle operasjoner (inkl. sammenlikninger) må gjøres mellom verdier som ligger i registre, og der kopiering mellom lageret og registre bare kan gjøres med egne LOAD- og STORE-instruksjoner. Under oversettelsen har vi en stakk med diskriptorer.

Vi skal se på det å oversette P-instruksjonen ″ldv v″.

Spørsmålet er om det da er fornuftigst å produsere en LOAD-instruksjon som henter verdien av variabelen ″v″ opp i et register, eller om det er best bare å legge en diskriptor på stakken som sier at denne verdien ligger i variabelen ″v″.

Drøft dette ut fra forskjellige forutsetninger, f.eks. ut fra hva språket som vi oversetter fra lover om rekkefølgen ved beregning av uttrykk (men også ut fra andre ting som du mener er aktuelle).

Oppgave 6 (Eksamen 2011, opg 4d)

Vi vil igjen oversette vår P-kode til maskinkode, slik som i oppgave **4c**, og vi skal anta at vi skal oversette én og én basal blokk, og at alle registre skal tømmes på kontrollert måte etter utførelsen av en basal blokk.

Spørsmålet her er hvilke data diskriptorene på stakken skal inneholde, og hva du eventuelt trenger av andre typer diskriptorer. Vi antar at vi kan tillate oss å lete gjennom alle kompilator-stakkens diskriptorer hver gang vi lurer på hvor visse verdier er etc., slik at informasjon vi kan finne på denne måten ikke behøver å lagres i ekstra diskriptorer.

Forklar også kort hvordan du kan finne den informasjonen du trenger under kodegenereringen, og hvordan du eventuelt vil bruke de ekstra diskriptorene du vil ha.