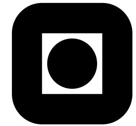
Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap



TDT4258 – Eksamen vår 2013

Løsningsforslag

Oppgave 1 Flervalgsoppgave (16 poeng)

Du får 2 poeng for hvert riktig svar og 0 poeng hvis svaret mangler. Hvis svaret er feil gis -0.5 poeng. Kun ett alternativ er riktig.

a) Hva er riktig om prosessorsamlebånd?

1. Throughput (antall instruksjoner utført per tidsenhet) økes
2. Latency (tid det tar å utføre en enkelt instruksjon) minskes
3. Samlebånd forutsetter cache
4. Samlebånd forutsetter virtuelt minne

Riktig svar: Alternativ 1

b) Hva brukes en interruptkontroller til?

1. Kontrollere at CPU håndterer interrupt innen deadline
2. Minke responstiden til interrupthandleren
3. Samle interruptlinjer fra flere I/O-enheter slik at mange enheter får mulighet til å gi CPU et interrupt
4. Implementere synkroniseringsmekanismer for interrupthandlere

Riktig svar: Alternativ 3

c) Hvilken av følgende årsaker kan *ikke* føre til at en CPU får interrupt?

1. En bruker trykker på en knapp
2. En ALU-operasjon fører til overflyt
3. En I/O-enhet trenger mer data fra prosessoren
4. En DMA-enhet er ferdig med å overføre data

Riktig svar: Alternativ 2

d) Hva er hensikten med symboltabellen i en objektfil?

1. Nødvendig for at kode i én objektfil skal kunne referere til andre objektfiler
2. Gjøre det mulig å ha private variabler
3. Gjøre disassemblering mer oversiktlig
4. Dokumentasjonsgenerering

Riktig svar: Alternativ 1

e) Hva er den vanligste måten å få til preemptiv multitasking på?

1. Hver prosess gir eksplisitt fra seg kontrollen ved OS-kallet yield()
2. En egen HW enhet tar seg av prosesschedulering
3. Timerinterrupt som kjører prosesscheduler jevnlig
4. En egen systemservice i user space tar seg av schedulering

Riktig svar: Alternativ 3

f) Hva er *ikke* riktig å si om drivere i Linux?

1. En bug i driveren kan ødelegge for hele systemet
2. En driver lages ofte som en kjernemodul
3. Grensesnittet til en driver er ofte gjennom device-filer
4. En driver har tilgang til alle vanlige C-bibliotek

Riktig svar: Alternativ 4

g) Hva er *ikke* riktig om prosesser og tråder?

1. En prosess kan kommunisere med andre prosesser
2. En prosess har ofte sitt eget virtuelle adresserom
3. En tråd deler adresserom med andre tråder i samme prosess
4. En tråd deler som oftest adresserom med alle tråder i alle prosesser

Riktig svar: Alternativ 4

h) Hva er *ikke* en vanlig OS-teknikk for å redusere energiforbruk til en datamaskin?

1. Justere spenning og klokkefrekvens avhengig av arbeidsbelastning
2. Endre sidetabeller for mer effektiv bruk av TLB
3. Skru av og på individuelle I/O-enheter dynamisk ut i fra bruksmønster
4. Sette CPU i sleep-mode når OS har vært idle en stund

Riktig svar: Alternativ 2

Oppgave 2 Buss (12 poeng)

a) Du skal spesifisere en bussprotokoll for kommunikasjon mellom en CPU og to I/O-kontrollere som ligger på adressene 0x100 og 0x200.

Dette skal være en asynkron buss, hvor følgende signaler skal brukes:

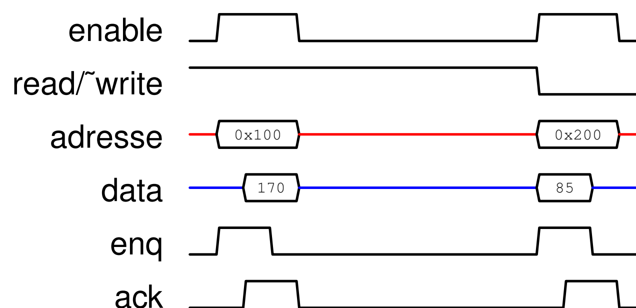
- enable (1 bit)
- read/˜write (1 bit)
- adresse (16 (bit)
- data (8 bit)
- enq (1 bit)
- ack (1 bit)

Tegn et timingdiagram hvor du viser hvordan du gjør følgende operasjoner:

1. Lese en byte fra adresse 0x100 (hvor tallet 170 ligger)
2. Skrive tallet 85 til adresse 0x200

Dersom du har gjort antagelser som ikke går klart fram i diagrammet må disse forklares med tekst.

Løsning: Det finnes flere måter å løse denne oppgaven på. Et eksempel vises i figur 1. Vi benytter *enq* og *ack* til handshake. *enable* angir om I/O-kontrollerne må lytte på bussen eller ikke, men er strengt tatt ikke nødvendig i dette eksempelet. *read/˜write* angir retningen på dataoperasjonen, hvor høyt signal angir leseoperasjon. *data* er bidireksjonell og kan drives av alle enhetene på bussen. Ved en leseoperasjon vil CPU lese databussen et sted mellom *ack* går høy og *enq* går lav. Ved en skriveoperasjon vil I/O-kontrolleren lese databussen et sted mellom *enq* går høy og *ack* går høy.



Figur 1: Timingdiagram

b) Hva ligger i begrepet "burst transfer" for en synkron buss?

Løsning: På en synkron buss uten burst så må master sette opp en adresse før hvert ord som skal overføres. Dette kan medføre dårlig bussutnyttelse dersom en sekvens av ord skal overføres. En teknikk for å redusere overhead er burst, hvor kun en startadresse settes opp. Deretter kan ett og ett ord overføres per sykel helt til hele sekvensen er overført.

Oppgave 3 Kompilator og programmering (14 poeng)

Gitt følgende C-kode.

```
x = a + b
x = x + c
y = x + c
z = a - b
```

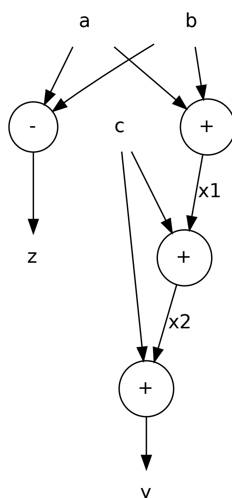
a) Omgjør C-koden til "single assignment form" og tegn tilhørende dataflytgraf (DFG). Navngi alle kantene i grafen med variabelnavn.

Løsning: Single assignment form:

```
x1 = a + b
x2 = x1 + c
y = x2 + c
z = a - b
```

Dataflytgrafen er vist i figur 2.

b) Gjør en levetidsanalyse for variablene.



Figur 2: Dataflytgraf

Bruk graph-coloring til å finne antall registre som er nødvendig for å implementere dette i assembly.

Løsning: En livstidsgraf er vist i figur 3. Denne viser når de forskjellige variablene er i bruk. En vertikal strek angir hver av de fire kodelinjene.

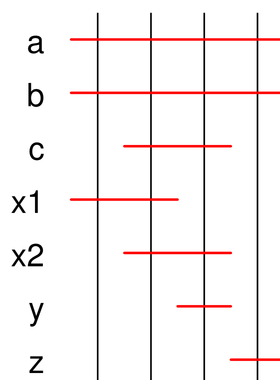
Figur 4 viser en konfliktgraf hvor hver kant i grafen angir at variablene er i bruk samtidig. Graph coloring er brukt til å finne antall nødvendige registre. Her trengs fem forskjellige farger, og dermed **fem forskjellige registre**.

c) Hva kan gjøres med programmet i punkt b som minsker antall nødvendige registre uten å endre sluttresultat?

Løsning: Vi kan bruke teknikken "operator scheduling", dvs endre rekkefølgen på kodelinjene, slik at antall nødvendige registre reduseres. Her er programmet etter endring:

```
z = a - b
x1 = a + b
x2 = x1 + c
y = x2 + c
```

Etter denne endringen vil en vi få livstidsgrafen i figur 5 og konfliktgrafen i figur 6, som viser at vi trenger tre registre.



Figur 3: Livstidsgraf

d) Forklar hvordan en C-kompilator kan implementere funksjonskall som støtter rekursjon, argumenter, returverdi og lokale variabler.

Løsning: Funksjonskall implementeres ved å benytte CPU-ens mekanisme for kall av subrutiner. Dette gjøres ved hjelp av en *stakk*.

Hvert funksjonskall fører til at det blir opprettet en *stack frame*, som består av data lokale for akkurat dette kallet av subrutina. Siden hvert kall oppretter en ny stakkframe kan vi støtte *rekursjon*.

Egne instruksjoner brukes for hopp til subrutinen som tar seg av å lagre programteller, enten rett på stakken eller i et eget *link register*.

For å unngå å ødelegge for kallende kode så må registre som brukes av funksjonen *mellomlagres på stakken* slik at de kan gjenopprettes når funksjonen returnerer.

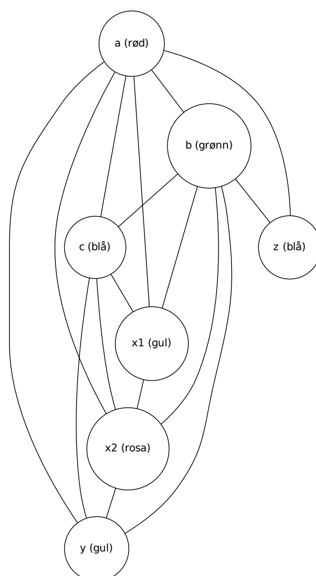
Argumenter kan håndteres enten ved å pushe de på stakken, eller hvis det er nok registre tilgjengelig, legge de i registre. Tilsvarende kan returverdien legges i et register som kallende kode kan sjekke.

Lokale variable kan reserveres på stakken. Et register, *frame pointer*, settes til adressen til framen og lokale variabler og argumenter kan refereres relativt til frame pointeren.

Oppgave 4 Operativsystem (8 poeng)

Forklar begrepet “priority inversion” og beskriv hvordan dette kan håndteres av scheduleren.

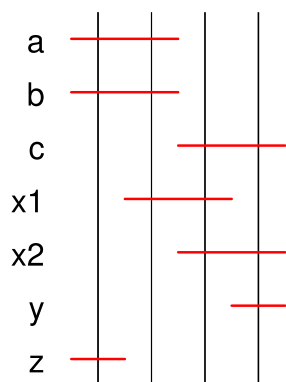
Løsning: Dersom en situasjon kan oppstå hvor en lavere prioritets prosess kan hindre enn høyere prioritetsprosess fra å kjøre så kalles dette *priority inversion*. Dette kan f.eks oppstå når man har minst tre prosesser med forskjellige prioriteter og en delt ressurs. Hvis P1 har lavest pri og holder



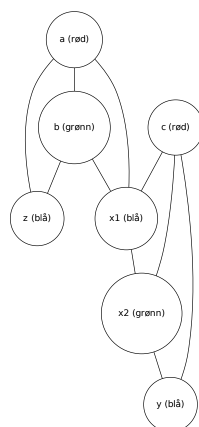
Figur 4: Konfliktgraf

ressursen, kan ikke P3 med høyest pri kjøre hvis den ønsker samme ressurs. Dersom P2 ikke trenger tilgang til ressursen, kan den preempte P1, og vi har priority inversion hvor P2 kan kjøre så lenge den ønsker selv om P3 har høyere pri.

Dette kan løses ved å gi prosesser midlertidig høyere prioritet mens de holder en delt ressurs, slik at disse ikke kan preemptes så lenge ressursen er i bruk.



Figur 5: Livstidsgraf etter endring



Figur 6: Konfliktgraf etter endring