NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

Løsningsforslag til eksamen i SIF8025 Datamaskiner og operativsystemer og SIF8025 Datamaskinarkitektur og operativsystemer Onsdag 27. november 2002 Kl. 0900 - 1400

Oppgave 1 – Flervalgsspørsmål ("multiple choice") – 15 %

Dersom du finner flere alternativer som synes å passe, setter du kryss for det ene som passer best. For å unngå at gode tippere blir belønnet, vil et galt svar gi færre poeng enn om oppgaven forblir ubesvart.

Alternativ→ Oppgave↓	**1**	**2**	**3**	**4**	**5**
a)					X
b)				X	
c)		X			
d)			X		
e)				X	
f)	X				
g)				X	
h)		X			
i)	X				

j)				X	
k)		X	X		
l)		X			
m)	X				
n)				X	
o)					X

NB! På grunn av en feil i oppgaveteksten til k) (alternativ 2 og 3 var like), må både alternativ 2 og 3 godkjennes.

Oppgave 2 – Busser – 7 %

a) Det er vanlig å skille mellom serielle og parallelle busser. Dette gjelder både interne og eksterne busser.

1. Gi eksempler på to serielle og to parallelle busser.

   De som er omtalt i pensum er:
   Serielle: USB, FireWire, InfiniBand, SerialATA, FibreChannel
   Parallell: PCI, SCSI (Systembuss er også akseptabelt)

2. Hvilke fordeler har serielle busser sammenlignet med parallelle busser.

   ** Smalere kabler, mindre og enklere kontakter.*
   ** Enklere synkronisering av signaler når man har færre ledere.*
   ** Mindre risiko for interferens mellom de elektriske lederne som bussen består av.*

b) Hva betyr det at PCI bruker skjult og sentralisert arbitrering?

Skjult: Samtidig som en busstransaksjon pågår, arbitreres neste busstransaksjon. Dette gjør at det trengs ekstra ledere for arbitrering.
Sentralisert: En enhet på bussen bestemmer hvem som skal få bruke bussen. Alle som er interessert, må varsle denne enheten.

Oppgave 3 – Lager – 9 %

a) I datamaskiner er lageret organisert i et hierarki.

1. Tegn en enkel skisse over lagerhierarkiet i en typisk datamaskin.

   Se figur 4.1 i læreboka – side 100.

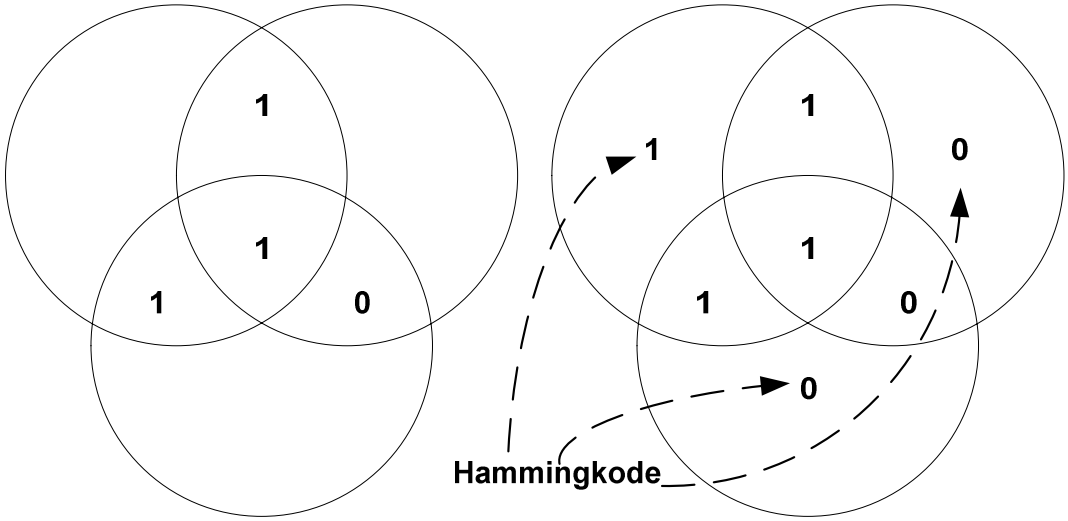
Det viktigste: Registre -> Hurtigbuffer -> Hovedlager -> Sekundærlager

2. Hva oppnår man med å organisere lagret slik?

   Enkelt sagt: Ytelse for en billig penge.
   Øverst i hierarkiet finnes raske, små, dyre og ikke-persistente teknologier, mens nederst finnes trege, store, billige og persistente teknologier. På grunn av lokalitetsprinsippet vil mesteparten av aksessene skje i de hurtige lagrene slik at ytelsen vil være bedre enn gjennomsnittet til de forskjellige lagerteknologiene.

b) Figur 1 viser hvordan man kan lage en enkeltfeil-korrigerende kode bestående av tre bit til bitstrengen 1110. Forklar hvordan man kan utvide denne koden til også å oppdage dobbeltfeil.

Ved å inkludere en ekstra bit som f.eks. er paritet for alle de syv eksisterende bit'ene.
Se side 153 i læreboka.



Figur 1: Konstruksjon av Hammingkode for enkeltfeil-korrigering

c) Tenk deg at du skal designe et system for masselagring av data basert på harddisker.

1. Hva kan du gjøre for å oppnå høy MTTF (Mean-time-to-failure)?

   Bruk av disker i parallell - RAID.
   I tillegg bør man ta i bruk f.eks. replisering av diskkontrollere, strømforsyninger etc.

2. Hva kan du gjøre for å oppnå lav MTTR (Mean-time-to-repair)?

   Her finnes det ingen fasitsvar.
   Et mulig svar kan være: MTTR er definert som den gjennomsnittlige tiden det tar fra en feil blir detektert til den er blitt reparert. Hvordan en feil kan repareres hurtig, vil variere etter hvilken type feil det gjelder. Her kan det være snakk om å ha f.eks. reservedeler tilgjengelig (kabler, disker, strømforsyninger osv.), sikkerhetskopi av data på egnet medium (f.eks. bånd), innarbeidede rutiner og å ha teknisk personell tilgjengelig.

Oppgave 4 – Samlebånd og superskalaritet – 12 %

a) Et *n*-stegs samlebånd kan teoretisk gi en ytelse som er *n* ganger høyere enn uten samlebånd. Gi en kort oversikt over årsaker til at denne forbedringen ikke oppnås i praksis.

Rangert mhp. viktighet:
** Forgreninger kan gjøre at gal instruksjon blir hentet. Dermed må samlebåndet tømmes og de riktige instruksjonene hentes inn ("branch penalty"). Prosedyral avhengighet er det samme.*
** Ressurskonflikter (f.eks. samtidig lageraksess)*
** Sanne dataavhengigheter (ut- og anti-avhengigheter kun problem ved superskalaritet)*
** Ulik steglengde – samlebåndet må designes med tanke på det tregeste steget*
** "Latch delay" mellom hvert steg i samlebåndet.*
(Ekstra kostnad ved oppstart av samlebånd – betyr lite i den store sammenheng)*

b) To teknikker for dynamisk forgreningspredikering er bruk av historebits og historietabell. Forklar den viktigste fordelen historietabell har i forhold til historebits.

Ved bruk av historebits, lagres adressen til forgreningsinstruksjonen + historieinformasjon. Ved bruk av historietabell, lagres dessuten måladressen til forgreningsinstruksjonen. Denne informasjonen er ikke tilgjengelig før etter "fetch operands"-steget dersom man kun bruker historebits. Dermed er det enklere å hente inn neste instruksjon når man predikerer hopp dersom man bruker historietabell.

c) Gitt følgende kodesnutt:

I1: R1 ← R2 + R3
I2: R2 ← R1 + 5
I3: R2 ← R2 – 10
I4: R4 ← R1 + R5

1. Identifiser eventuelle sanne dataavhengigheter.

   ** I2 leser R1 som I1 har skrevet.*
   ** I3 leser R2 som I2 har skrevet.*
   ** I4 leser R1 som I1 har skrevet.*

2. Lag en revidert kodesnutt som bruker registeromdøping til å fjerne eventuelle ut- og anti-avhengigheter.

   ** Utavhengighet: I2 -> I3 (pga. R2)*
   ** Antiavhengigheter: I1-> I2 (pga. R2), I1 -> I3 (pga. R2)*

   Revidert kode:
   R1 ← R2 + R3
   R2a ← R1 + 5
   R2b ← R2a – 10
   R4 ← R1 + R5

   Pga. de sanne dataavhengighetene oppnår man ikke spesielt mye med denne registeromdøpingen. Omdøpingen vil nok bli gjennomført likevel siden dette er enklere

enn å detektere når omdøpinger ikke er nødvendig.

d) Hvorfor er fast instruksjonsformat en fordel for superskalare prosessorer?

Dersom man har variabel lengde på instruksjonsformatet, blir prefetch av instruksjoner vanskeligere ettersom de må (delvis) dekodes før man kan si lengden på dem. Hvis man i en superskalar prosessor ønsker å f.eks. hente de fire neste instruksjonene samtidig, er det med variabel lengde på instruksjonsformatet umulig å vite hvor mange byte man må hente.

Oppgave 5 – IA-64 – 7 %

Gitt følgende kodesnutt:

```
st8 [r4] = r12          // Lagre 8 bit fra register r12 til minneadressen gitt av r4
ld8 r6 = [r8]           // Hent 8 bit fra minneadressen gitt av r8 og legg disse i r6
add r5 = r6, r7         // Adder innholdet i r6 og r7 og legg resultatet i r5.
```

a) Hvorfor er det ønskelig å flytte ld8-instruksjonen før st8?

Instruksjoner som medfører lageraksess tar gjerne lang tid i forhold til register-til-register instruksjoner. I denne kodesnutten, skal resultatet av ld8-instruksjonen brukes av neste instruksjon som derfor sannsynligvis må vente. Hvis man derimot bruker ”delayed load” eller ”spekulativ lasting”, kan man flytte ld8 tidligere i programmet slik at andre instruksjoner kan utføre mens man venter på at ld8 skal fullføre. Dermed slipper man en ”venteperiode” er prosessoren ikke utfører noe fornuftig.

b) En IA-64 kompilator har endret kodesnutten til:

```
ld8.a r6 = [r8]
st8 [r4] = r12
ld8.c r6 = [r8]
add r5 = r6, r7
```

Forklar hva som vil skje når prosessoren utfører denne siste kodesnutten.

Se ”Data speculation” på side 558 i læreboka.
L8d vil lese fra minne og legge minneadressen inn i ALAT (Advanced Load Address Table).
St8 vil skrive til minne og fjerne eventuelle innslag i ALAT med samme minneadresse.
Ld8.c vil sjekke om minneadressen fremdeles ligger i ALAT. Hvis den gjør det, kan den bruke verdien lest av ld8. Hvis ikke, har st8 skrevet til denne adressen og leseoperasjonen må derfor utføres på nytt. Den siste addisjonen er bare med for å skape motivasjon til å foreta ld8 tidligere enn normalt.