NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

Løsningsforslag til eksamen i TDT4155 Datamaskiner og operativsystemer Mandag 15. desember 2003 Kl. 0900 - 1300

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			

l)			X		
m)	X				
n)		X			
o)					X

Oppgave 2 – Busser – 10 % (2,5 % pr. deloppgave)

a) Hva er fordelene og ulempene ved synkrone busser i forhold til asynkrone busser?

Fordeler: Enklere implementasjon – ingen protokoll nødvendig. Dermed billigere og potensielt raskere.

Ulemper: Hastighet er begrenset av den tregeste enheten på bussen siden alle går etter samme klokke. Rask klokke krever kort buss for at alle enheter skal ha den samme oppfattelsen av hvor man er i klokkesyklen (transmisjonsforsinkelse).

b) Bør lange busser være serielle eller parallelle? Forklar hvorfor.

De kan med fordel være serielle. Over lange avstander får man problemer med synkronisering av de forskjellige linjene som en parallell buss består av. Problemet med interferens mellom ledere blir også større jo lengre bussen er. Ikke like viktig, men parallelle kabler er dyrere per meter pga. flere ledere.

c) Hvilken fordel oppnår man ved å bruke skjult arbitrering?

Samtidig mens en busstransaksjon pågår, kan den neste transaksjonen arbitreres. Dermed slipper man å bruke ekstra tid på arbitrering. (Dette krever egne ledere for arbitreringsformål)

d) Gi en sammenligning av egenskapene og bruksområdene til PCI og SCSI-bussen.

Felles egenskaper: Parallell buss, tidsmultipleksing av adresse- & data-linjer.
PCI: Skjult, sentralisert arbitrering; SCSI: Distribuert arbitrering.
PCI: Intern buss (lydkort, nettverkskort o.l.); SCSI: Ekstern buss (harddisk, CD-rom o.l.).
PCI: 32-64 bits bred; SCSI: 8-16 bits bred.
PCI: Alltid synkron; SCSI: Både synkron og asynkron (asynkron som standard)

Oppgave 3 – Lager – 10 % (2,5 % på a og b, 5 % på c)

a) Vil innføring av hurtigbuffer på minnemodulene i hovedlageret kunne erstatte vanlig hurtigbuffer på prosessoren? Forklar hvorfor / hvorfor ikke.

Det som gjør at hovedlageraksess er tregt, er ikke bare hastigheten på minnebrikkene men også overføringstiden over systembussen. Overføringstiden vil ikke kunne bli forbedret ved introduksjon av hurtigbuffer på minnebrikkene. En slik teknikk vil derfor kun være et

supplement til vanlig hurtigbuffer (som ikke har problemet med lang overføringstid).

b) Lagerteknologier kan grupperes etter hvilken måte data aksesseres på. Forklar kort følgende aksessmåter og gi et eksempel på en lagerteknologi for hver av dem: Sequential access, direct access, random access.

Sequential access: Data er ordnet lineært. All aksess medfører sekvensielt søk. Eksempel: Tape
Direct access: Adresse gir generelt område der data ligger lagret. Aksess medfører direkte aksess til dette området etterfulgt av sekvensielt søk for å finne endelig lokasjon. Eksempel: Harddisk.
Random access: Adresse gir nøyaktig lokasjon. Aksesstid er dermed uavhengig av hva som ble aksessert sist. Eksempel: Halvlederlager (RAM).

c) Gitt en datamaskin med følgende spesifikasjoner:

- Prosessoren utfører gjennomsnittlig 2 instruksjoner pr. klokkesyklus
- Gjennomsnittlig hver åttende instruksjon aksesserer lager.
- Hurtigbufferen har 90% treffrate og aksesstid på 5 ns.
- Hver hurtigbufferlinje som hentes inn fra hovedlager er på 32 byte.
- Bussen mellom prosessor og hovedlager er på 200 MHz og er 32 bit bred.

Hvor hurtig kan prosessoren være (i MHz) før bussen (i gjennomsnitt) ikke greier å levere data hurtig nok?

Setter antall MHz = X.
*Lageraksess pr. sekund = X * 10^6 * 2 / 8 = X * 10^6 / 4.*
*Hovedlageraksess pr. sekund = X * 10^6 * 0,1 / 4 = X * 10^6 / 40.*
*Hovedlageroverføring (bytes/sek.)= X * 10^6 * 32 / 40 = X * 10^6 * 4 / 5*
*Maksimal overføring (bytes/sek)= 200 * 10^6 * 4 = 800 * 10^6.*
*Setter uttrykkene lik hverandre => 800 * 10^6 = X * 10^6 * 4 / 5*
*Forkorter og forenkler => X = 800 * 5 / 4 = 1000.*
Svar: 1000 MHz (= 1 GHz).

Dette svaret ser bort ifra henting av instruksjoner fra lager. Hvis man vil ha det med, må man gjøre antagelser utover det som er gitt i oppgaven.

Oppgave 4 – Superskalaritet – 15 % (5% pr. deloppgave)

a) Denne oppgaven dreier seg om anti-avhengigheter i superskalare samlebånd.

i. Forklar hva antiavhengigheter er og hvordan de kan oppstå.

Gitt følgende eksempelkode:
I1: R1 <- R2 + R3
I2: R2 <- R4 + R5

Hvis I2 får skrive til R2 før I1 får lest fra R2, vil I1 lese gal verdi for R2. Denne avhengigheten mellom I1 og I2 kalles antiavhengighet – en instruksjon må få lese en verdi før den blir overskrevet av en senere instruksjon.

Dette kan kun oppstå dersom man bruker ut-av-rekkefølge tildeling i et superskalart samlebånd. Hvis ikke dette er tilfellet, vil I1 alltid få lese registre før senere instruksjoner skriver til dem.

ii. Forklar hva som kan gjøres for å unngå antiavhengigheter.

Antiavhengigheter kan unngås ved at prosessoren bruker registeromdøping. Da ville koden over kunne blitt:
I1: R1b <- R2a + R3a
I2: R2b <- R4a + R5a

Vi ser nå at I2 skriver til et annet register enn I1 leser fra. Antiavhengigheten har dermed forsvunnet og instruksjonene kan utføres uavhengig av hverandre.

b) Denne oppgaven dreier seg om forgreningspredikering og Pentium 4.

i. Hvorfor er det spesielt viktig med god forgreningspredikering for Pentium 4?

P4 har et spesielt langt samlebånd – hele 20 steg. Hvis man gjetter feil i forgreningspredikeringen, er det dermed svært mye arbeid som går tapt og svært mange steg som må tømmes.

ii. Forklar i korte trekk hvordan Pentium 4 utfører forgreningspredikering.

P4 henter inn kode med x86 CISC-instruksjoner, men oversetter disse til RISC-lignende mikrooperasjoner som er bedre egnet til superskalare samlebånd. På grunn av dette, må P4 utføre forgreningspredikering både for CISC-instruksjonene og mikrooperasjonene.

CISC-instruksjoner: Statisk predikering – historietabell (BTB-branch target buffer) med 4096 linjer.

Mikrooperasjoner: Dynamisk predikering – historietabell med 512 linjer og 4 bits historie. Første utføring: Statisk predikering basert på instruksjonstype og adresse.

c) Gitt en RISC-prosessor med et superskalart samlebånd som har to dekodingsenheter, to heltallsenheter, en load/store-enhet, og to tilbakeskrivingsenheter. Alle heltall- og store-operasjoner tar en klokkesyklus, mens load-operasjoner tar to. Gå ut fra at prosessoren bruker registeromdøping, ut-av-rekkefølge tildeling og ut-av-rekkefølge fullføring.

Som i læreboka, kan du anta at samlebåndet bruker ”internal forwarding”. Det vil si at hvis en instruksjon trenger resultatet av en tidligere instruksjon, holder det at de kommer etter hverandre i utføringssteget (trenger ikke vente på tilbakeskrivingssteget).

En kompilator har generert følgende maskinkode til denne prosessoren:

```
I1: Load R4 <- [R5]      ; R5 gir minneadr. Legg i R4.
I2: R5 <- R5 + 4
I3: R7 <- R4 + R9
I4: Store [R6] <- R7     ; R6 gir minneadr. Hent fra R7.
I5: Load R4 <- [R5]
I6: R6 <- R6 + 4
I7: R7 <- R4 + R9
I8: Store [R6] <- R7
```

Lag en tabell som viser hvordan instruksjonene beveger seg gjennom samlebåndet. Pass på å forklare hvordan du har tenkt.

I og med at det brukes både ut-av-rekkefølge tildeling og fullføring, vil ut- og antiavhengigheter kunne medføre problemer. Bruker derfor registeromdøping først. Koden blir da:

```
I1: Load R4b <- [R5a]
I2: R5b <- R5a + 4
I3: R7b <- R4b + R9a
I4: Store [R6a] <- R7b
I5: Load R4c <- [R5b]
I6: R6b <- R6a + 4
I7: R7c <- R4c + R9a
I8: Store [R6b] <- R7c
```

Med ut- og antiavhengigheter ute av bildet, er utføringen begrenset av sanne dataavhengigheter og ressurskonflikter. De sanne dataavhengighetene er: I1 ➔ I3, I2 ➔ I5, I3 ➔ I4, I5 ➔ I7, I6 ➔ I8, I7 ➔ I8 (En skriver det en senere leser). For hver sanne dataavhengighet A ➔ B, er det slik at B må komme etter A i utføringssteget (senere klokkesyklus). Når det gjelder ressurskonflikter, kan disse oppstå hvis man har to load/store-instruksjoner eller tre heltallsinstruksjoner klare til utføring samtidig. Da må en av disse vente siden man bare har mulighet til å ha to heltallsinstruksjoner og en load/store-instruksjon under utføring i hver klokkesykel. Dermed får man følgende tabell:

Syklus		**Dekoder**			**Vindu**		**Heltall**		**Lager**		**Tilbakeskriv**	
1		I1	I2									
2		I3	I4		I1, I2		I2		I1			
3		I5	I6		I3, I4				I1		I2	
4		I7	I8		I3-I6		I3	I6	I5		I1	
5					I4,I7,I8				I5		I3	I6
6					I4,I7,I8		I7		I4		I5	
7					I8				I8		I4	I7
8											I8	