Løsningsforslag eksamen TDT4160 høsten 2005

NB! Ved en feil er summen av prosentvektene for alle oppgavene 90 % og ikke 100 %. For å korrigere dette, ble alle resultater delt på 0,9.

Oppgave 1

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

a) Underflyt vil si at en flyttallsoperasjon gir et resultat som er for nærme null til å kunne bli representert. I praksis vil dette si en eksponent som er mindre (negativt tall) enn hva som kan bli representert.

b) I1: R3 = R3 * R5
I2: R4 = R3 + 1
I3: R3 = R5 + 1
I4: R7 = R3 * R4

- Sann dataavhengighet. En senere instruksjon leser det en tidligere har skrevet. I eksempelet leser I2 R3 som blir skrevet av I1. I1 må dermed få skrive før I2 leser.
- Utavhengighet. To instruksjoner skriver til samme lokasjon (for eksempel register). I eksempelet skriver både I1 og I3 til R3. I1 må dermed få skrive før I3 skriver.
- Antiavhengighet. En senere instruksjoner skriver til samme lokasjon (for eksempel register) som en tidligere leser fra. I eksempelet skriver I3 til R3 som I2 leser fra. I2 må dermed få lese før I3 skriver.

c) Følgende steg skjer under behandling av et avbrudd:

- Enhet aktiverer avbruddslinje
- CPU godkjenner avbrudd
- Enhet oppgir avbruddsvektor
- CPU lagrer programinformasjon (programteller, statusord)
- CPU bruker avbruddsvektor til å finne avbruddsrutine
- Avbruddsrutine lagrer registerverdier
- Avbruddsrutine finner ekstra info om avbrudd (hvilken enhet, hvilken situasjon har oppstått)
- Avbrudd håndteres
- Avbruddsrutine setter tilbake registerverdier
- Programteller og statusord blir satt tilbake

d) Hvis LOAD og STORE aksesserer forskjellige registre og hovedlagerceller er de ikke avhengig av hverandre og de kan fritt bytte rekkefølge. Hvis LOAD og STORE bruker samme register, har vi en vanlig avhengighet som håndteres som normalt. Hvis LOAD og STORE bruker samme (eller overlappende) hovedlagercelle har vi også en avhengighet. Denne kan være vanskelig å oppdage i kjøretid ettersom adressen kan være ukjent ved kompilering. Eksempel:

- Store [R4], R2
- Load R6, [R8]

Hvis R4 = R8 betyr dette i praksis at R6 blir satt lik R2. Hvis instruksjonene derimot bytter rekkefølge vil ikke dette være resultatet lengre. IA-64 inneholder ”advanced load”-instruksjoner som løser akkurat dette problemet.

Oppgave 3

a) Lokalitetsprinsippet er årsaken til at hurtigbuffre fungerer så bra som de gjør. Dette prinsippet sier at (hoved-)lager ikke blir aksessert i et tilfeldig mønster. Dermed kan man forutse fremtidige lageraksesser og passe på å ha data i hurtigbuffer før de trengs. Lokalitetsprinsippet kan deles i to: Lokalitet i tid og lokalitet i rom.
Lokalitet i tid sier at hvis en lagerlokasjon blir aksessert, er det sannsynlig at samme lokasjon vil bli aksessert i nær fremtid. Hvis noe hentes fra hovedlager, bør det derfor lagres i hurtigbuffer.
Lokalitet i rom sier at hvis en lagerlokasjon blir aksessert, er det sannsynlig at en lokasjon i nærheten vil bli aksessert i nær fremtid. Hvis noe hentes fra hovedlager, bør data fra etterfølgende adresser derfor også hentes og lagres i hurtigbuffer.

b) ”Write through”: Data som blir skrevet til en hurtigbuffer, blir også skrevet til lavere nivå (1. nivå => 2. nivå; 2. nivå => hovedlager). Dette er en enkel løsning som også sikrer at vi ikke har inkonsistens (ulik verdi for samme adresse på forskjellig nivå i minnehierarkiet). Ulempen er at det medfører flere skriveoperasjoner enn strengt tatt nødvendig. En fordel er at innlegging av nye hurtigbufferlinjer tar kortere tid ettersom man slipper å skrive tilbake en gammel. Dessuten er dette en enklere løsning enn ”Write back”.
”Write back”: Data som blir skrevet til en hurtigbuffer blir ikke skrevet til lavere nivå før det er absolutt nødvendig. Dette er en mer komplisert løsning som tillater inkonsistens i minnehierarkiet. Fordelen er at det blir så få skriveoperasjoner som mulig.
Ulempen med ”write though” betyr ikke så mye når det er snakk om skriving fra 1. til 2. nivås hurtigbuffer. Disse ligger uansett inne på prosessorbrikken begge to. Derimot tar skriveoperasjoner mellom 2. nivås hurtigbuffer og hovedlager ”lang” tid og det er fornuftig å ta ekstra kompleksitet i form av ”write back” for å minimere antallet.

c) Med direkte avbildning blir hovedlageradresser brukt som følger: Siden vi har 4 linjer, 2 ord per linje og hvert ord er på 32 bit, vil vi få følgende oppdeling av adressen:

- Tag 15 (= 20 - 2 - 1 - 2) bit
- Linje 2 bit
- Ord 1 bit
- Byte 2 bit

Dette betyr at innholdet i minneadresser hvor linje-bit'ene er like, vil havne på samme hurtigbufferlinje. Instruksjoner (starter på adresse 0 og er lagret fortløpende, hver instr. er 32 bit):

- Instruksjon 1 & 2, startadr 0000 0000 ➔ linje 0
- Instruksjon 3 & 4, startadr 0000 1000 ➔ linje 1
- Instruksjon 5 & 6, startadr 0001 0000 ➔ linje 2
- Instruksjon 7 & 8, startadr 0001 1000 ➔ linje 3

Når det gjelder data, var det oppgitt at de lå lagret fra adresse 0010 1000. De 5 siste bit'ene gir linje, ord og byte-plassering i hurtigbufferet. Dette gir:

- X, adr 0010 1000 ➔ linje 1, ord 0, byte 0

Lister opp innholdet i hurtigbuffer etter at hver instruksjon har blitt utført. Kaller de 8 bit'ene som blir lest for "X".

1. MOV R0, #0

Linje	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
0	Instruksjon 1				Instruksjon 2			
1								
2								
3								

2. MOV R5, #40

Linje	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
0	Instruksjon 1				Instruksjon 2			
1								
2								
3								

3. LD R1, [R5]

Linje	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
0	Instruksjon 1				Instruksjon 2			
1	X	(…)	(…)	(…)	(…)	(…)	(…)	(…)
2								
3								

(…) = Data etterfølgende X i hovedlager. Instruksjon 3 og 4 vil først bli hentet inn til linje 1, men vil rett etterpå bli overskrevet når LOAD-instruksjonen aksesserer hovedlageret.

4. ADD R1, R1, R9

Linje	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
0	Instruksjon 1				Instruksjon 2			
1	Instruksjon 3				Instruksjon 4			
2								
3								

5. ST [R5], R1

Linje	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
0	Instruksjon 1				Instruksjon 2			
1	Y	(…)	(…)	(…)	(…)	(…)	(…)	(…)
2	Instruksjon 5				Instruksjon 6			
3								

Her får vi skriving til en adresse der data ikke finnes i hurtigbuffer. Pga. ”write allocation” hentes data (hurtigbufferlinje) inn først. Kaller innholdet i R1 for Y.

6. ADD R5, R5, #4

Linje	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
0	Instruksjon 1				Instruksjon 2			
1	Y	(…)	(…)	(…)	(…)	(…)	(…)	(…)
2	Instruksjon 5				Instruksjon 6			
3								

7. SUB R0, R0, #1

Linje	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
0	Instruksjon 1				Instruksjon 2			
1	Y	(…)	(…)	(…)	(…)	(…)	(…)	(…)
2	Instruksjon 5				Instruksjon 6			
3	Instruksjon 7				Instruksjon 8			

8. ADD R9, R9, #1

Linje	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
0	Instruksjon 1				Instruksjon 2			
1	Y	(…)	(…)	(…)	(…)	(…)	(…)	(…)
2	Instruksjon 5				Instruksjon 6			
3	Instruksjon 7				Instruksjon 8			

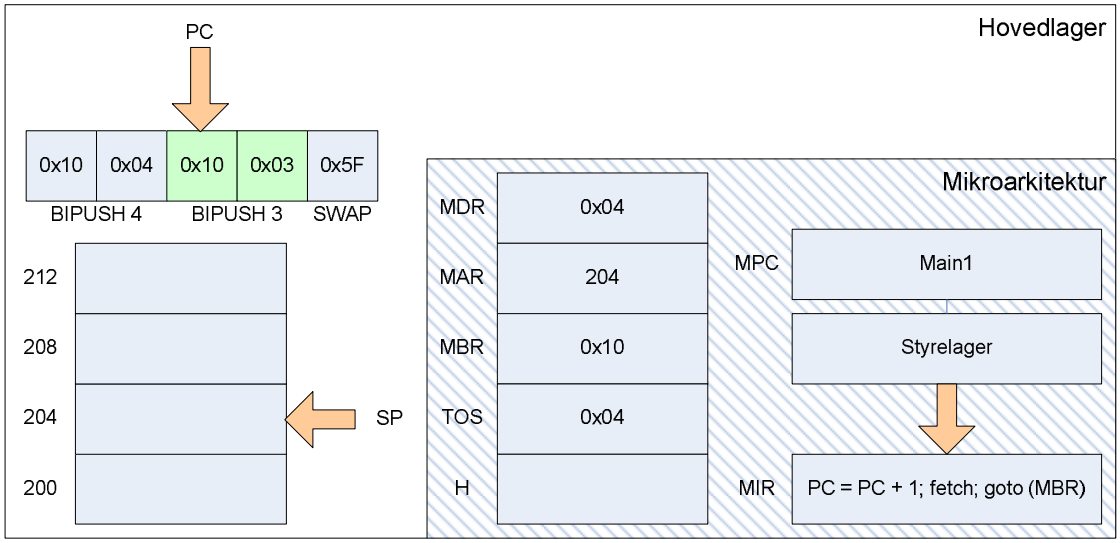
Oppgave 4

a) Mikroarkitektur kan deles opp i styreenhet (kontrollenhet el. ”control unit”) og utførende enhet (”datapath”). Den utførende enhet inneholder registrene og ALU og er den som faktisk utfører operasjoner. Styreenheten inneholder logikk for å få den utførende til å gjøre de rette operasjonene – for eksempel utføre et mikroprogram som implementerer en ISA-instruksjon. I tillegg bestemmer styreenheten neste tilstand, henter inn og tolker ISA-instruksjoner.

b) ISA-instruksjoner er de instruksjonene som er tilgjengelig for (assembly-) programmerere. De definerer dermed grensesnittet mellom maskinvare og programvare.
Styreenheten i mikroarkitekturen leser inn ISA-instruksjoner fra et program og sørger for at disse blir utført av den utførende enheten. Styresignalene som blir generert fra styreenheten kan enten være generert av en sekvensiell krets (hardwired) eller via et mikroprogram. Et mikroprogram er en sekvens av mikroinstruksjoner som til sammen implementerer en ISA-instruksjon. En mikroinstruksjon er dermed en del av et mikroprogram og bestemmer hva den utførende enheten skal gjøre i en hel klokkesyklus. Mikroinstruksjonen vil også inneholde informasjon om hvilken mikroinstruksjon som skal utføres etterpå (styring av styreenheten).

c) De tre figurene er gitt på neste side.

(Pga. av at det i oppgaven står at MIR lastes i slutten av en klokkesyklus mens det i læreboka er slik at MIR lastes i starten av en klokkesyklus, ble det også godtatt at MIR inneholder mikroinstruksjonene til hhv. Bipush3, Bipush3 og Swap6).



Legg merke til at skriveoperasjonen ("wr") mot hovedlager enda ikke har tatt effekt.