Lösningar till tentamen i kursen EDA330

Datorsystemteknik D

1/6 1999

Följande är skisser till lösningar av uppgifterna. Full poäng på en uppgift kräver i de flesta fall en något fylligare motivering. I en del fall är alternativa lösningar möjliga.

1.

a. 32 KB / (32 B/block) = 1 Kblock totalt i cacheminnet. 1 Kblock / (4 block/set) = 256 set totalt i cacheminnet. Blockoffset kräver 5 bitar ($2^5 = 32$). Index kräver 8 bitar ($2^8 = 256$). Eftersom hela adressen är 30 bitar, återstår 30-8-5 = 17 bitar för tag. För varje block behövs också en valid bit, en dirty bit (för write back), och 2 replacements bits (4 block/set). För varje block måste alltså cacheminnet lagra 17+1+1+2+32×8 = 277 bitar. Totalt för hela cacheminnet krävs alltså 277 bitar/block × 1 Kblock = **283648 bitar**.

b. En virtuell adress kräver 27 bitar. Sidstorleken 8 KB innebär att 13 bitar krävs för sidoffset, varför virtuella sidnummer består av 27-13 = 14 bitar. Härav fås att varje process har upp till 16 Ksidor. De fysiska adresserna kräver också 27 bitar (2^{27} B = 128 MB), varför fysiska sidnummer också består av 14 bitar. För varje virtuell sida behöver sidtabellen också lagra en valid bit (finns sidan i primärminnet?), en dirty bit (för write back), 2 protection bits (enligt uppgiften), och 8 replacement bits (tidsstämplar). För varje virtuell sida krävs alltså 14+1+1+2+8 = 26 bitar i sidtabellen. Eftersom varje sidöversättning ska ta upp ett jämnt antal byte krävs alltså 4 byte per virtuell sida. Med 32 processer och 16 Ksidor/process och 4 B/sida krävs totalt 32 × 16K × 4 = **2 MB** för sidtabellerna.

c. **Se kursboken s. 585**.

2.

a. Lösningarna för de båda multiplexrarna skiljer sig bara i om man tittar på rs- eller rt-fältet i instruktionen. Konflikter kan uppstå mellan EX och något eller båda av MEM eller WB. Då det är en konflikt med både MEM och WB ska forward göras från MEM eftersom det senaste värdet för registret finns där. Hänsyn till detta måste alltså tas när man tittar på konflikter med WB (denna kontroll är inte helt korrekt utförd i boken!). Vid kontroll om en konflikt uppstått med ett steg måste två saker kontrolleras; skriver instruktionen i det aktuella steget till något register utom register noll (skrivningar till register noll har inte någon effekt eftersom detta register alltid ska innehålla noll), och skriver den till det register som används i EX. Fallet att ingen konflikt uppstår består i att kontrollera att ingen konflikt uppstår med något av MEM eller WB.
fwdA = 2: MEM.RegWrite & MEM.DestReg≠0 & MEM.DestReg=EX.rs
fwdA = 1: WB.RegWrite & WB.DestReg≠0 & WB.DestReg=EX.rs & (MEM.RegWrite & MEM.DestReg≠0 & MEM.DestReg=EX.rs)
fwdA = 0: (MEM.RegWrite & MEM.DestReg≠0 & MEM.DestReg=EX.rs) & (WB.RegWrite & WB.DestReg≠0 & WB.DestReg=EX.rs)
fwdB = 2: MEM.RegWrite & MEM.DestReg≠0 & MEM.DestReg=EX.rt
fwdB = 1: WB.RegWrite & WB.DestReg≠0 & WB.DestReg=EX.rt & (MEM.RegWrite & MEM.DestReg≠0 & MEM.DestReg=EX.rt)
fwdB = 0: (MEM.RegWrite & MEM.DestReg≠0 & MEM.DestReg=EX.rt) & (WB.RegWrite & WB.DestReg≠0 & WB.DestReg=EX.rt)
Se även kursboken avsnitt 6.4. (OBS! Lösningen i boken är inte helt korrekt. Hänsyn till detta tas vid rättningen av denna tentamensuppgift.)

b. Detta uppstår om det finns en minnesläsning i MEM som ska skriva till något av källregistren för instruktionen i EX. Observera att vi utgår ifrån att en minnesläsning implicerar en registerskrivning varför vi inte behöver kolla MEM.RegWrite. Det är dock inte fel att göra det. Det är också nödvändigt att kontrollera att källregistren används av instruktionen i EX. I den aktuella pipelinen används alltid rs, men rt används inte om imm används istället (ALUSrc=1) och det inte är en store-instruktion (MemWrite=1).
MEM.MemRead & MEM.DestReg≠0 & (MEM.DestReg=EX.rs | (MEM.DestReg=EX.rt & EX.ALUSrc=1 & EX.MemWrite=0)).

c. I de fall som uppstå här räcker det med att göra en **stall under en cykel som håller kvar instruktionerna i IF, ID, och EX**. Instruktionen i MEM hinner då avancera till WB och konflikten kan lösas med forwarding.

3.

a. Det krävs alltid en minst en cykel. Sedan vet man om det blev träff eller miss. Blir det en miss (sannolikhet $1-h_1$) så måste ett block hämtas in från primärminnet. Med sannolikheten w måste det block som då kastas ut skrivas till primärminnet eftersom det uppdaterats och write back tillämpas. Räknat i busscykler tar en blocköverföring t_b (vänta på busstillgång) + 1 (lägg ut adress) + t_1 (vänta på första ord) + $(B-1)t_2$ (vänta på följande ord) + 1 (överför sista ordet) busscykler. Vi gör här det rimliga antagandet att t_1 och t_2 är längre än tiden det tar att överföra ett ord över bussen (dvs en busscykel). Vi antar också att ett ord som ska skrivas till primärminnet överförs först efter åtkomsttiden. Andra antaganden kan godkännas, men bör då redovisas. Eftersom bussen kör på halva processorfrekvensen kräver en blocköverföring alltså $2(t_b+1+t_1+(B-1)t_2+1)$ processorcykler. Medelåtkomsttiden räknat i processorcykler blir då $\mathbf{1 + (1-h_1)(1+w)2(t_b+2+t_1+(B-1)t_2)}$.

b. Även i detta fall krävs alltid minst en cykel. Man får sedan skilja på fallen läsning (sannolikhet r) och skrivning (sannolikhet 1-r). I fallet med läsning måste man om det blev en miss (sannolikhet $1-h_2$) hämta in det sökta blocket såsom beskrevs i förra deluppgiften. Ingen tillbakaskrivning av utbytt block krävs eftersom primärminnet alltid är uppdaterat vid write through. I fallet med skrivning måste alltid ett ord skrivas till primärminnet vilket tar $2(t_b+1+t_1+1)$ processorcykler. Vi har här gjort samma antaganden som i förra deluppgiften. Medelåtkomsttiden räknat i processorcykler blir då totalt $\mathbf{1 + r(1-h_2)2(t_b+2+t_1+(B-1)t_2) + (1-r)2(t_b+2+t_1)}$. $\mathbf{h_1 \neq h_2}$ **eftersom olika strategier tillämpas för inhämtning av block till cache-minnet**.

c. **Se kursboken s. 554**.

d. **Se kursboken s. 554**.

4.

a. Baserat på vad som beskrivs i uppgiften kan vi utgå från att I/O-väntetiden för beräkningen är försumbar, varför det handlar om att tillämpa formeln beräkningstid = CPU-tid = $I \times CPI \times T_c$, eller i detta fall beräkningstid = $I \times CPI / f$, där f är processorns klockfrekvens. Beräkningstiden och klockfrekvensen f är kända, varför det återstår tre okända; I, CPI_{100} = CPI vid 100 MHz, och CPI_{150} = CPI vid 150 MHz. Det finns också tre olika fall beskrivna varför det går att sätta upp ett enkelt ekvationssystem och lösa ut de okända. I fall 1 är beräkningstiden 12 s, frekvensen 100 MHz, antalet exekverade instruktioner I, och CPI-talet CPI_{100}. I fall 2 är beräkningstiden 10 s, frekvensen 150 MHz, antalet exekverade instruktioner I, och CPI-talet CPI_{150}. I fall 3 är beräkningstiden 9 s, frekvensen 150 MHz, antalet exekverade instruktioner I - 60×10^6, och CPI-talet CPI_{150}. Ekvationerna är alltså:

$$12 \text{ s} = I \times CPI_{100} / 100 \text{ MHz}$$
$$10 \text{ s} = I \times CPI_{150} / 150 \text{ MHz}$$
$$9 \text{ s} = (I - 60\times10^6) \times CPI_{150} / 150 \text{ MHz}$$

Ur dessa ekvationer kan man lösa ut att $I = 600\times10^6$, **CPI vid 100 MHz = 2, och CPI vid 150 MHz = 2,5**.

b. Förändringen beror på att minnessystemets hastighet inte förändrats trots att processorn går snabbare, varför det blir **fler stallcykler pga cachemissar vid den högre klockfrekvensen**.

c. Baserat på observationen i föregående deluppgift kan man sätta upp följande följande ekvation för CPI-talet:

$$CPI = CPI_{ideal} + P_{miss} \times MP$$

där CPI_{ideal} är CPI om alla minnesreferenser resulterar i cache-träffar, P_{miss} är sannolikheten att en instruktion ger upphov en cache-miss, och MP (miss penalty) är det genomsnittliga antalet processorcykler som processorn måste göra stall vid en cache-miss. P_{miss} kan beräknas ur de givna uppgifterna som 0,01 (sannolikhet för instruktionsmiss) + 0,2×0,1 (sannolikhet för att en instruktion kräver en minnesreferens som ger upphov till en datamiss), dvs totalt 0,03 eller 3%. Eftersom vi har en processor med en enkel rak pipeline kommer minnesreferenserna att ske i samma ordning oberoende av klockfrekvensen, vilket gör att P_{miss} här är konstant (med en modern processor med dynamisk schemaläggning hade inte detta gällt!). Skillnaden i CPI mellan de två fallen i deluppgift a kan då uttryckas som

$$\Delta CPI = CPI_{150} - CPI_{100} = 0{,}03 \times \Delta MP,\ \Delta MP = MP_{150} - MP_{100}$$

där MP_x är miss penalty räknat i processorcykler vid klockfrekvensen x MHz. Vi får eftersom $\Delta CPI = 0{,}5$ att $\Delta MP = 16{,}7$. Vidare kan MP_x uttryckas som

$$MP_x = MA \times x \times 10^6$$

där MA är den verkliga minnesåtkomsttiden vid en cachemiss. Eftersom minnessystemet inte påverkas av klockfrekvensändringen så är MA konstant. Detta ger

$$\Delta MP = 16{,}7 = MA \times (150\text{-}100) \times 10^6 \text{ Hz, och därmed}$$
$$MA = 16{,}7 / (50\times10^6 \text{ Hz}) = 333 \text{ ns.}$$

Eftersom varje cachemiss kräver överföring av 1,11 cacheblock får vi att åtkomsttiden per cacheblock blir 333 ns / 1,11 = **300 ns**. Det motsvarar 30 processorcykler per blocköverföring (inte per miss!) vid 100 MHz, och 45 processorcykler per blocköverföring vid 150 MHz.

Observera att MP egentligen alltid ska avrundas uppåt till ett helt antal cykler. Med tanke på att vi här talar om statistiska medelvärden, att det i realiteten troligen finns många fler felkällor, samt att vi ändå bara försöker göra en uppskattning så kan man bortse från det här. Skillnaden i resultat blir ändå relativt liten.

5.

Deluppgift	1	X	2
a		X	
b			2
c	1		
d	1		
e		X	
f			2
g			2
h		X	
i			2
j		X	
k	1		
l			2