Lösningar till tentamen i kursen EDA330

Datorsystemteknik

30/5 1998

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. **Programmet tar två tal, och returnerar dem sorterade i storleksordning**.

b. **Indata tas från register $a0 och $a1. Resultaten returneras i register $v0 (det minsta talet) och $v1 (det största)**.

c. Tre stall-cykler orsakas av databeroendet mellan slt-instruktionen och den därpå följande hoppinstruktionen (vi antar att ett nytt registervärde kan läsas först en cykel efter att det skrivits). Varje hoppinstruktion orsakar dessutom tre stallcykler (det tar två cykler tills hoppet beräknats, och ytterligare en cykel tills rätt instruktion hämtats in). För de givna argumenten kommer endast den första hoppinstruktionen att utföras, så bidraget från hoppkonflikter blir totalt tre stallcykler. **Totalt krävs** alltså **6 extra cykler** orsakade av pipelinekonflikter. OBS! Det är tillåtet att anta att skrivning och läsning kan ske samtidigt, men det krävs då en cykel mindre.

d. I den givna koden behöver endast en återhoppsinstruktion, `jr $ra`, läggas till allra sist (efter etiketten L2). Dessutom måste en etikett (label), t.ex. SORT sättas på första instruktionen. Inga register behöver sparas undan eftersom endast $a0, $a1, $v0, och $v1 används, och inga ytterligare subrutinanrop görs från subrutinen. Den modifierade koden blir alltså:

```
SORT:
     slt  $v0, $a0, $a1
     beq  $v0, $zero, L1
     add  $v0, $a0, $zero
     add  $v1, $a1, $zero
     beq  $zero, $zero, L2
L1:
     add  $v0, $a1, $zero
     add  $v1, $a0, $zero
L2:
     jr   $ra
```

Anrop av subrutinen kan då enkelt ske genom:

```
     addi      $a0, $zero, 5
     addi      $a1, $zero, 1
     jal       SORT
```

Denna lösning förutsätter att adressen som motsvarar SORT kan representeras i en jump-instruktion.

2.

a. **Se avsnitt 5.6 i kursboken.** I/O device request, invoke OS from user program, arithmetic overflow, undefined instruction, hardware malfunction.

b. **Se avsnitt 6.7 i kursboken.** De modifieringar som behöver göras är att lägga till ett register (EPC) för att lagra adress (+4) till den instruktion som orsakat avbrottet, möjlighet att tömma pipelinen fram till den orsakande instruktionen, ett Cause-register för att lagra orsaken till avbrottet, samt en möjlighet att ladda PC med adressen till avbrottshanteraren.

c. **Se avsnitt 6.7 i kursboken**.

3.

a. Antal block i cache = **S/B**.

b. Antal set i cache = antal block i cache/associativitet = **S/(B*A)**.

c. Adresserna delas upp i tag, index, och offset. Offset kräver b bitar eftersom B är blockstorleken i bytes och varje adress är en byte-adress. Antalet bitar i index bestäms av antalet set eftersom varje set ska ha ett eget index. Antal set fås från förra deluppgiften, och antalet bitar i index blir alltså $\log_2(S/(B*A))$, vilket också kan skrivas som s-b-a. Eftersom en hel adress har W bitar kan antalet bitar i en tag lätt räknas ut genom att dra bort antalet bitar för index och offset, dvs W-(s-b-a)-b = W-s+b+a-b = **W-s+a**.

d. För varje block ska lagras data, tag, valid-bit, och dirty-bit (det behövs ingen bit för utbytesalgoritmen eftersom den är random). Data är ett block om B byte, vilket alltså är 8B bitar. Tag är enligt förra deluppgiften W-s+a bitar. Valid och dirty kräver tillsammans två bitar. Totalt måste man alltså per block lagra 8B+W-s+a+2 bitar, och för hela cacheminnet **S/B*(8B+W-s+a+2)** bitar.

e. Parallell cacheåtkomst och adressöversättning kräver att sidoffset-delen av adresserna (den som inte översätts) räcker till index och blockoffset i cacheadresserna. Antalet bitar för index och blockoffset fås från deluppgift c ovan, och är s-b-a+b=s-a. Sidoffset måste alltså vara minst s-a bitar, vilket innebär att sidorna måste vara större än 2^{s-a} = **S/A** bytes.

4. Det finns fyra olika konflikter som kan orsaka pipeline-bubblor och därmed påverka CPI: Hoppkonflikter då hopp tas, datakonflikter då en load-instruktion följs direkt av en instruktion som använder det som läses in (kan ej hanteras med forwarding), cachemiss vid instruktionshämtning, och cachemiss vid läsning eller skrivning av data. Vi behandlar dess i tur och ordning:

   Då ett hopp tas har redan en felaktig instruktion hämtats in som måste ersättas av en nop. Detta händer för 20% * 70% = 14% av instruktionerna och ger därmed ett bidrag på 0,14 * 1 = 0,14 till CPI.

   Då en minnesläsning direkt följs av en instruktion som använder det som läses in måste stall göras under en cykel tills minnesläsningen genomförts i MEM-steget. En nop får då skjutas in. Detta händer för 20% * 50% = 10% av instruktionerna och ger därmed ett bidrag på 0,1 * 1 = 0,1 till CPI.

   Vid en miss i ICACHE måste man vänta tills det sökta cacheblocket lästs in från primärminnet över processor-minnebussen. Den läsningen tar 2 busscykler för väntan på upptagen buss och arbitrering, 1 busscykel för att skicka adressen till minnet, 200 ns * 100 MHz = 20 busscykler för väntan på åtkomst av första ordet, 20 ns * 100 MHz = 2 busscykler för sändning av första ordet och åtkomst av andra ordet, 2 busscykler för sändning av andra ordet och åtkomst av tredje ordet, 2 busscykler för sändning av tredje ordet och åtkomst av fjärde ordet, och 1 busscykel för sändning av fjärde ordet. En busstransaktion tar alltså totalt 2+1+20+2+2+2+1 = 30 busscykler. Eftersom bussen och processorn har samma klockfrekvens betyder det att miss penalty blir 30 processorcykler. Detta inträffar för 1% (träffsannolikhet för ICACHE är 99%) av instruktionerna, och ger därmed ett bidrag på 0,01 * 30 = 0,3 till CPI.

   Vid en miss i DCACHE måste en transaktion göras på samma sätt som vid missar i ICACHE för att hämta det sökta blocket, vilket alltså tar 30 cykler. Det kan dock ibland krävas en extra transaktion på grund av write-back för att skriva ett block som kastas ut till primärminnet. Det inträffar vid 50% av missarna, och en miss i DCACHE kräver därför i genomsnitt 30 + 0,5*30 = 45 cykler extra. Detta sker alltså för de (20% + 10%)*(1-90%) = 3% av instruktionerna som är läsningar eller skrivningar som orsakar en miss, och ger därmed ett bidrag på 0,03 * 45 = 1,35 till CPI.

   Då CPI för en pipeline utan konflikter är 1 (vi bortser från effekter av uppstart), blir totala CPI i detta fall alltså 1 + 0,14 + 0,1 + 0,3 + 1,35 = **2,89**.

5.

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