Lösningar till tentamen i kursen EDA330 för D och EDA370 för E

# Datorsystemteknik

19/8 2000

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 absolutbeloppet av differensen mellan talen.**

b. **Indata tas från register $a0 och $a1. Resultatet returneras i register $v0.**

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 både den första och andra hoppinstruktionen att utföras, så bidraget från hoppkonflikter blir totalt sex stallcykler. **Totalt krävs** alltså **9 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. Den givna koden använder sig av register $t0 och $s0. Register $s0 måste lagras undan på stacken under exekveringen av subrutinen samt återhämtas vid subrutinens slut. En återhoppsinstruktion, `jr $ra`, måste läggas allra sist. En etikett (label), t.ex. DIFF, måste sättas på första instruktionen. Den modifierade koden blir således:

```
DIFF:
     addi $sp, $sp, -4
     sw   $s0, 0($sp)
     slt  $t0, $a0, $a1
     beq  $t0, $zero, L1
     sub  $s0, $a1, $a0
     beq  $zero, $zero, L2
L1:
     sub  $s0, $a0, $a1
L2:
     add  $v0, $s0, $zero
     lw   $s0, 0($sp)
     addi $s0, $s0, 4
     jr   $ra
```

Det är också tillåtet att modifiera om koden på följande vis:

```
DIFF:
     slt $t0, $a0, $a1
     beq $t0, $zero, L1
     sub $t1, $a1, $a0
     beq $zero, $zero, L2
L1:
     sub $t1, $a0, $a1
L2:
     add $v0, $t1, $zero
     jr  $ra
```

Anrop av subrutinen kan enkelt ske genom:

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

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

2. **Se kursboken**.
*Always stall*: Stanna alltid pipelinen från det att en hoppinstruktion hämtats in tills hoppadress och hoppvillkor är beräknade. Ger stor effektivitetsförlust, men är relativt enkelt att implementera.
*Assume not taken*: Fortsätt exekvera instruktioner direkt efter ett hopp som om hoppet inte skulle tas. Om hoppet tas måste pipelinen tömmas på felaktigt startade instruktioner. Kräver möjlighet att tömma pipelinen, men är effektivare än *always stall*.
*Assume taken*: Så snart en hoppinstruktion hämtats börjar instruktioner från hopp destinationen hämtas. Om hoppet inte tas måste pipelinen tömmas på felaktigt startade instruktioner. Kräver möjlighet att snabbt upptäcka hoppinstruktioner och beräkna hoppadresser, men är effektivare än *assume not taken* eftersom det är fler hopp som tas än som inte tas.
*Delayed branch*: Ändra betydelsen av hoppinstruktioner så att programmeraren/kompilatorn vet att ett visst antal instruktioner efter en hoppinstruktion utförs innan hoppet tas. Kräver ingen extra hårdvara att implementera, men kan vara svår att utnyttja effektivt för programmerare/kompilator.

3.

a. Utgå från formeln för CPU-tid:
$T = I * CPI * T_C$
Kalla totaltiden för konvertering med det gamla systemet för $T_{gammal}$:
$T_{gammal} = T_{beräkn,gammal} + T_{övriga,gammal}$
där $T_{beräkn,gammal} = 0,6 * I_{gammal} * 2,5 / (350 * 10^6)$ och
$T_{övriga,gammal} = 0,4 * I_{gammal} * 1,3 / (350 * 10^6)$
Totaltiden för konvertering med det nya systemet, $T_{ny}$, blir:
$T_{ny} = T_{beräkn,ny} + T_{övriga,ny}$
där $T_{beräkn,ny} = 0,2 * I_{ny} * 0,5 / (500 * 10^6)$ och
$T_{övriga,ny} = 0,8 * I_{ny} * 1,3 / (500 * 10^6)$
Vi vet att $0,8 * I_{ny} = 0,4 * I_{gammal}$ vilket medför att

$I_{ny} = I_{gammal} * 0,4 / 0,8 = 0,5 * I_{gammal}$
$T_{ny} = 0,2 * 0,5 * I_{gammal} * 0,5 / (500 * 10^6) + 0,8 * 0,5 * I_{gammal} * 1,3 / (500 * 10^6)$

Uppsnabbning $= T_{gammal} / T_{ny} = (0,6 * I_{gammal} * 2,5 / (350 * 10^6) + 0,4 * I_{gammal} * 1,3 / (350 * 10^6)) / (0,2 * 0,5 * I_{gammal} * 0,5 / (500 * 10^6) + 0,8 * 0,5 * I_{gammal} * 1,3 / (500 * 10^6)) = (0,6 * 2,5 / (350 * 10^6) + 0,4 * 1,3 / (350 * 10^6)) / (0,2 * 0,5 * 0,5 / (500 * 10^6) + 0,8 * 0,5 * 1,3 / (500 * 10^6)) =$ **5,06 ggr**

b. **Se kursboken.**

4.

a. Tabellen nedan ger innehållet i varje cacheblock efter varje referens genom att ange tag för de block som ligger i cacheminnet. De fyra första kolumnerna ger byte-adressen, motsvarande block-adress (i detta fall byte-adress/4), motsvarande set-nummer (i detta fall block-adress modulo 4), och tag (i detta fall block-adress/4). Tag för senast refererade cache-block är markerad med fet stil.

| Set block | | | | | 0 | | | | 1 | | | | 2 | | | | 3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte | Block | Tag | Set | Hit | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| 12 | 3 | 0 | 3 | n | | | | | | | | | | | | | **0** | | | |
| 30 | 7 | 1 | 3 | n | | | | | | | | | | | | | 0 | **1** | | |
| 13 | 3 | 0 | 3 | y | | | | | | | | | | | | | **0** | 1 | | |
| 76 | 19 | 4 | 3 | n | | | | | | | | | | | | | 0 | 1 | **4** | |
| 5 | 1 | 0 | 1 | n | | | | | **0** | | | | | | | | 0 | 1 | 4 | |
| 14 | 3 | 0 | 3 | y | | | | | 0 | | | | | | | | **0** | 1 | 4 | |
| 44 | 11 | 2 | 3 | n | | | | | 0 | | | | | | | | 0 | 1 | 4 | **2** |
| 116 | 29 | 7 | 1 | n | | | | | 0 | **7** | | | | | | | 0 | 1 | 4 | 2 |
| 15 | 3 | 0 | 3 | y | | | | | 0 | 7 | | | | | | | **0** | 1 | 4 | 2 |
| 61 | 15 | 3 | 3 | n | | | | | 0 | 7 | | | | | | | 0 | **3** | 4 | 2 |
| 101 | 25 | 6 | 1 | n | | | | | 0 | 7 | **6** | | | | | | 0 | 3 | 4 | 2 |
| 76 | 19 | 4 | 3 | y | | | | | 0 | 7 | 6 | | | | | | 0 | 3 | **4** | 2 |
| 6 | 1 | 0 | 1 | y | | | | | **0** | 7 | 6 | | | | | | 0 | 3 | 4 | 2 |
| 40 | 10 | 2 | 2 | n | | | | | 0 | 7 | 6 | | **2** | | | | 0 | 3 | 4 | 2 |
| 31 | 7 | 1 | 3 | n | | | | | 0 | 7 | 6 | | 2 | | | | 0 | 3 | 4 | **1** |

**Hit rate = 5/15 = 33%**

b. Beteckna blockstorleken B räknad i byte, och datakapaciteten räknad i byte N. Antal bitar för blockoffset = b = $\log_2$ B. Antal block i cacheminnet = N/B. Antal set i cacheminnet = N/2B. Antal bitar för setindex = s = $\log_2$ N/2B = $\log_2$ N - b - 1. Antal bitar för tag = t = 30 - s - b = 30 - $\log_2$ N + b + 1 - b = 31 - $\log_2$ N.

Antal bitar att lagra per block= 8B+ 31 - $\log_2$ N + 1 = 8B + 32 - $\log_2$ N.

Antal byte att lagra per block = B + (32 - $\log_2$ N)/8 = B + 4 - $\log_2$ N/8
Totalt antal byte i cachminnet = 72 KB = N/B * (B + 4 - $\log_2$ N/8).

Tillåtna värden på B är givna i uppgiften, och vi ser att $\log_2$ N måste vara jämnt delbart med 8, dvs N är $2^{16}$ , $2^8$ eller $2^0$ (större N är inte möjliga eftersom N inte kan överskrida totala cache-storleken). Det är nu lätt att se att enda möjliga lösningen är N=$2^{16}$ och B=16. Alltså är **datakapaciteten 64 KB och blockstorleken 16 B.**

5.

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