LÖSNINGAR 2020-06-04

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. Två stall-cykler orsakas av databeroendet mellan slt-instruktionen och den därpå följande hoppinstruktionen (vi antar att ett nytt registervärde kan läsas i samma cykel som 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 **5 extra cykler** orsakade av pipelinekonflikter. OBS! Det är tillåtet att anta att skrivning och läsning inte kan ske samtidigt, men det krävs då en cykel mer.

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.
Antal cacheblock: 128/16 = 8.

(a) Direktmappad cache -> 1 block per set -> 8 sets -> 3 bits index. 16 bytes per block -> 4 bits offset. Resten är Tag.

Tabell:

Dec	bin	index	h/m	Kommentar
4	00000 000 0100	0	m	första access till detta index
8	00000 000 1000	0	h	samma Tag som föregående access till detta index
144	00001 001 0000	1	m	första access till detta index
1116	01000 101 1100	5	m	första access till detta index
8	00000 000 1000	0	h	samma Tag som föregående access till detta index
512	00100 000 0000	0	m	ny Tag för detta index
1104	01000 101 0000	5	h	samma Tag som föreg access till detta index

(b) Associativity är nu = 2, dvs 2 blocks per set -> 4 sets -> 2 bits Index fält.

dec	bin	index	h/m	Comment
4	000000 00 0100	0	m	första access till detta index
8	000000 00 1000	0	h	samma Tag som föreg access till detta index
144	000010 01 0000	1	m	första access till detta index
1116	010001 01 1100	1	m	second access to this index
8	000000 00 1000	0	h	samma Tag som föreg access till detta index
512	001000 00 0000	0	m	2:a access till detta index
1104	010001 01 0000	1	h	Tag finns för detta index

3.

a) Total CPU time för programmet är $T = I \cdot CPI \cdot T_C = I \cdot CPI/f_C$. Så $CPI = T \cdot f_C/I$.

Vi vet att $T = 4{,}2$ och $f_C = 100 \cdot 10^{6,}$ och $I = 200 \cdot 10^6 + 20 \cdot 2 \cdot 10^6 = 240 \cdot 10^6$, så $CPI = 100 \cdot 4{,}2/240 = 1.75$.

b)
Alla cachemissar sammanlagt ger ett CPI bidrag på 1,75 – 1,5 = 0,25. 1% av alla instruktionshämtningar missar i cache och det kostar 10 cykler varje gång; så CPI-bidraget från dessa missar är $0{,}01 \cdot 10 = 0{,}1$. Därför ger datacachemissarna ett CPI-bidrag på 0,25 – 0,1 = 0,15.

Det totala antalet instruktioner som accessar dataminnet är $0{,}1 \cdot 200 \cdot 10^6 + 0{,}4 \cdot 20 \cdot 2 \cdot 10^6 = 36 \cdot 10^6$; detta motsvarar 36/240 = 0,15 så med en misskostnad på 10 cykler måste det vara 1 miss av 10 accesser om CPI-bidraget skall vara 0,15. Därför är andelen missar 10%.

(c) Skulle kunna vara om de två cachearna använder olika blockstorlekar för då kan misskostnaden vara olika. Ett större block kan ta längre tid att föra mellan cache och primärminnet.

(d) Vi såg att datacachemissarna gav ett CPI-bidrag på 0,15, och nu är misskostnaden 1,5x större, så det extra CPI-bidraget skulle vara 0,075 och totala CPI bli 1,75 + 0,075 = 1,825. Klockfrekvensen och det totala antalet instruktioner är oförändrat så därför minskar prestanda med faktorn 1,75/1,825 = 0,96.

4.

a) 1 block = 4 ord = 16 byte = 128 bitar. Av adressen krävs alltså 4 bitar i blockoffset. Cacheminnet kan lagra 32 KB/16B = 2048 block. Tvåvägs associativitet innebär 2 block/set, och alltså totalt 1024 set i cacheminnet. Av adressen krävs därför 10 ($2^{10} = 1024$) indexbitar för att hitta i vilket set ett block kan ligga. Det finns maximalt 64 MB primärminne så de fysiska adresserna som används av cachet måste vara 26 bitar breda (2^{26} B = 64 MB). För varje block i cacheminnet måste därför 26-10-4 = 12 bitar tag lagras. För varje block krävs dessutom en valid bit och en dirty bit (pga write-back-strategin).

Totalt krävs alltså 128+12+1+1 = 142 bitar/block. Med 2048 block blir totala cache-storleken 2048*142 = **290 816 bitar** .

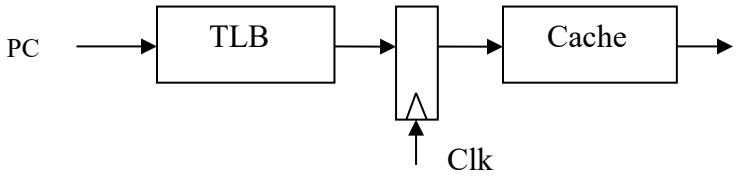
b) Sidoffset kräver 14 bitar (2^{14} B = 16 KB). Virtuella sidnummer kräver därmed 32-14 = 18 bitar, vilket motsvarar 2^{18} = 256 K sidor. Fysiska sidnummer kräver 26-14 = 12 bitar. I sidtabellen ska för varje virtuell sida kunna lagras fysiskt sidnummer (12 bitar), valid bit, dirty bit, protection bit, och två bitar för utbytesalgoritmen, dvs totalt 17 bitar. Varje sidtabell kräver därmed 256 K sidor * 17 bitar/sida = 4 456 448 bitar. Med 32

processer (som var och en har sin egen sidtabell) krävs totalt 32 * 4 456 448 = 142 606 336 bitar = ca 17 MB för sidtabeller.

c) TLB kan lagra information om 64 virtuella sidor. Den information som behöver lagras är fysiskt sidnummer, protection bit, och dirty bit (talar om ifall sidan uppdaterats), totalt 14 bitar. För varje sida måste också lagras en valid bit och en tag. Eftersom TLB är fullt associativ och uppslagning sker med virtuella sidnumret som adress, så måste hela virtuella sidnumret (18 bitar) lagras som tag. Alltså måste 12 + 1 + 1 + 1 + 18 = 33 bitar lagras för varje plats i TLB, och totala antalet bitar i TLB blir 64*33 = **2112 bitar** . (not. man kan även tänka sig att LRU-bitarna finns i TLBn. Då tillkommer 2 bitar per TLB ingång. Då får vi 64*35=**2240 bitar**.

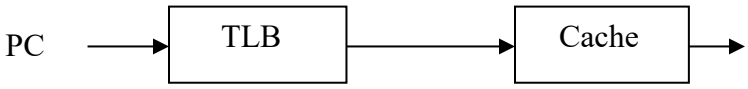
d)
Om fysiskt adresserad cache :

1. Med pipelining av TLB & Cache :



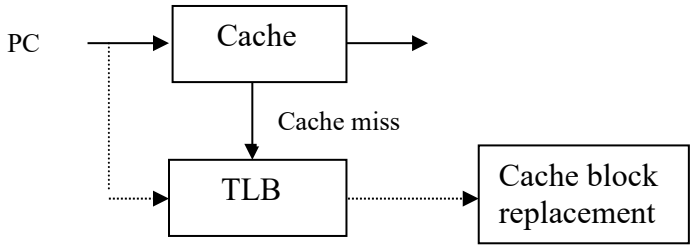
Långsammaste pipesteget begränsar så f_{Clk} = 1 / 3 ns = 333 MHz .

2. Ingen pipelining av TLB &Cache :



f_{Clk} = 1 / (2ns + 3ns) = 200 MHz.

3. Om virtuellt adresserad cache:



TLB är här inte i den ”kritiska vägen”. TLB access endast vid en cache miss.

f = 1 / 3 ns = 333 MHz