

1.

TSEA28

davwi 279

31/5 2021

sid 1.

a) 5.  $buss := GR_x$   $ASR := buss$ ,  $R := 1$ ,  $S := 0$ ,  $MP_C = MP_C + 1$

// läser in GR3 på ASR som  
effektiv-adress

6.  $buss := GR_x$   $AR := buss$ ,  $R := 1$ ,  $S := 0$ ,  $MP_C = MP_C + 1$

// skriver GRx till AR för att kunna  
uppdatera med A-värde

7.  $buss := IR$   $AR := AR + buss$ ,  $MP_C = MP_C + 1$

// adderar A-fältet, inga flaggor

8.  $buss := AR$   $GR_x := buss$ ,  $R := 1$ ,  $S := 0$ ,  $MP_C = K1(OP)$

// uppdaterar GR3 och sätter MPC till rätt instruktions-  
adress.

1. b)

00 0000011  
01 0000100  
10 0000101  
11 0001001

TSEA28

davwi279

31/5 2021

sid 2.

antar en felskrivning i tentan, att

det borde stått  $M=10$  som i deluppgift a.

c)

$n$  buss :=  $PM(ASR)$   $GRx := buss, R=0, S=0, MPC=0$

Hämtfasen tar 3 klockcykler

Adresseringsmoden tar 4 klockcykler

Själva laddningskoden tar 1 klockcykler

$\Rightarrow$  totalt 8 klockcykler

2. Den superskalära processorn kan

a) starta 4 instruktioner per

klockcykel men varje instruktion

tar 8 klockcykler att bli färdig.

Den icke superskalär processorn startar  
en instruktion i taget och de  
tar 5 klockcykler att bli färdiga.

$$\text{superskalär: } 4 \cdot \frac{1\ 000\ 000}{8} = 125\ 000 \cdot 4 = 500\ 000 \rightarrow$$

→ färdiga instruktioner per sekund

$$\text{icke superskalär: } \frac{3\ 000\ 000}{5} = 600\ 000 \text{ färdiga}$$

instruktioner per sekund

Svar Den icke-superskalär processorn har

högst genomströmningshastighet

2 b) Vid addition så fangerar 2C-algoritmen till både pos. heltal och

2C. Om sista biten är 1 i det ena talet så påverkar det max den sista biten i svaret och carry.

Vid multiplikation så påverkar en 1 som MSB många fler bitar och kan ändra värdet likså tecknet

c) För att programmet vet inte om att det kommer en branch längre fram då den måste passera en hämtfas och adresseringsfas innan den hoppar till branchinstruktionen.

d) En VLIW-processör utför långa instruktionsord uppförda av flera små instruktioner i varje klockcykel.

Instruktionsorden sätts samman vid kompilering.

En SIMD-processör utför många uträkningar samtidigt, men har bara en instruktion åt gången. Den gör alltså samma operation på många objekt samtidigt.

Subrutin mov r4, #0  
mov r3, #0

TSE428

3.

loop: mov r1, #(0x10001234 & 0xffff)

dav wi 279

movt r1, #(0x10001234 >> 16)

31/5 2021

ldr r2, [r1, r3]

sid5

add r3, r3, #1

// Summan och inkrementet till adressen börjar på 0. Värdet på adress + ink. sparas i r2 och inkrementet räknas upp.

AND r5, r2, #0x3 - tar ut 3 lsb

AND r0, r2, #0xf8 - tar ut 5 msb

AND r7, r2, #0x80 - tar ut sista biten

lsr r0, r0, #3 - skiftar ner 5 msb till att börja på bit 0

lsl r0, r0, r5 - skifta vänster så många steg som 3 lsb sa  
- är sista biten 1?

bne tecken0 - hoppa om sista biten är

ORR r0, r0, #0xff00 - teckenförläng med 1:or

b cont - hoppa till fortsättning

tecken0 ORR r0, r0, #0x0000 - teckenförläng med 0:or

cont push {lr} - r3 och r4 pushas automatiskt

bl Subrutin1

pop {lr}

add r4, r4, r0

- lägger till till summan

str r4, 0x10001232

- spara summan i minnet

cmp r2, #0x00

- är det sista värdet

beq bx lr

- avsluta om färdig

b loop

- looppa om inte

4. avbrott: push {lr, r4, r5, r6, r7, r8} · pushar  
 : ldrb r1, 0x40005000 · ladda in  
 AND r2, r1, #0x3 en byte  
 AND r2, r1, #0x3 · titta på  
 cmp r2, #0x3 två lsb  
 bne zeros · är båda 1:or?  
 AND r0, r1, #0x78 · hoppa om inte  
 lsr r0, r0, #3 · placka ut rätt bitar  
 b part2 · flytta dom till rätt  
 zeros: AND r3, r1, #0x80 plats och spara i r0  
 lsr r3, r3, #3 · kör del två  
 AND r4, r1, #0x1C · ta ut rätt bitar  
 lsr r4, r4, #2 · skifta bit 7 till rätt plats  
 ORR r0, r3, r4 · de andra bitarna  
 ORR r0, r3, r4 · skifta till rätt plats  
 · spara i r0
- part2: ldrh r5, 0x40007000 · ladda in 16 bitar  
 AND r6, r5, #0x3C0 · ta fram bit 9-6  
 lsr r6, r6, #2 · flytta dom  
 ORR r0, r0, r6 · slå ihop med r0 från  
 tidigare
- bl Subrutin2 · kör subrutin2  
 pop {lr, r4, r5, r6, r7, r8} · popa register  
 bx lr · avsluta

31/5 2021

sid 7

$$\begin{aligned}
 158_{10} &= 128 + 16 + 8 + 4 + 2 = 10011110_2 \\
 -(75_{10}) &= -(64 + 8 + 2 + 1) = -(01001011_2) = \\
 &= (10110100 + 1)_{2c} = 10110101_{2c}
 \end{aligned}$$

$$\begin{array}{r}
 \begin{array}{r}
 1 & 1 & 1 & 1 \\
 - & 1 & 0 & 0 & 1 & 1 & 1 & 0
 \end{array} \\
 + 10110101 \\
 \hline
 01010011_{2c} // 
 \end{array}$$

b)  $z=0$  ty svaret är nollskilt

$N=0$  ty MSB = 0

$C=1$  ty carry ut från MSB

$I=1$  ty olika carry ut från MSB och MSB-1

$$\begin{array}{r}
 0100 \\
 \times 1011 \\
 \hline
 0100 \\
 0000 \\
 + 0100 \\
 \hline
 0101100 //
 \end{array}$$

6. 64 byte cacheline

TSEA28

davwi279

$\Rightarrow$  6 bitar bytess.

31/5 2021

32 bitar adress

sid 8

index är 11 bitar

2 vägar

a)  $2^{\text{index}} = 2^{11}$  cachelines per väg

$\Rightarrow 64 \text{ bytes/cacheline} \cdot 2^{\text{cacheline/väg}} \cdot 2 \text{ vägar} =$

= 262 144

b) tag = total - index - bytess =

=  $32 - 11 - 6 = 15$  bitar tag

$15 \cdot 2^{11} \cdot 2 = 61\ 440$  bitar

↑      ↑  
cacheline    vägar  
per väg

6. c)

bytepos är bit 0 till  
bit 5 (6 st)

TSEA28  
darwi279  
31/5 2021

index är bit 6 till

sid 9

bit 16 (11 st)

Inkrementet på läsadresserna sker på  
bit 14 ( $0x4000 = 0b\ 0100\ 0000\ 0000\ 0000$ )

Mellan 14 och 16 är 3 bitar

$\Rightarrow 2^3$  läsningar/väg  $\Rightarrow 8$  läsningar/väg

$\Rightarrow 8 \cdot 2 = 16$  läsningar innan cachen börjar skriva  
över gamla värden //