TSEA28 Datorteknik Y, lösningar till tentamen 150818, reviderad 171025

1. a) K2 innehåller adressen i mikrokoden för starten av adresseringmode baserat på fältet M i instruktionsordet. 2 bitar ger 4 adresser, men bara 3 implementerade. Mikrominnet har 7 bitars adress (indikerat mha punkter i diagrammet över datormodellen). Enligt kommentarerna till koden är följande adresser använda:

```
Adress   Data

 0000  0000011 (3)  ; direktadressering
 0001  0000100 (4)  ; omedelbar adressering
 0010  0000110 (6)  ; indirekt adressering
 0011  -------      ; inte använd av nuvarande instruktioner
```

Detta är dock inte helt korrekt. Indirekt adressering behöver två steg för att beräkna adressen, så rad 5 ingår också. Därför är en mer korrekt beskrivning av K2:

Adress Data

```
 0000  0000011 (3)  ; direktadressering
 0001  0000100 (4)  ; omedelbar adressering
 0010  0000101 (5)  ; indirekt adressering
 0011  -------      ; inte använd av nuvarande instruktioner
```

b) Lägg till instruktionen på adress 11 i mikroprogrammet. Algoritm: Flytta värde fån GRx till AR, subtrahera värde från mikrominnet, lagra resultat i GRx.

```
Adress Mikrokod                                          Kommentar
 11    buss:=GRx, S:=0, R:=0, AR:=buss, uPC:=uPC+1 ; flytta GRx till AR
 12    buss:=1 (från mikrominnet),AR:=AR-buss      ; subtrahera 1
 13    buss:=AR, GRx:=buss, S:=0, R:=0, uPC:=0     ; Resultat till GRx
```

c) Det bitmönster som ska lagras i PM för instruktionen DECR GR1 är 0100 01 00 00000000 = $4400, då adresseringsmod och A-fältet är satta till 0.

d) K2-minnet lagrar mikrokodsadresser till mikrokod för respektive adresseringsmod. Det finns 4 adresseringsmoder så det är 4 adresser, och varje adress innehåller 7 bitar (alla register i modellen indikerar antal bitar i register med punkter). Dvs K2-minnet är 4x7 bitar.

2. a) Ett FLASH-minne är icke-flyktigt (behåller informationen även vid spänningsbortfall). Ett FLASH-minne kan skrivas om många gånger, men kräver att större segment (många byte) raderas innan nytt data skrivs.

Ett RAM är flyktigt (tappar informationen vid spänningsbortfall). Varje ord i minnet kan skrivas och ändras var för sig. RAM är dyrare per byte än FLASH.

b) Ökad sannolikhet för cacheträff fås genom att läsa data i förväg genom s k förhämtning (prefetch). Svarstid kan minskas genom att cachelinen inte fylls linjärt, utan att data läses först till t ex sista halvan av cacheline och sist fylls första halvan av cacheline. Båda dessa metoder används av cache i lab 5.

c) I-flaggorna beskriver vilken avbrottsnivå processorn körs i (tolkat som ett 3-bitars binärt tal). I-flaggorna kan användas för att förhindra avbrott genom att sätta ett högre värde på dom. Avbrott av lägre prioritet kan då inte avbryta processorn.

d) Virtuellt minne tillåter en logisk adress som processorn använder vara placerad på en annan fysisk adress. Det kan implementeras mha ett minne som använder några av de mest signifikanta bitarna som adress, och som data leverar den fysiska adressens mest signifikanta bitar. Denna funktion implementeras ofta i en MMU (Memory Management Unit).

e) En direktmappad cache har bara en möjlig cacheline som kan lagra respektive adress. Den associativa cachen kan ha flera olika möjliga cachelines för en specifik adress. En direktmappad cache är enklare att implementera, då den associativa cachen måste jämföra alla möjliga cachelines med aktuell adress, medan den direktmappade endast behöver jämföra en.

3. Subrutinen behöver för varje nibble (4 bitar) i D0 upprepa samma översättning till ASCII. Skapa därför en extra subrutin som översätter de minst signifikanta 4 bitarna i D0 till ett ASCII-tecken och skriver ut mha subrutinen på adress $4010. Anropa denna 4 gånger där D0 skiftats åt höger olika mycket. Inget krav är givet om att register inte får förändras, så D1 och D0 har ändrat värde när subrutinen returnerar.

```
Printword:  move.w  D0,D1       ; spara D0 i D1
            lsr.w   #12,D0      ; Skifta D0 så bit 12 till 15 hamnar som bit 0 till 3
            jsr     printhex    ; Skriv ut  som ASCII
            move.w  D1,D0       ; Återställ D0
            lsr.w   #8,D0       ; Skifta D0 så bit 8 till 11 hamnar som bit 0 till 3
            jsr     printhex
            move.w  D1,D0       ; Återställ D0
            lsr.w   #4,D0       ; Skifta D0 så bit 4 till 7 hamnar som bit 0 till 3
            jsr     printhex
            move.w  D1,D0       ; Sista bitarna redan på rätt plats
            jsr     printhex
            rts

printhex:   and.b   #$0F,D0     ; Sätt bit 7 till 4 till värde 0
            add.b   #$30,D0     ; Lägg till start för ASCII '0'
            cmp.b   #$3A,D0     ; Kontrollera om A-F
            blt     printchar   ; Hoppa om 0-9
            add.b   #7,D0       ; $41 - $3A = 7, dvs lägg till 7 för att få 'A'-'F'
printchar:  jsr     $4010       ; skriv ut ASCII-tecknet i D0
            rts
```

TSEA28 Datorteknik Y, lösningar till tentamen 150818, reviderad 171025

4. Avbrottsrutin kräver att RTE används istället för RTS. Dessutom måste alla register återställas innan retur från avbrott. Algoritm: Läs in byte från port A, skifta 8 bitar, läs in 8 bitar, rotera 1 steg.

```
Avbrott: move.w  D0,-(A7)      ; spara undan D0
         move.b  $10080,D0     ; läs data från port A
         lsl.w   #8,D0         ; Flytta data till bit 8-15
         move.b  $10082,D0     ; läs 8 bitar från port B
         rol.w   #1,D0         ; rotera alla bitar 1 steg åt vänster
         move.b  D0,$10082     ; nytt värde till port B
         lsr.w   #8,D0         ; Flytta bit 8-15 till 0-7
         move.b  D0,$10080     ; nytt värde till port A
         move.w  (A7)+,D0      ; återställ D0
         rte                   ; återhopp från avbrott, återställer även SR
```

5. a) Addition: \$BA+\$6C = \$126. Resulterande 8 bitar är \$26 = 00100110_2, och den 9:e biten hamnar i carryflaggan (C=1). Z-flaggan blir 0 eftersom resultatet inte är 0. V-flaggan blir 0 eftersom resultatet sett som tvåkomplement inte ger overflow (\$BA$_{2C}$ = -\$46, \$6C positivt, summa fortfarande inom talområdet +/- \$7F)

b) $-12_{10} = -00001100_2 = (11110011+1)_{2C} = 11110100_{2C}$. Dvs -12 = -(8+4). Bitvis invertering plus LSB ger tvåkomplementsformen.

6. a) Denna cache har 1024 cacheline per väg (32768/16 = 2048 cachelines, 2 vägar => 2048/2 = 1024 cachelines per väg) delar upp adressen i 4 bitar för position i cacheline, 10 bitar för index i varje väg, och resten av adressen blir tag. Följande sekvens av tag, index och position blir resultatet.

Adress	tag	index	pos	väg	träff/miss
\$004004	\$001	\$000	4	0	miss
\$004000	\$001	\$000	0	0	träff
\$004040	\$001	\$004	0	0	miss
\$004044	\$001	\$004	4	0	träff
\$008080	\$002	\$008	0	0	miss
\$008008	\$002	\$000	8	1	miss (väg 0 redan full)
\$068000	\$01A	\$000	0	0	miss (väg 1 läst senast)
\$068008	\$01A	\$000	8	0	träff
\$006502	\$001	\$250	2	0	miss
\$068020	\$01A	\$002	0	0	miss
\$056000	\$015	\$200	0	0	miss
\$056002	\$015	\$200	2	0	träff
\$004004	\$001	\$000	4	1	miss (väg 0 senast läst)
\$008080	\$002	\$008	0	0	träff

$068000	$01A	$000	0	0	träff
$056000	$015	$200	0	0	träff

b) En dubbelt så stor cache med samma längd på cacheline och samma antal vägar ger dubbelt så många cachelines. Index ökar då med 1 bit, och tag minskar med 1 bit. Motsvarande sekvens blir då:

Adress	tag	index	pos	väg	träff/miss
$004004	$000	$400	4	0	miss
$004000	$000	$400	0	0	träff
$004040	$000	$404	0	0	miss
$004044	$000	$404	0	0	träff
$008080	$001	$008	0	0	miss
$008008	$001	$000	8	0	miss
$068000	$00D	$000	0	1	miss
$068008	$00D	$000	8	1	träff
$006502	$000	$650	2	0	miss
$068020	$00D	$002	0	0	miss
$056000	$00A	$600	0	0	miss
$056002	$00A	$600	2	0	träff
$004004	$000	$400	4	0	träff
$008080	$001	$008	0	0	träff
$068000	$00D	$000	0	1	träff
$056000	$00A	$600	0	0	träff

Dvs det blir 8 missar istället för 9.

7. a) Stackinformation:

$6FF4 $1020	returadress efter subrutinanrop på adress $101E
$6FF8 $100E	returadress efter subrutinanrop på adress $100C
$6FFC $xxxx	returadress efter subrutinanrop till adress $1000
$7000 $xxxx	Initialt värde innan subrutinanrop till $1000

Stacken pekar alltid på senaste objektet som lagts på stacken, i detta fall $6FF4 när instruktionen på adress $102C utförs.

b) Subrutinen lägger ihop de 16-bitars tal A0 pekar på (dom ligger efter varandra) och D1/$10 är antal tal att addera. Subrutinen $1028 kontrollerar om nästa tal har värdet $1234 och tar i så fall bort sin egen returadress och tvingar istället fram en retur från subrutinen $101C. Subrutinen $101C anropas 2 gånger, och 1:a gången adderas 1+2 till D0 innan värdet $1234 hittas, sedan adderas $10+$20 till D0 utan att hitta värdet $1234. Slutresultat blir därför D0=$1+$2+$10+$20=$33.