TSEA28 Datorteknik Y, lösningar till tentamen 160531 reviderad 160808

1. a) Det finns inga adresser givna för var i mikroprogrammet respektive instruktion startar. Därför går det inte att ange hela minnets innehåll i binär form. Det som går att ange är vilka adresser som finns (4 bitar opcode ger 16 olika adresser) och i vilka adresser i K1 som respektive startadress till instruktionerna skulle ligga. Pga detta ändrades kravet för betyg 3 från 21 till 20 poäng.

```
Adress   Data                          Adress   Data

 0000  -------  ; inte använd           1000  -------  ; inte använd
 0001  -------  ; inte använd           1001  -------  ; inte använd
 0010  -------  ; inte använd           1010  ADD      ; Adress till ADD
 0011  -------  ; inte använd           1011  -------  ; inte använd
 0100  JSR      ; Adress till JSR       1100  -------  ; inte använd
 0101  JNC      ; Adress till JNC       1101  -------  ; inte använd
 0110  -------  ; inte använd           1110  -------  ; inte använd
 0111  STORE    ; Adress till STORE     1111  -------  ; inte använd
```

b) Hämtfasen ökar PC med 1, så detta steg ska inte göras av mikrokoden för JSR. Algoritm: Minska GR3 med 1 (lägg till $FFFF), Flytta GR3 till ASR, flytta PC till PM, flytta IR till PC. Antag rutinen startar på adress n:

```
Adress Mikrokod                                Kommentar
  n    buss:=GRx, R:=1, AR:=buss, uPC:=uPC+1 ; flytta GR3 till AR
  n+1  buss:=uM, uM=$FFFF, AR=AR+buss,       ; Addera -1 utan att ändra flaggor
       uPC:=uPC+1
  n+2  buss:=AR, R:=1, GRx:=buss, uPC:=uPC+1 ; Spara nya stackpekare i GR3
  n+3  buss:=AR, ASR:=buss, uPC:=uPC+1       ; Peka på stacken i minnet
  n+4  buss:=PC, PM:=buss, uPC:=uPC+1        ; Spara återhoppsadressen
  n+5  buss:=IR, PC:=buss, uPC:=0            ; nästa instruktion på adress ADR,
                                             ; börja nästa hämtfas
```

c) Maskininstruktionen JSR $12 har formen 0100--0000010010 där -- är två bitar som kan väljas fritt, och $12 = 00010010 är adressen som subrutinen ska hoppa till.

2. a) Ett EEPROM (Electrically Erasable Programmable Read Only Memory) är ett läsminne som kan raderas elektriskt. Det är möjligt att radera och skriva om valbar adress. Tyvärr anger kursboken att hela EEPROM raderas varje gång, vilket inte stämmer. Ett FLASH är också ett EEPROM, men med begränsningen att större områden i minnet måste raderas samtidigt (många adresser på en gång) men inte hela minnet. FLASH är snabbare att skriva till, men långsammare att läsa. Programmeringsutrustning skiljer inte (moderna chip använder samma typ av gränssnitt).

b) En styrkonflikt betyder att nästa instruktion att utföra inte kan hämtas i förväg (nästa instruktion ligger inte direkt efter nuvarande). Detta betyder att det bildas bubblor i pipeline i en superskalär processor (processorn väntar på att få reda på var nästa adress ska hämtas ifrån och startar då inga nya instruktioner).

Exempel på hur dessa konflikter kan undvikas är branch prediction (förutse var nästa instruktion finns) och delay slot (tillåta instruktioner efter hopp att utföras ändå).

c) Bakgrundsprogrammet kan stänga av avbrott mha I2 I1 I0 i SR-registret. Sätts dessa till 111 så förhindras avbrott på avbrottsnivå 1-6. Ett mindre lämpligt alternativ är att stänga av avbrottsgenerering i I/O-enheterna.

d) Relativa hopp gör det möjligt att lägga assemblerad kod var som helst i minnet. Dessutom sparar relativa hopp utrymme i programminnet då instruktionen bara behöver innehålla hur lång hoppet är, och om bara korta hopp görs kan en kortare version av instruktionen användas (och därmed lite snabbare exekvering). Ett absolut hopp kräver att hela adressen lagras i instruktionen och det går inte att flytta den koden till en annan plats i minnet utan att assemblera om.

e) Ett subrutinanrop behöver inte använda stacken. T ex ARM använder istället ett internt register (LR) som lagrar aktuell återhoppsadress, och det är sedan upp till programmeraren att tömma/flytta LR om ytterligare subrutinanrop ska göras. Dock måste återhoppsadressen hanteras på något sätt, eftersom återhoppet vid subrutinens slut måste bero på varifrån den anropades.

3. Inga krav är givna om att spara registervärden, så alla register får användas utan att återställas. Algoritm: Nollställ D0, Loop: hämta adress från minnet på adress som A0 pekar på till A1, räkna upp A0, kontrollera om A1 är noll, om inte hämta från minnet på adress som A1 pekar på och lägg till D0 samt hoppa tillbaks till loop.

```
Subrutin:   move.l   #0,D0       ; nollställ D0
Loop:       move.l   (A0)+,A1    ; Hämta adress ur listan samt öka A0 med 4
            cmp.l    #0,A1       ; Är den hämtade adressen =0?
            beq      End         ;   Ja, hoppa ur loop
            add.l    (A1),D0     ; Hämta värden från adressen och summera
            bra      Loop        ; Nästa varv i loop

End:        rts                  ; Klar, hoppa tillbaks
```

4. a) 8-bitars addition. I binär form (bitposition 0 längst till höger):

```
 1
  10010001
 +10001000
 ---------
  00011001
```

Detta ger 8-bitars resultat i D1 = 00l011001 = $19. Dessutom blir C=1 (carry från position 7 till position 8), Z=0 (resultat inte =0) och N=0 (bitposition 7 i svaret är 0). För den sista flaggan (V) kan noteras att om tvåkomplementsrepresentation antas är indata båda negativa (bitposition 7 = 1) och resultatet vid addition borde därför vara negativt. Eftersom resultatet är positivt (N=0) sätts alltså V=1. Alternativt sätt att se detta är att notera att carry från bitposition 6 till bitposition 7 är 0 (ingen carry) medan carry från bitposition 7 till bitposition 8 är 1 (C=1), och därför sätts V=1.

b) Uppgiften anger inte om värdena i D0 och D1 antas vara positiva heltal eller i 2-komplementsform. Resultatet blir olika beroende på detta antagande.

Om D0 och D1 antas vara positiva heltal ska hopp göras om subtraktionen ger ett lån, dvs C=1, eller om Z=1. Det räcker inte med att titta på N-flaggan eftersom om D0 är väldigt stort och D1=1 blir N-flaggan fortfarande satt trots att D0-D1 inte gav lån.

Om D0 och D1 antas vara i 2-komplementsform ska hopp göras om Z=1 eller beroende på värdet hos N och V flaggorna. Om N=1 och V=0 alternativt om N=0 och V=1 ska hopp göras, därför resultatet efter subtraktionen blev negativt. Det räcker inte att bara använda N=1 eftersom subtraktionen kan ha gett ett negativt tal som inte går plats i resultatet (därför även V kontrolleras).

c) Det finns två olika multiplikationsinstruktioner då multiplikation med positiva heltal ger ett annat svar än multiplikation med 2-komplementstal. För positiva heltal skapas alltid partialprodukter med positivt tecken (alla vikter i talrepresentationen är positiva) medan multiplikation av 2-komplementstal måste ta hänsyn till att teckenbiten har negativ vikt, och därför behöver addition med teckenförlängning samt subtraktion implementeras i samband med hanteringen av teckenbitens bidrag.

5. Avbrottsrutin ska alltid återställa registervärden efter slutförd exekvering. Algoritm: läs värdet på adress $5004 och om värdet är 0 hoppa till retur från avbrottet. Annars läs adress från $5000. Om adress mindre än $4400 (dvs $4000+4*256) ska återhoppsadressen hämtas från stacken (utan att ändra stackpekaren) och placeras i bufferten. Buffertadressen räknas upp med 4 och sparas i $5000. Därefter återställ register och returnera från avbrott.

```
Avbrott:  cmp     #0,$5004    ; Ska adress sparas?
          beq     end         ; värdet är 0, hoppa ur
          move.l  A0,-(A7)    ; Spara A0 för senare användning
          move.l  $5000,A0    ; Hämta adress i buffert
          cmp.l   #$43FF,A0
          bgt     full        ; Om adress > $43FF är buffert full
          move.l  A1,-(A7)    ; Spara A1 för senare användning
          move.l  A7,A1       ; Kopiera aktuell stackposition
          sub.l   #10,A1      ; Stack innehåller just nu A1, A0, SR, PC, dvs
                              ; PC 10 byte längre ned i minnet än aktuellt stackpekare
          move.l  A1,(A0)+    ; Spara PC från stacken i buffert, öka pekare i buffert
          move.l  A0,$5000
          move.l  (A7)+,A1    ; återställ A1
Full:     move.l  (A7)+,A0    ; återställ A0
End:      rte                 ; återhopp från avbrott
```

En viktig sak att undvika: får inte ändra A7 ifall andra avbrott sker medan detta avbrott hanteras. Om A7 skulle ändras kan saker på stacken skrivas över.

6. a) 1024 byte minne, cacheline är 8 bytes lång. 1024/8=128 cachelines i cache. 128=2^7, dvs det behövs 7 bitar för att välja rätt cacheline och 3 bitar för att välja rätt byte i

cacheline. Totalt används 10 bitar för val av cacheline och position i cacheline. Övriga bitar blir tag, i detta fall 32-10=22 bitar.

b) Skriv ned alla läsningar som sker under programmets exekvering

Adress	tag	index	pos	Instruktion	Träff/Miss
$00001000	$000004	$00	0	clr.l D0	M
$00001002	$000004	$00	2	move.l #$2000,A0	T
$00001004	$000004	$00	4	resten av instr.	T
$00001008	$000004	$01	0	move.l #3,D1	M
$0000100a	$000004	$01	2	add.l (A0),D0	T
$00002000	$000008	$00	0	argument (A0)	M
$0000100c	$000004	$01	4	add.l #4,A0	T
$0000100e	$000004	$01	6	sub.l #1,D1	T
$00001010	$000004	$02	0	bne loop	M
$0000100a	$000004	$01	2	add.l (A0),D0	T
$00002004	$000008	$00	4	argument (A0)	T
$0000100c	$000004	$01	4	add.l #4,A0	T
$0000100e	$000004	$01	6	sub.l #1,D1	T
$00001010	$000004	$02	0	bne loop	T
$0000100a	$000004	$01	2	add.l (A0),D0	T
$00002008	$000008	$01	0	argument (A0)	M
$0000100c	$000004	$01	4	add.l #4,A0	M
$0000100e	$000004	$01	6	sub.l #1,D1	T
$00001010	$000004	$02	0	bne loop	T
$00001012	$000004	$02	2	rts	T
$xxxxxxxx	$xxxxxx	$xx	x	återhoppsadress	T (mest sannolikt)

Totalt blir det 6 missar och 15 träffar. Total tid = 6*10+15*1=75 klockcykler.