TSEA28 Datorteknik Y, lösningar till tentamen 150601 reviderad 160530

1. Förtydligande: varje värde i tabellen anger ett ascii-tecken. Värdet 0 kallas för NULL, 2 kallas STX, och texten är (med start på adress 40): NULL, “ Hej!”, NULL, NULL, STX. Pekaren i exemplet pekar på H, och därför är det “Hej!”+NULL = 5 tecken.

a) K1 och K2 används som case-satser i mikrokoden. K1 används till för att hoppa till rätt kod beroende på OP-fältet i instruktionen lagrad i IR-registret. K2 används för att välja rätt kod beroende på vald adresseringsmod (M-fältet i IR-registret).

b) Antar att instruktionshämtning och adresseringsmod redan gjorts. I detta exempel används PC för att peka på nästa adress till strängen. Måste då först spara undan orginalvärde i PC till HR, och sedan återställa PC på slutet. Algoritm: leta efter 0 i PM, öka PC allt eftersom, och när 0 hittats i PM beräkna GRx = PC – GRx, återställl sedan PC. Andra (enklare att förstå) algoritmer inkluderar att använda HR som adresspekare och sätta Grx initialt till 0 och sedan räkna upp HR och GR för varje tecken.

```
          HR:=PC ; uPC=uPC+1                    ;; spara undan värdet i PC
          PC:=GRx ; R:=0 ; S:=0 ; uPC=uPC+1     ;; sätt PC till värdet i register GRx
          ASR:=PC ; AR:=0 ; uPC:=startloop      ;; Läs adress GRx pekade på

loop:     ASR:=PC ; AR:=0 ; uPC:= klar om Z = 1  ;; Sätt minnesadr., hoppa ur om tkn 0
startloop: AR:=AR OR PM ; PC:=PC + 1; uPC := loop ;; kontrollera om tkn=0, öka räknare

klar:     AR:=PC ; uPC:=uPC + 1                ;; Hittat nollan, räkna hur många tecken
          AR:=AR – GRx ; S:=0 ; R:=0 ; uPC:=uPC + 1     ;; antal = slutaddr - startaddr
          GRx:=AR ; S:=0 ; R:=0 ; uPC:=uPC + 1          ;; Svaret till register GRx
          PC:= HR ; uPC:= 0                  ;; Återställ PC, hämta nästa instruktion.
```

c) Om det inte finns något värde 0 i PM kommer instruktionen aldrig slutföras. När adressen når maxvärdet och ökas med 1 kommer adressen bli 0, så hela minnet läses om och om igen.

2. a) En 4-stegs RISC processor startar 1 instruktion per klockcykel, men behöver 4 klockcykler för att slutföra instruktionen. Den hinner med 2 Giga operationer per sekund. En 4-vägs superskalär processor startar 4 instruktioner per klockcykel. Den kommer då hinna med 4 Giga instruktioner per sekund. Dvs den superskalära processorn kan exekvera flest antal instruktioner per sekund. Allt detta under förutsättning att inga problem med hopp, databeroenden etc uppkommer.

b) Alla operationer i instruktionsordet i en VLIW processor exekveras alltid samtidigt och alla databeroenden måste vara lösta när koden kompilerats. En superskalär processor försöker starta flera instruktioner samtidigt, men tar hänsyn till om en delinstruktion behöver resultat från andra delinstruktioner mha enheten dispatch.

c) En datakonflikt (data hazard) i en RISC processor betyder att resultat från föregående instruktion inte finns tillgänglig som indata för nästa instruktion. Detta kan lösas genom att resultat från tidigare instruktion skickas direkt till ALU:n istället för att först placeras i registerfilen.

d) SRAM = Statiskt RAM, DRAM = Dynamisk RAM. Statiskt RAM används för mindre minnen som ska vara väldigt snabba (t ex cache-minnen) och som får kosta lite mer. Dynamiska RAM är billigare, låńgsammare och kan användas till t ex primärminne. Dynamiska RAM kräver dessutom att alla minnesceller läses med en viss frekvens (refresh) då dom bygger på

en teknik med laddning lagrad i kondenatorer för varje bit, medan SRAM använder återkopplad logik (D-vippor).

f) Några skäl till att en MMU (Memory Management Unit) behövs i en modern dator är 1. Adressöversättning: Möjliggöra för flera program att köras samtidigt utan att dom behöver kompileras om för att hamna på olika minnesadresser, dvs översätter processorns adress till andra adresser i fysiska minnet (även för defragmentering). 2. Minnesskydd: Skydda ett program från andra program (både läsning och skrivning). 3. Tillåta virtuellt minne där programmen tror att dom har mer minne tillgängligt än vad som finns fysiskt i maskinen. Försök att läsa/skriva en adress som inte har en motsvarande fysisk adress skapar ett avbrott så processorn kan allokera fysiskt minne till adressen och sedan låta datorn fortsätta instruktionen.

3. a) Ett 8-bitars 2-komplmementtal kan representera -128 till +127. 227 är större än 127 och därför räcker inte 8 bitar för att lagra variabeln.

b) $A/2 = 11010100_{2C} >> 1 = 11101010_{2C}$. Viktigt här att kopiera teckenbiten till vänster så det fortfarande är ett negativt tvåkomplementstal. För additionen måste A/2 göras om till samma längd (12 bitar) som B. Görs genom teckenförlängning (kopiering av teckenbit).

$$Y = A/2+B = 111111101010_{2C} + 001101011000_{2C} = 001101000010_{2C}$$

Kontroll: A = -101011+1 = -44 => A/2 = -22, B = 512+256+64+16+8=856. Y = 856-22=834=001101000010_{2C}

c) 5-bitars tal kan representera -16 till +15. Minsta produkten är -16*15=-240. Maximala produkten är resultatet av (-16)*(-16)=+256. Största positiva värdet för 9 bitars tal är +255 och därför behövs 10 bitar för att representera alla produkten.

d) Man behöver inte ta hänsyn till att man adderar två-komplementstal (antar båda termerna har lika många bitar). I båda fall (positiva helt respektive tvåkomplement) måste man ta hänsyn till att om resultatet går utanför talområdet så tolkas resultatet fel.

4 a) Antag ingen hantering av avbrottsflaggor behövs. Antar registret är 16 bitar långt

```
Intr: move.l D0,-(A7),  -- Spara D0 på stacken (avbrottsrutin!)
      move.l D1,-(A7)   -- Spara D1 på stacken
      move.w $10080,D0  -- Läs data från register, D0 ska vara bit 2
      move.w D0,D1      -- D1 ska innehålla alla andra bitar
      and.w  #$0004,D0  -- nollställ alla bitar utom bit 2
      lsl.w  #2,D0      -- flytta bit 2 till bit 4 positionen
      and.w  #$FFEF,D1  -- nollställ bit 4
      or.w   D1,D0      -- lägg till bit
      move.w D0,$10080  -- skicka tillbaks till registret
      move.l (A7)+,D1   -- återställ D1 från stacken
      move.l (A7)+,D0   -- återställ D0 från stacken
      RTE               -- return från exception
```

b) Linus får fel värden ibland därför att kodraderna flyttar stackpekare till inuti stacken (lägger till 8 till A7). Om avbrott sker innan A7 återställs kommer toppen av stacken ändras när avbrottet placerar återhoppsadress och statusregister på stacken som skriver över data för D3 och

TSEA28 Datorteknik Y, lösningar till tentamen 150601 reviderad 160530

D2. Antar här att Linus program körs i supervisor-mode (får inte samma problem om programmet körs i user mode pga annan stack för avbrott).

c) RTS hämtar 4 byte från stacken och placerar i PC. RTE hämtar 6 byte från stacken, där 4 placeras i PC och 2 placeras i statusregistret.

5. Tabellen över minnesinnehållet i exemplet var lite komprimerat, de extra 4 inledande nollorna på alla värden är inte utskrivna. Däremot stämmer alla adresser etc.

Subrutinen behöver spara undan använda register på stacken. Därefter en loop som lägger till värdet som A0 pekar på, sedan ökar A0 med 4, hämtar värdet till A0, och kontrollerar om det inte är noll. Om A0 = 0 ska D0 innehålla totalsumman, A0 återställas, och subrutinen returnera. Antag A0 aldrig är 0 vid anrop till subrutin.

```
AddList:    move.l A0,-(A7)   ; Spara initialt värde på A0
            clr.l  D0         ; startvärde för summa = 0
Loop:       add.l  (A0)+,D0   ; Lägg till värde i lista till summa, öka adress
            move.l (A0),A0    ; Hämta adress till nästa element i listan
            cmp.l  #0,A0      ; Se om pekar på 0 (ska avsluta i så fall)
            bne    Loop       ; Läs nästa element om pekare inte noll
            move.l (A7)+,A0   ; Återställ A0
            rts               ; klar med subrutin
```

6. a) 32-byte cachelin => totalt 65536/32 = 2048 cachelines i cacheminnet. Dessa delas upp på 4 delar (4-vägars cache). Varje väg har därför 2048/4=512 olika cachelines i cachen.

b) Av adressen på 32 bitar används de 5 minst signifikanta bitarna för att bestämma position i cacheline (32 olika). Därefter används 9 bitar för att bestämma index (512 olika). Totalt är det 14 bitar som anger position i cache, och resterande 18 bitar i varje adress blir tag-värde i cachen. När steglängden är 4096 = $1000 kommer dom två högsta bitarna i index öka vid varje läsning. Dvs antag start på adress 0:

Adress (hexadecimalt)	**Tag (bit 31 till och med bit 14)**	**Index (bit 13 till och med 5)**	**Position i cacheline (bit 4 till och med 0)**
$00000000	000000000000000000	000000000	00000
$00001000	000000000000000000	010000000	00000
$00002000	000000000000000000	100000000	00000
$00003000	000000000000000000	110000000	00000
$00004000	000000000000000001	000000000	00000
$00005000	000000000000000001	010000000	00000

Var 4:e läsning i sekvensen ger samma index som tidigare. Det betyder att efter 8 läsningar har samma index dykt upp två gånger, och två av de 4:a vägarna i cachen har använts. Efter 4*4=16 läsningar kommer då alla vägar i cachen har fyllts. K får därför vara maximalt 16.

c) Adressen som anger start av de 16 bytes som läses är slumpmässigt placerad. Därför fås en cachemiss när första byte ska läsas (osannolikt att adressen redan finns inläst då n är slumpmässigt). Dessutom kommer i hälften av fallen startaddressen vara på en adress som ligger i andra halvan av cachelinen. Därmed fås två cachemissar när inte alla 16 bytes får plats i samma cacheline. I andra halvan av fallen startar läsningen på en adress som ligger i första halvan av en cacheline, och då får alla 16 bytes plats i en och samma cacheline. Därför fås bara 1 cachemiss i dom fallen.