TSEA28 Datorteknik Y, lösningar till tentamen 151020 reviderad 161010

1. a) Uppgiften anger att de värden som jämförs i GRx och PM beskriver positiva binära tal. Operationen A-B sätter då flaggan C=1 om A < B (carry kallas då ofta för borrow). Om A > B eller A=B fås istället C=0. C-flaggan får därför rätt värde (C=1 vid byte) om beräkningen GRx-PM utförs.

Ett problem (egentligen en miss vid definitionen av uppgiften) är att Z flaggan sätt till 1 om GRx=PM, och Z-flaggan ska enligt uppgiften istället vara 0 om talen är lika. Därför behöver fallet med GRx=PM hanteras testas för och flaggan ändras. Z-flaggan kan ändras (utan att påverka C-flaggan) om operationerna AR:=0 (sätta Z=1) respektive AR:=AR OR buss (nollställa Z) används (lämpligen med konstant från uM på bussen).

Lägg till instruktionen på adress 11 i mikroprogrammet. Notera att ASR redan innehåller adressen från instruktionen enligt beskrivning av nuvarande innehåll. Algoritm: Flytta värde från GRx till AR, subtrahera värde från PM, sätt Z=0 (beräkna AR:=AR OR 1). Om C=0 ta nästa instruktion, annars läs PM och placera i HR, skriv GRx till PM, flytta HR till GRx, ta nästa instruktion.

```
Adress Mikrokod                                      Kommentar
   11  buss:=GRx, S:=0, R:=0, AR:=buss,              ; kopiera GRx till AR
       uPC:=uPC+1
   12  buss:=PM, AR:=AR-buss, uPC:=uPC+1             ; subtrahera PM

   13  buss:=1 (uM=1), AR:=AR OR buss, uPC:=uPC+1 ; nollställ Z-flaggan

   14  buss:=PM, HR:=buss, uPC:=0 om C=0             ; PM->HR, C=0 om GRx >= PM
                                                     ; nästa instruktion om C=0
   15  buss:=GRx, PM:=buss, S:=0, R:=0,              ; kopiera GRx till PM
       uPC:=uPC+1                                    ;
   16  buss:=HR, Grx:=buss, S:=0, R:=0, uPC:=0       ; kopiera HR till GRx
```

b) K1 innehåller adresserna för starten av mikrokod för respektive opcode. 4 bitars adress där nuvarande instruktionsuppsättning innehåller 4 instruktioner.

```
Adress   Data                          Adress   Data

 0000  ------- (-)  ; inte använd       1000  ------- (-)  ; inte använd
 0001  0001000 (8)  ; JNE               1001  ------- (-)  ; inte använd
 0010  ------- (-)  ; inte använd       1010  ------- (-)  ; inte använd
 0011  0001010 (10) ; LOAD              1011  ------- (-)  ; inte använd
 0100  0000111 (7)  ; STORE             1100  ------- (-)  ; inte använd
 0101  ------- (-)  ; inte använd       1101  ------- (-)  ; inte använd
 0110  ------- (-)  ; inte använd       1110  ------- (-)  ; inte använd
 0111  0001011 (11) ; CMPSWP            1111  ------- (-)  ; inte använd
```

2. a) ROM = Read Only Memory. Minne som programmeraren inte fritt kan skriva till. Exempel: maskprogrammerat ROM: fast innehåll som bestäms vid tillverkning av ROM. EPROM Erasable Programmable ROM: Raderbart programmerbart ROM, kan programmeras om med speciell utrustning. FLASH: Elektriskt raderbart minne.

b) En SIMD (Single Instruktion Multiple Data) processor utför en instruktion på flera dataströmmar parallellt.

c) Bland fördelarna med PCI-express märks högre hastighet, möjlighet att kommunicera på längre avstånd, färre trådar i gränssnittet, standardiserat (processoroberoende) gränssnitt.

d) En big-endian maskin (t ex M68000 i tutorsystemet) skriver ord bestående av flera byte (word och long som upptar flera adresser i minnet) med mest signifikant byten på lägst adress i minnet, medan en little-endian maskin (t ex intel x86) skriver den minst signifikanta byten på lägst adress i minnet.

e) Stacken i MC68000 påverkar även av avbrott som lägger aktuell PC och SR på stacken utan att en assemblerinstruktion angett detta.

3. Avbrottsrutin kräver att alla register återställs när återhopp sker. Information om vilken bit som skickats ut måste då placeras i den tillgängliga minnearean $3010-$3020. Antag $3010 är en biträknare som anger hur många bitar som skickats iväg så här lång. Antag den initieras till 0. Tolka räknaren som: 0 – ingen bit har skickats, 1-32 motsvarande bitposition skickades ut förra avbrottet.

Avbrottets funktion: Kontrollera om data skickas (dvs att $3004 = 1), om inte hoppa ur.

Om data skickas kontrollera om biträknare = 32. Om så sätt $3004 = 0 och port A bit 0 = 0, biträknare = 0, och hoppa ur.

Läs värdet på adress $3000 och skifta fram rätt bit. Placera bit på bit 0 i PIAA-porten. Räkna upp biträknaren.

```
Avbrott:  cmp.b   #0,$3004    ; Aktiv sändning?
          bne     aktiv       ; utför koll av vilken bit
          tst     $10080      ; nollställ avbrottsflaggan
          rte
aktiv:    move.w  D0,-(A7)    ; spara D0 på stack
          move.l  D1,-(A7)    ; spara D1 på stack
          move.l  $3000,D1    ; hämta data att skicka
          move.b  $3010,D0    ; hämta antal bitar som redan skickats
          cmp.b   #32,D0      ; kontrollera om alla bitar skickats redan
          bpl     sistabit    ; sista biten sänd
          lsr.l   D0,D1       ; skifta D1 höger D0 steg (rätt bit till LSB positionen)
          and.b   #1,D1       ; nollställ alla bitar utom LSB
          move.b  $10080,D0   ; hämta nuvarande port A värde
          and.b   #$FE,D0     ; nollställ bara bit 0
          or.b    D0,D1       ; placera ny bit på position 0 i data från port A
          move.b  D1,$10080   ; skriv till port A
          add.b   #1,$3010    ; öka biträknaren
          move.l  (A7)+,D1    ; återställ D1 från stack
          move.w  (A7)+,D0    ; återställ D0 från stack
          rte
```

```
sistabit:   move.b  #0,$3010    ; nollställ biträknaren
            move.b  #0,$3004    ; indikera klar med sändning
            and.b   #$FE,$10080 ; nollställ bit 0 i port A.
            move.l  (A7)+,D1    ; återställ D1 från stack
            move.w  (A7)+,D0    ; återställ D0 från stack
            rte
```

4. Eftersom inga register får vara ändrade efter subrutinanropet behöver dom register som ändras sparas först på stacken. Använd D0 som varvräknare och D1 för att spara F(n), D2 för F(n-1) och D3 för att spara F(n-2).

```
Fibonacci:  move.w  D0,-(A7)    ; spara D0 på stack
            move.w  D1,-(A7)    ; spara D1 på stack
            move.w  D2,-(A7)    ; spara D2 på stack
            move.w  D3,-(A7)    ; spara D3 på stack
            move.l  A0,-(A7)    ; spara A0 på stack
            move.w  #0,(A0)+    ; F(0)
            sub.l   #1,D0       ; var det ett värde som behövdes?
            beq     klar        ; ja, avsluta
            move.w  #1,(A0)+    ; F(1)
            sub.l   #1,D0       ; var det ett värde som behövdes?
            beq     klar        ; ja, avsluta
            move.w  #1,D2       ; Initiera för loop, D2 = F(n-1), D3=F(n-2)
            move.w  #0,D3
loop:       move.w  D3,D1       ; Beräkna nästa F(n) (förstör äldsta värdet, F(n-2))
            add.w   D2,D1
            move.w  D1,(A0)+    ; spara nästa värde i vektor
            sub.l   #1,D0       ; var det sista värdet?
            beq     klar        ; ja, avsluta
            move.w  D2,D3       ; nej, skifta värden ett steg
            move.w  D1,D2
            bra     loop        ; nästa värde
klar:       move.l  (A7)+,A0    ; återställ A0 från stack
            move.w  (A7)+,D3    ; återställ D3 från stack
            move.w  (A7)+,D2    ; återställ D2 från stack
            move.w  (A7)+,D1    ; återställ D1 från stack
            move.w  (A7)+,D0    ; återställ D0 från stack
            rts
```

TSEA28 Datorteknik Y, lösningar till tentamen 151020 reviderad 161010

5. a) Talet 7 beskrivet i binär tvåkomplementsform är 0111. Det behövs alltså 4 bitar. Den första 0:an krävs för att indikera att talet är positivt eftersom 2-komplementsrepresentationens MSB beskriver tecken.

b) Addition av två N-bitars tal ger N+1 bitars resultat. Addition av två N+1-bitars tal ger N+2 bitars resultat. Addition av två N+2-bitars resultat ger N+3 bitars resultat. Addition av åtta tal kan ses som 4 additioner av N-bitars tal vars summor (N+1 bitar var) sedan adderas parvis och skapar 2 stycken N+2 bitars värden. När dessa två sedan adderas fås slutsumman som behöver N+3 bitar.

c) Instruktionerna visar att D0-värdet skiftas vänster 2 steg, dvs multipliceras med 4 följt av en addition med orginalvärdet. Utryckt matematiskt: D0resultat = 4D0+D0=5D0. Dvs rutinen muliplicerar D0 med 5.

6. a) Direktmappad cache betyder att det bara finns en väg. Varje adress ur minnet kan bara placeras på ett index i cache. 4096 byte total storlek och 16 byte per line => 4096/16 = 256 cachelines. Det behövs 8 bitar för att peka ut rätt cacheline. Av de 32 bitarna i adressen används då 8 bitar som index i cache (peka på cacheline) och 4 bitar för att beskriva position inom cacheline. Då återstår 32-8-4 = 20 bitar som används som tag.

b) I en direktmappad cache kan en specifik adress endast placeras på en plats i cachen. Det behövs därför bara en jämförelse mellan tag för denna cacheline och adressens tag vid varje minnesaccess.

c) Dela upp adressen i tag, index och position: ökning med $100 hos adressen ger en ökning av index med $10 (4 LSB används till position). Index är 8 bitar, vilket ger $100/$10=$10 olika index innan samma index används igen för en annan adress (med annan tag). Alltså går det att göra 16 läsningar innan data från första läsningen kastas ut ur cache.

d) Om cachelines ökar i storlek behöver fler byte läsas från minnet vid cachemissen. En dubblering av cachelinelängd ger dock mindre än en dubblering av lästid, eftersom resten av minneskommunikationen (jfr cachelabben) oftast är mycket längre än tiden att skicka en byte.