Tentamen
Datorteknik Y, TSEA28

Datum	2013-08-20
Lokal	TER2
Tid	8-12
Kurskod	TSEA28
Provkod	TEN1
Kursnamn	Datorteknik Y
Institution	ISY
Antal frågor	6
Antal sidor (inklusive denna sida)	13
Kursansvarig	Andreas Ehliar
Lärare som besöker skrivsalen *Telefon under skrivtiden*	Andreas Ehliar
Besöker skrivsalen	Cirka 9 och 11
Kursadministratör	Ylva Jernling
Tillåtna hjälpmedel	Inga
Betygsgränser	**Poäng** **Betyg** 41-50 5 31-40 4 21-30 3 0-20 U

Viktig information

- För de uppgifter där du måste göra uträkningar är det viktigt att du redovisar alla relevanta mellansteg.
- Om du i en viss uppgift ska förklara något, tänk på att du gärna får rita figurer för att göra din förklaring tydligare.
- Uppgifterna i tentan är inte ordnade efter svårighetsgrad.
- Skriv inga svar i detta häfte!

Lycka till!

Fråga 1: Binär aritmetik(5p)

(a) (3p) Följande två instruktioner körs i syfte att avgöra om ett tvåkomplementskodat tal är större än ett annat tvåkomplementskodat tal:

```
cmp.l d0,d1     ; Sätt flaggorna som om operationen D1=D1-D0 har körts
bge   label     ; Hoppa om D1 var större än D0
```

Instruktionen `cmp.l` jämför inkommande operander och sätter sedan flaggorna så att instruktionen `bge` hoppar om den andra operanden till `cmp.l` var större än den första enligt kommentarerna ovan *notera att* `bge` *antar att talen som tidigare jämförts är tvåkomplementskodade tal.*

Förklara hur `bge` avgör om hoppet ska tas. För full poäng krävs ett booleskt uttryck där lämpliga flaggor ingår, samt även en eller ett par meningar som förklarar uttrycket.

(b) (2p) Följande två binära tal är tvåkomplementskodade: `11111001`, `11100001`. Multiplicera dessa på valfritt sätt.

Fråga 2: Allmän teori(6p)

(a) (2p) Förklara kortfattat varför en minneshierarki är nödvändig i en modern generell dator.

(b) (2p) Förklara kortfattat vad en datakonflikt är i en pipelinead (*överlappad i Roos*) processor. Ge minst ett exempel.

(c) (2p) I en modern dator finns det vanligtvis flera olika mekanismer som gör det möjligt för ett operativsystem att hindra en godtycklig process från att utöka sina rättigheter (genom att exempelvis skriva och läsa direkt till/ifrån hårddisken). Diskutera kortfatat en sådan mekanism.

Fråga 3: Mikroprogrammering(10p)

I figur 1 återfinns en bekant datormodell som du ska mikroprogrammera i denna uppgift. Notera att det är viktigt att du skriver vilken additionsoperation du valt (den som påverkar flaggorna eller den som inte påverkar flaggorna). (Om du ej skriver något speciellt kommer jag att anta att du ej vill modifiera flaggorna.)

(a) (4p) Skriv mikrokoden för instruktionen `POW2 GRx`. Denna operation ska utföra beräkningen 2^{GRx} och lägga resultatet i GRx. Om värdet i `GRx` är större än 15 får instruktionen returnera godtyckligt värde i `GRx`. Instruktionen får ta max 400 klockcykler att köra, oavsett vilket värde som finns i `GRx`!. Flaggorna får påverkas på godtyckligt sätt av denna instruktion.

Exempel: `GR3` innehåller värdet 5. Efter att instruktionen `POW2 GR3` har körts ska `GR3` innehålla värdet 2^5, dvs 32.

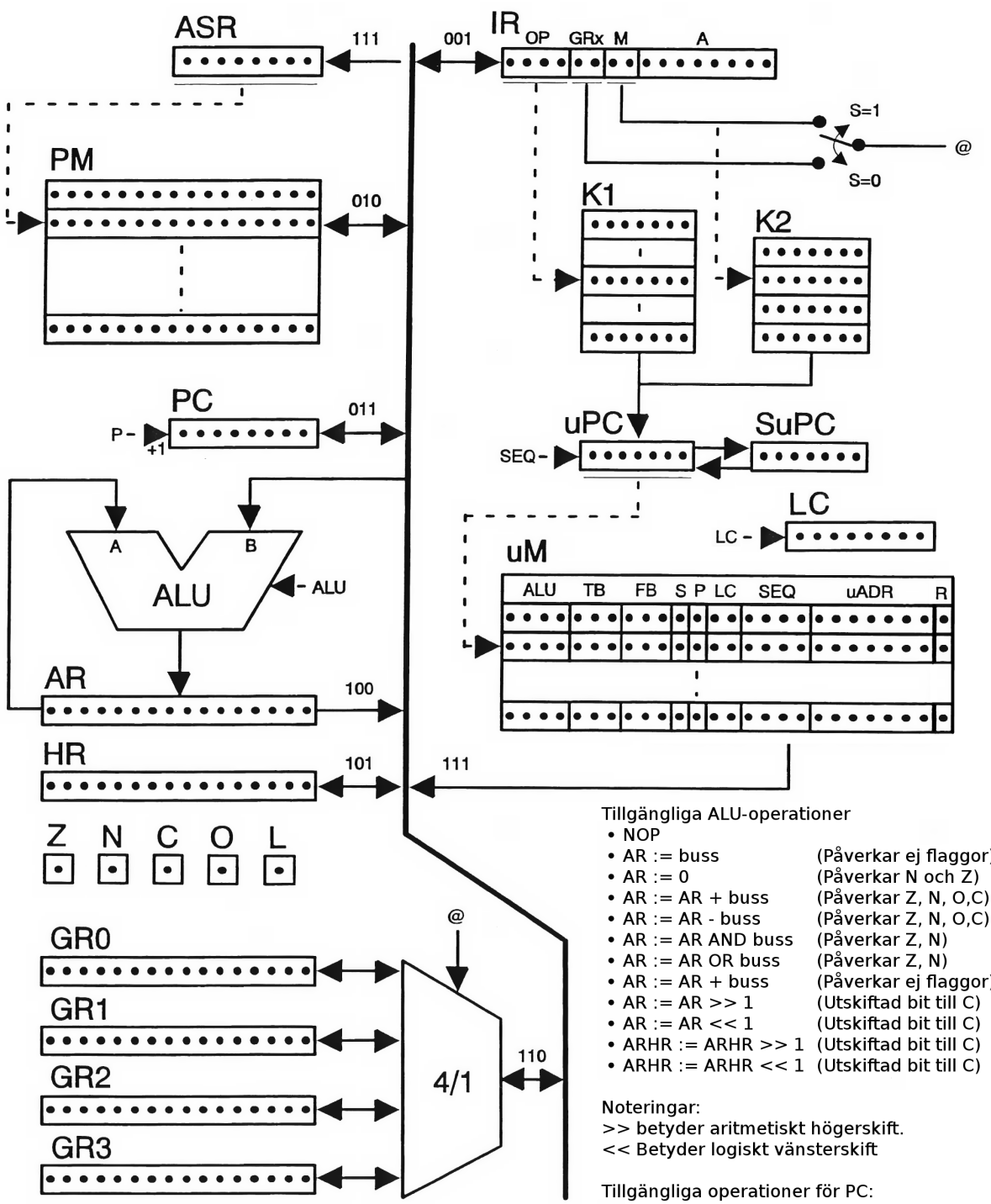
(b) (6p) Skriv mikrokoden för instruktionen `FINDNONZERO GRx`. Denna instruktion ska börja på adress GRx i minnet och sedan leta efter det första ordet som är nollskilt och returnera adressen till detta värde i `GRx`. Flaggorna får påverkas på godtyckilgt sätt av denna instruktion.

Exempel: `GR2` innehåller värdet 33.

PM innehåller följande:

Adress	Värde
30	15
31	0
32	0
33	0
34	0
35	0
36	1953
37	0
38	0

Efter det att instruktionen `FINDNONZERO GR2` har körts ska `GR2` innehålla värdet 36.



Tillgängliga ALU-operationer
- NOP
- AR := buss (Påverkar ej flaggor)
- AR := 0 (Påverkar N och Z)
- AR := AR + buss (Påverkar Z, N, O,C)
- AR := AR - buss (Påverkar Z, N, O,C)
- AR := AR AND buss (Påverkar Z, N)
- AR := AR OR buss (Påverkar Z, N)
- AR := AR + buss (Påverkar ej flaggor)
- AR := AR >> 1 (Utskiftad bit till C)
- AR := AR << 1 (Utskiftad bit till C)
- ARHR := ARHR >> 1 (Utskiftad bit till C)
- ARHR := ARHR << 1 (Utskiftad bit till C)

Noteringar:
>> betyder aritmetiskt högerskift.
<< Betyder logiskt vänsterskift

Tillgängliga operationer för PC:
- NOP
- PC := PC + 1 (PC räknas upp med ett)
- PC := buss

Tillgängliga operationer för LC:
- NOP
- LC räknas ned med ett
- LC laddas från uADR

Tillgängliga operationer för uPC:
- uPC := uPC + 1 (uPC räknas upp med ett)
- uPC := K1(OP)
- uPC := K2(M)
- uPC := 0
- uPC := uADR
- uPC := uADR om (valfri) flagga är 1, annars uPC+1
- uPC := uADR om (valfri) flagga är 0, annars uPC+1

Under varje klockcykel kan även en av följande enheter skicka data från bussen: IR, PM, PC, AR, HR, GRx eller uM. En av följande enheter kan också ta emot data ifrån bussen varje klockcykel: IR, PM, PC, AR, HR, GRx eller ASR Viktigt: När uM (de 16 minst signifikanta bitarna) skickas ut till bussen kan enbart en ALU-operation och/eller en bussöverföring göras. uPC räknas också automatiskt upp med ett i detta fall.

Signalen S väljer om M eller GRx-fältet ska användas till att adressera GR0-GR3 (S=0 innebär att GRx-fältet adresserar GR0-GR3). Slutligen används signalen R=1 för att tvinga styrsignalen @ att bli 3.

Figur 1: En välkänd datormodell (Björn Lindskog 1981)

Fråga 4: Avbrott(9p)

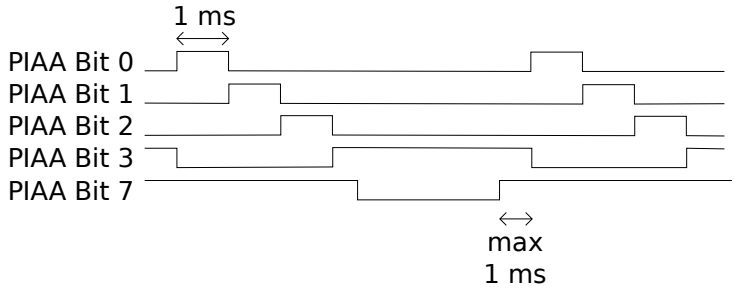
(a) (3p) Följande avbrottsrutiner är implementerade på TUTOR-systemet varav en är korrekt implementerad och en är felaktigt implementerad.

```
avbrottsrutin0:
    AND.W  #$f8ff,SR    ; Acknowledge interrupt(?)
    JSR    GET_DATA
    RTE                 ; Return from exception
avbrottsrutin1:
    TST.B  $10080       ; Acknowledge interrupt(?)
    JSR    GET_DATA
    RTE                 ; Return from exception
```

Förklara vad `AND.W #$f8ff,SR` respektive `TST.B $10080` gör på TUTOR-systemet samt förklara varför en av dessa avbrottsrutiner inte fungerar korrekt och hur felet yttrar sig.

(b) (6p) Linus vill styra en stegmotor som är inkopplade så att följande sekvens måste skickas ut på bit 0-3 på PIAA i TUTOR-systemet för att stegmotorn ska rotera åt rätt håll: 0001, 0010, 0100, 1000, 0001, 0010, ...

Varje del i sekvensen ska skickas ut i en millisekund innan nästa del skickas till PIAA. Förutom detta ska det vara möjligt att stoppa sekvensen genom att skicka in en nolla på bit 7 på PIAA. När bit 7 går hög får det dröja max en millisekund innan sekvensen går igång igen. (Se figur 2.)



Figur 2: Styrsekvensen som krävs för att den stegmotor som Linus använder ska rotera.

För att skicka ut denna sekvens kan exempelvis följande assemblerprogram användas (under antagandet att datariktningsregistret i PIAA redan är korrekt inställt).

```
loop:
    move.b #$1,$10080
    jsr    delay1ms
wait1:
    move.b $10080,d0
    and.b  #$80,d0
    beq    wait1
    move.b #$2,$10080
    jsr    delay1ms
wait2:
    move.b $10080,d0
    and.b  #$80,d0
    beq    wait2
    move.b #$4,$10080
    jsr    delay1ms
wait3:
    move.b $10080,d0
    and.b  #$80,d0
    beq    wait3
    move.b #$8,$10080
    jsr    delay1ms
wait4:
    move.b $10080,d0
    and.b  #$80,d0
    beq    wait4
    bra    loop
```

Nackdelen med detta program är dock att processorn i princip spenderar all tid med att loopa runt i subrutinen `delay1ms`. **Gör om detta program så att det använder avbrott istället för att använda subrutinen `delay1ms` för att fördröja programmet.** Till avbrottsingången som hör till PIAA har det kopplats in en signal som har en frekvens på 1 kHz. Avbrottsrutinen ska vara skriven så att bakgrundsprogrammet som körs ej ska märka av att avbrottsrutinen körs (förutom då att bakgrundsprogrammet inte kör lika snabbt).

Fråga 5: Stack och programförståelse(10p)

Ett visst program har haft vissa mystiska problem som kan härledas till den disassemblerade programkoden nedan:

```
ADRESS    MASKINKOD            INSTRUKTION
-------------------------------------------------
002000    2F01                 MOVE.L  D1,-(A7)
002002    4280                 CLR.L   D0
002004    1218                 MOVE.B  (A0)+,D1
002006    4A01                 TST.B   D1
002008    6712                 BEQ.S   $00201C
00200A    0C01003C             CMP.B   #60,D1
00200E    6606                 BNE.S   $002016
002010    4EBA0014             JSR     $00002026(PC)
002014    60EE                 BRA.S   $002004
002016    5280                 ADDQ.L  #1,D0
002018    12C1                 MOVE.B  D1,(A1)+
00201A    60E8                 BRA.S   $002004
00201C    221F                 MOVE.L  (A7)+,D1
00201E    4E75                 RTS
002020    1E3C00E0             MOVE.B  #224,D7
002024    4E4E                 TRAP    #14
002026    2F00                 MOVE.L  D0,-(A7)
002028    1018                 MOVE.B  (A0)+,D0
00202A    0C000000             CMP.B   #0,D0
00202E    670A                 BEQ.S   $00203A
002030    0C00003E             CMP.B   #62,D0
002034    66F2                 BNE.S   $002028
002036    201F                 MOVE.L  (A7)+,D0
002038    4E75                 RTS
00203A    D1FCFFFFFFFF         ADD.L   #-1,A0
002040    4E75                 RTS
```

(a) (6p) Antag att subrutinen på adress $2000 anropas med A0 satt till $4A00, A1 satt til $6A00 samt A7 satt till $7000.

På adress $4A00 finns följande ASCII-kodade meddelande:

`Linus hade <I>mycket</I> roligt eftersom han studerade <B>datorteknik</B>`

Detta meddelande följs av värdet 0 (alltså inte ASCII-tecknet 0 utan det binära värdet 0).

Förklara vad subrutinen gör när det anropas med parametrar enligt ovan. Förklara också vad som händer med stacken i samband med att denna subrutinen körs. För varje värde som pushas eller popas från stacken vill jag ha reda på så mycket som möjligt av följande parametrar:

- Längd
- Värde (I vissa fall kanske inte hela värdet är känt)

För att kunna svara på denna fråga behöver du veta att ASCII-koden för < är `$3c` samt att ASCII-koden för > är `$3e`.

(b) (2p) Vad händer om det på adress $4A00 och framåt enbart ligger följande meddelande följt av värdet 0?

`Vi tappade bort ett <B>viktigt</B tecken.`

Hur fixar du buggen?

(c) (2p) Vad händer om det på adress $4A00 och framåt istället ligger 5000 mellanslag, (ASCII-kod `$20`)? (Ledning: Håll ordning på stacken!)

Fråga 6: Cache(10p)

(a) (4p) Linnea har fått i uppdrag att snabba upp en datastruktur som används i ett visst program. Datastrukturen är uppbygd på följande sätt:

Minnesadress	Innehåll
`$100000-$10001f`	Information om objekt 0
`$100020-$10003f`	Information om objekt 1
`$100040-$10005f`	Information om objekt 2
...	...
`$8fffe0-$8fffff`	Information om objekt 262143

Det vill säga, den innehåller 262144 objekt, där varje objekt tar upp 32 bytes.

Det finns också en subrutin som används för att läsa in dessa objekt för vidare bearbetning. Skriven i MC68000-assembler skulle denna subrutin kunna se ut exempelvis som följer:

```
; Denna subrutin ska anropas med d0 satt till det objekt vi är intresserade av
process_object:
    ; Beräkna adressen till objektet
    mulu.l #32,d0   ; d0 = d0 * 32
    move.l #$100000,a0
    add.l  d0,a0

    move.l (a0)+,d0
    move.l (a0)+,d1
    move.l (a0)+,d2
    move.l (a0)+,a1
    move.l (a0)+,d3
    move.l (a0)+,d4
    move.l (a0)+,d5
    move.l (a0),a2

    ; Plus programkod här för att analysera datan som precis lästs in, men den
    ; har ingen betydelse för detta exempel.

    rts
```

Som programmet är skrivet så läses objekten i nästan helt slumpmässig ordning. Det vill säga, du kan anta att subrutinen `process_object` anropas med d0 satt till i princip slumpmässiga värden (mellan 0 och 262143).

Efter en lång arbetsvecka har Linnea lyckats ändra på programmet så att datastrukturen ser ut på följande sätt:

Minnesadress	Innehåll
`$100000-$10001b`	Information om objekt 0
`$10001c-$100037`	Information om objekt 1
`$100038-$100053`	Information om objekt 2
...	...
`$7fffe4-$7fffff`	Information om objekt 262143

Det vill säga, hon har minskat datastrukturens storlek ifrån 32 till 28 bytes och således har hon minskat den totala storleken på datastrukturen till 7 MiB (ifrån 8 MiB). Du kan anta att subrutinen `process_object` ser ut i princip som tidigare, förutom att d0 multipliceras med 28 istället för 32 och att instruktionen (`move.l (a0)+,d0`) försvinner. Tanken är att eftersom datastrukturen tar mindre plats borde cachen utnyttjas mer effektivt.

Tyvärr visar det sig att subrutinen `process_object` nu har blivit betydligt långsammare än tidigare. **Förklara vad detta beror på.** Antag att programet körs på en modern processor i stil med den ni använde i cachelaborationen.

(b) (6p) En viss cache innehåller 131072 (2^{17} bytes). Den är gruppassociativ och har två vägar. En cacheline är 16 bytes. Du kan anta att cachen är tömd (flushad). Du kan också anta att hela minnet är markerat som cachebart. Cachen delar in adressen på så sätt att de mest signifikanta bitarna används till adressen i märkarean (*tag*). I övrigt får du anta godtyckliga (men rimliga) parametrar för detaljer som exempelvis ersättningsalgoritm, tillbakaskrivningspolicy och så vidare, så länge du i uppgiften förklarar vilka detaljer du antagit.

Antag att ett program gör följande minnesaccesser:

Minnesåtkomst nummer 0-3	`$004004`, `$008008`, `$008080`, `$008086`
Minnesåtkomst nummer 4-7	`$FF4000`, `$FF4001`, `$FF4002`, `$FF4003`
Minnesåtkomst nummer 8-11	`$00CAFE`, `$00F00D`, `$011000`, `$012000`
Minnesåtkomst nummer 12-15	`$013000`, `$014000`, `$021000`, `$022000`
Minnesåtkomst nummer 16-20	`$023000`, `$024000`, `$004000`, `$004000`

Rita en tabell som beskriver cachens tillstånd efter de läsningar som skett ovan. För full poäng måste du också redovisa varför cachens tillstånd ser ut som det gör genom att beskriva vad som händer (eller inte händer) på bussen för varje läsning ovan.

Kort repetition av M68000

Mnemonic	Kort förklaring	Mnemonic	Kort förklaring
ADD	Addition	EXG	Exchange
ADDX	Add with X flag	EXT	Sign extend
AND	Logic and	EXTB	Sign extend a byte to 32 bit
ASL	Arithmetic shift left	LEA	Load effective address
ASR	Arithmetic shift right	LSL	Logic shift left
BCC	Branch on carry clear	LSR	Logic shift right
BCS	Branch on carry set	MOVE	Move
BEQ	Branch on equal	MULS	Signed multiplication
BGT	Branch on greater than	MULU	Unsigned multiplication
BLT	Branch on less than	NEG	Negate
BNE	Branch on not equal	NOP	No operation
BRA	Branch always	OR	Logic OR
BSR	Branch to subroutine	ROL	Rotate left
CLR	Clear	ROR	Rotate right
CMP	Compare (Destination - Source)	RTE	Return from exception
DIVS	Signed division	RTS	Return from subroutine
DIVU	Unsigned division	SUB	Subtract
EOR	Logic XOR	TST	Set integer condition codes

```
; Exempel på M68000 kod som kopierar 200 bytes från $2000 till $3000
    MOVE.L #$2000,A0
    MOVE.L #$3000,A1
    MOVE.B #50,D0
loop
    MOVE.L (A0)+,(A1)+
    ADD.B #-1,D0
    BNE loop
```

Lösningsförslag fråga 1

a)

Om `D1 = D0` är Z=1, villkoret ej uppfyllt. Om vi inte har hamnat utanför talområdet gäller även följande: N=0 om `D1` är större än `D0`. Hamnar vi utanför talområdet (V=1) gäller istället motsatsen (N=1 om `D1` är större än `D0`).
Villkoret för att hoppet ska tas är alltså: $\overline{Z} \cdot \overline{(N \oplus V)}$

b)

Lättast räknar man troligtvis ut detta genom att inse att talen är teckenutökade. Det vill säga:
`11111001 x 11100001 = 1001 x 10001` för tvåkomplementskodade tal:

```
     100001   (-31)
   *  1001   (-7)
   -------
 1111100001   (-31)
    00000
   00000
-1100001      (-31)*(-8)
-----------
 0011011001   (Kontroll: 1+8+16+64+128 = 217)
```

Lösningsförslag fråga 2

a)

Full poäng kan här ges exempelvis för den som diskuterar pris kontra prestanda på DRAM respektive SRAM samt sekundärminne, samt ställer detta i relation till att de flesta program uppvisar en lokalitet som gör att en minneshierarki faktiskt ger en prestandavinst.

b)

I en pipelinead processor kan det uppstå en situation där en viss instruktion håller på att beräkna ett resultat som en senare instruktion behöver. Om resultatet inte är färdigberäknat i pipelinen innan den senare instruktionen behöver resultatet uppstår en datakonflikt (av typen RAW, read-after-write). Ett exempel:

move.b (a0)+,d0 ; Minnesläsning sker troligtvis sent i pipelinen add.b d0,d1 ; Operander till add läses troligtvis tidigt i ; pipelinen —¿ datakonflikt

c)

Full poäng ges för exempelvis en kortfattad diskussion om hur antingen en MMU eller S-biten i statusregistret på MC68000 (eller motsvarande på andra processorer) används för att se till så att ett program inte kan förstöra för andra program. Se läroboken (för MMU) eller labmanualen (för S-biten).

Lösningsförslag fråga 3

```
; Om inget annat anges gäller uPC++

pow2:
   AR := 1   ; Ettan kommer ifrån uM
   HR := AR  ; HR kommer till slut att innehålla 2^GRX
   AR := AR + GRx ; AR innehåller hur mycket vi ska skifta + 1 (behöver ej påverka flaggor)
   AR := AR AND 15 ; Värdet 15 kommer ifrån uM
                   ; (För att se till att vi garanterat kommer
                   ; under 400 cykler.)
loop:
   AR := AR - 1    ; (ettan från buss)
   uPC := pow2klar om Z = 1, annars uPC++
   ARHR := ARHR << 1
   AR := AR >> 1 ; uPC := loop

pow2klar:
   uPC := 0 ; GRx := HR

findnonzero:
   ASR := GRx; AR := 0
   AR := AR OR PM
   AR := GRx; uPC := findnonzeroklar om Z = 0, annars uPC++
   AR := AR + 1 ; Ettan från uM (behöver ej påverka flaggor)
   GRx := AR; uPC := findnonzero
findnonzeroklar:
   uPC := 0
```

Notering: Det var en hel del som missade poäng på b-uppgiften då de använde PC som adressräknare utan att återställa denna till ursprungsvärdet när `findnonzero` kört klart. (Så länge man sparar undan PC så att den återställs i slutet på `findnonzero` så är det däremot ok.)

Lösningsförslag fråga 4

a)

- `AND.W #$f8ff,SR` sänker avbrottsnivån till 0 på MC68008.
- `TST.B $10080` gör en läsning ifrån PIAA, vilket gör att PIAA förstår att vi tagit hand om avbrottet.

Den korrekta varianten är således den med `TST.B`. Den andra varianten kommer att leda till att MC68008 får ett avbrott av samma typ på direkten, eftersom PIAA ännu inte vet om att vi tagit hand om avbrottet. Eftersom RTE inte körts i detta läge kommer stacken att växa med 6 bytes varje gång detta inträffar vilket slutligen kommer att leda till att TUTOR kraschar inom mindre än en sekund.

b)

```
handle_interrupt:
    move.b d0,-(a7)
    move.b $10080,d0

    and.b  #$80,d0   ; Om PIAA Bit 7 -> Pausa sekvensen
    beq    done

    move.b $10080,d0 ; Läs upp och skifta
    lsl.b  #1,d0
```

```
    and.b  #$f,d0
    beq    restart_sequence

    move.b d0,$10080
done:
    move.b (a7)+,d0
    rte

restart_sequence:
    move.b #$1,$10080
    bra paused
```

Lösningsförslag fråga 5

a)

Subrutinen tar strängen på adress $4A00 och kopierar den till adress $6a00 förutom att de delar av strängen som är inneslutna i <> ej kopieras (tecknen < och > kopieras inte heller).

Det som händer på stacken när subrutinen körs är följande:

PC	Stackoperation	Antal bytes	Kommentar Kommentar
$2000	PUSH(Okänt)	4	D1 sparas undan
$2010	PUSH($00002014)	4	Återhoppsadress från JSR
$2026	PUSH(11)	4	D0 sparas undan
$2036	POP(11)	4	D0 återställs
$2038	POP($00002014)	4	Återhoppsadressen från subrutinen
Dessa fyra händelser upprepas nu tre gånger till, fast med värdet 17, 48 och 59 istället för 11			
$201C	POP(Okänt)	4	D1 återställs
$201E	POP(Okänt)	4	Återhoppsadress till den subrutin som anropade $22000

(`D0` används till att spara hur många tecken som kopierats hittills.)

b)

Programmet kör i subrutin $2026 när det når sista tecknet i strängen. Då kommer subrutinen att på adress $2040 göra ett återhopp utan att först ha poppat `D0` vilket kommer att leda till en krasch eftersom den då försöker hoppa tillbaka till totalt fel adress. Detta fixas lätt genom att lägga in en `MOVE.L (A7)+,D0` innan `RTS` på rad $2040.

c)

När 5000 mellanslag har kopierats så kommer programmet att krascha eftersom stacken har skrivits över med $20202020. När den kopierat alla tecken kommer den då att sätta PC till $20202020[1] vilket troligtvis inte är programmerarens avsikt att tillåta.[2]

[1] Eftersom MC68008 enbart har 20 bitars adressrymd kommer den dock glatt att ignorera de översta 12 bitarna och hoppa till adress $02020 istället. Vad som händer sedan beror på om vi kör i supervisor-läge eller user-läge. Kör vi i supervisor-läge kommer vi att krascha eftersom vi har förstört TUTORs supervisor-stack. Kör vi i user-läge kommer dock systemanropet på adress $2024 att lyckas.

[2] Speciellt inte om det är så att texten kommer ifrån en källa som programmeraren inte kontrollerar, exempelvis en websida. I detta läge kan man tänka sig att en person med tveksamma avsikter kan manipulera texten som dyker upp så att den innehåller maskinkod för MC68008 och sedan manipulera slutet på strängen så att det värde som poppas på adress $201E pekar rätt på maskinkoden. Detta brukar kallas för en *buffer overflow attack* och är tyvärr ett av de allra vanligaste säkerhetshålen.

Lösningsförslag fråga 6

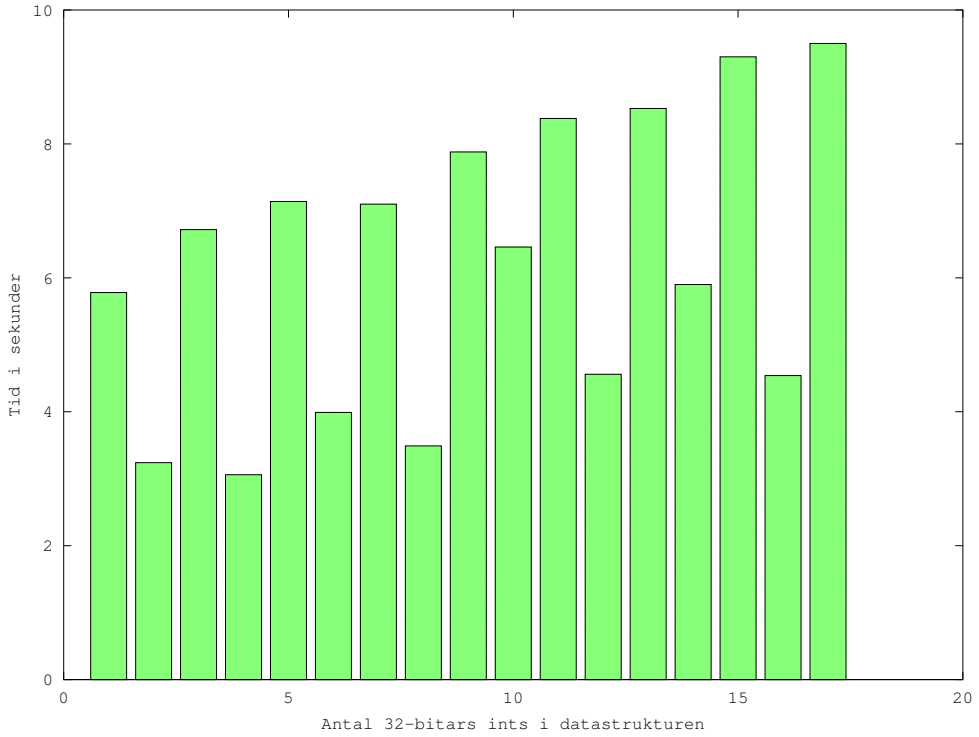
a)

Om vi antar att en cacheline är minst 32 bytes kommer den ursprungliga varianten aldrig att behöva läsa in mer än en cacheline per anrop till `process_object`. (Åtminstone om vi antar att alla cachemissar sker innan datan analyseras i denna subrutin.)

Ändras datastrukturen så att den istället är 28 bytes kommer många av objekten att befinna sig på mer än en cacheline vilket kommer att resultera i en klar prestandaminskning.

Prestandamätning

För den som är intresserad kan jag nämna att jag testkörde en liknande applikation på en Intel Core 2 Duo (E7400) där jag jämförde prestandan för slumpmässiga läsningar av samtliga element i objekt av olika storlek. Resultatet kan ses i figur 3. Notera att prestandan för exempelvis 4, 8, samt 16 "intar" i datastrukturen är betydligt bättre än för 3, 7, samt 15 intar.



Figur 3: Tidsåtgång för slumpmässig läsning av datastrukturer i minnet

Det kan också nämnas att man i exempelvis källkoden till Linux kan se att programmerare har anpassat vissa datastrukturer så att de precis ska passa in i ett visst antal cachelines. Notera dock att detta enbart är ett stort problem när programmet uppvisar liten lokalitet i sina minnesläsningar! Om detta inte gäller är det sannolikt att Linneas ändring faktiskt hade förbättrat prestandan. (Alternativt hade prestandan kunnat förbättras om optimeringen gjorde att alla objekt helt plötsligt fick plats i exempelvis L2-cachen.)

b)

En adress delas upp på följande sätt om vi använder denna cache:

23 16	15 4	3 0
Tag	Index	Byte i cacheline

Om vi antar att ersättningsalgoritmen LRU används kommer bussaktiviteten se ut på följande sätt:

Åtkomst nr.	Minnesadress	Bussaktivitet
1	`$004004`	Cachemiss, $4000-$400F läses
2	`$008008`	Cachemiss, $8000-$800F läses
3	`$008080`	Cachemiss, $8080-$808F läses
4	`$008086`	Cacheträff, ingen aktivitet
5	`$FF4000`	Cachemiss, $FF4000-$FF400F läses
6	`$FF4001`	Cacheträff, ingen aktivitet
7	`$FF4002`	Cacheträff, ingen aktivitet
8	`$FF4003`	Cacheträff, ingen aktivitet
9	`$00CAFE`	Cachemiss, $CAF0-$CAFF läses
10	`$00F00D`	Cachemiss, $F000-$F00F läses
11	`$011000`	Cachemiss, $11000-$1100F läses
12	`$012000`	Cachemiss, $12000-$1200F läses
13	`$013000`	Cachemiss, $13000-$1300F läses
14	`$014000`	Cachemiss, $14000-$1400F läses
		Cacheline med $4000-$400F invalideras
15	`$021000`	Cachemiss, $21000-$2100F läses
16	`$022000`	Cachemiss, $22000-$2200F läses
17	`$023000`	Cachemiss, $23000-$2300F läses
18	`$024000`	Cachemiss, $24000-$2400F läses
		Cacheline med $FF4000-$FF400F invalideras
19	`$004000`	Cachemiss, $04000-$0400F läses
		Cacheline med $014000-$01400F invalideras
20	`$004000`	Cacheträff, ingen aktivitet

I cachen ser det ut som följer (alla cachelines utan innehåll är markerade som ogiltiga (*invalid*)).

		Väg 0		Väg 1	
Index	Äldsta väg	Tag	Innehåll från minne	Tag	Innehåll från minne
`$100`	0	`$01`	`$011000-$01100F`	`$02`	`$021000-$02100F`
`$200`	0	`$01`	`$012000-$01200F`	`$02`	`$022000-$02200F`
`$300`	0	`$01`	`$013000-$01300F`	`$02`	`$023000-$02300F`
`$400`	1	`$00`	`$004000-$00400F`	`$02`	`$024000-$02400F`
`$800`	0	`$00`	`$008000-$00800F`		
`$808`	0	`$00`	`$008080-$00808F`		
`$CAF`	0	`$00`	`$00CAF0-$00CAFF`		
`$F00`	0	`$00`	`$00F000-$00F00F`		