Tentamen
Datorteknik Y, TSEA28

Datum	2014-05-28
Lokal	TER2 och TER3
Tid	14-18
Kurskod	TSEA28
Provkod	TEN1
Kursnamn	Datorteknik Y
Institution	ISY
Antal frågor	6
Antal sidor (inklusive denna sida)	14
Kursansvarig	Andreas Ehliar
Lärare som besöker skrivsalen *Telefon under skrivtiden*	Andreas Ehliar
Besöker skrivsalen	Cirka 15 och 17
Kursadministratör	Anita Kilander
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.
- **Förtydligande 2014:** I de uppgifter där du ska skriva assembler- eller mikro-kod är det viktigt att du kommenterar koden.
- 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: Stack och avbrott(5p)

(a) (3p) Beskriv vad som händer i MC68000 när ett avbrott sker?

(b) (2p) I TUTOR-systemet kan avbrott ske genom att parallelporten är konfiguerad till att ge avbrott vid en viss händelse på en I/O-pinne. Avbrott kan också ske genom att vissa instruktioner körs i ett program. Nämn en instruktion som kan orsaka ett avbrott samt förklara vilka villkor som måste vara uppfyllda för att denna instruktion ska ge ett avbrott.

Fråga 2: Allmänbildning(10p)

(a) (2p) Vad innebär det att en processor är superskalär? (Skriv max fem meningar.)

(b) (4p) Under vilka omständigheter ettställs respektive nollställs C, N, V (ibland kallad O) och Z-flaggorna?

(c) (2p) Nämn minst en anledning till att du vill ha en MMU i en dator.

(d) (2p) En modern buss (såsom AXI) har ett antal detaljer som används för att förbättra prestandan på exempelvis minnesläsningar/skrivningar. Diskutera kortfattat en sådan detalj. (Skriv max fem meningar.)

Fråga 3: Mikroprogrammering(8p)

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.)

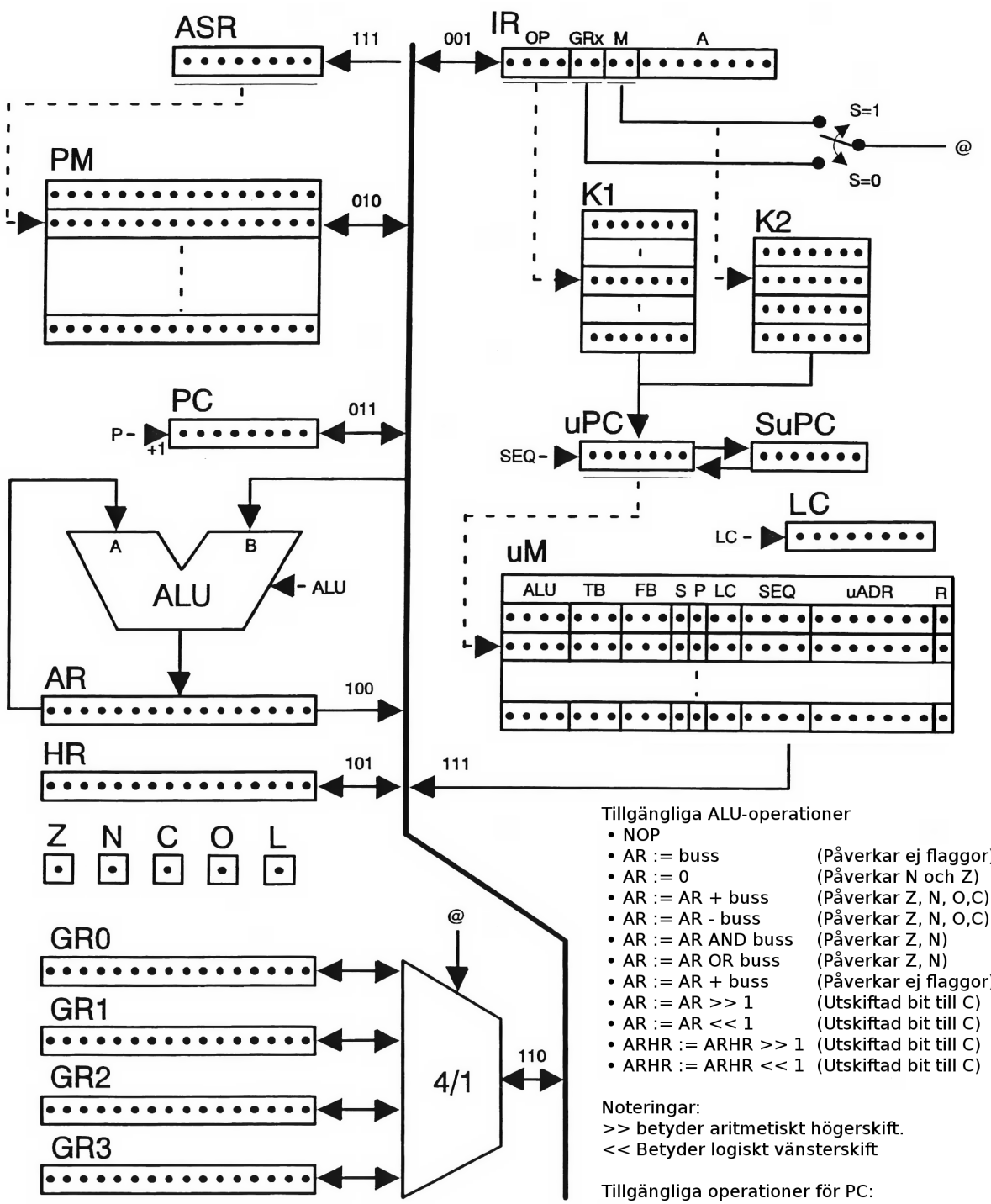
(a) (6p) Skriv all mikrokod (det vill säga mikrokod för alla nedanstående instruktioner samt mikrokod för hämtfas och nödvändiga adresseringslägen) som krävs för att nedanstående assemblerprogram ska kunna köras på denna processor:

```
; Destinationsregistret står längst till höger (precis som för MC68000)
    MOVE    #$10,GR0    ; Omedelbar operand
    MOVE    #$80,GR3
loop:
    CMP     (GR3),GR0   ; Indexerad adressering
    BEQ     ENDLOOP
    ADD     #1,GR3
    CMP     #$90,GR3
    BNE     loop

    HALT  ; Mikrokod för HALT behöver ej skrivas
```

(b) (1p) Förklara vad K1 respektive K2 används till?

(c) (1p) Uppskatta hur många klockcykler det tar att köra ditt program på denna dator.



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.

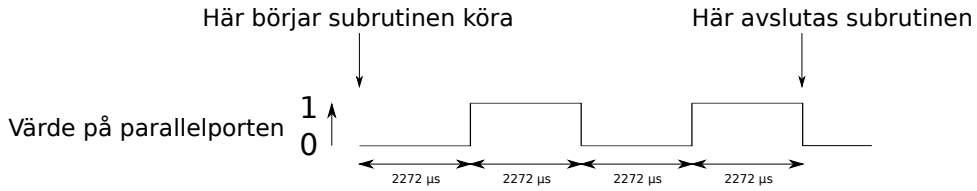
Signalen S väljer om M eller GRx-fältet ska användas till att adressera GR0-GR3 (S=0 innebär att GRx-fältet adresserar GR0-GR3). Slutligen används signalen R=1 för att tvinga styrsignalen @ att bli 3.

Figur 1: En välkänd datormodell (Björn Lindskog 1981)

Fråga 4: Assemblerprogrammering:(12p)

Notering: Uppgifterna i denna assemblerprogrammeringsuppgift hänger ihop, men är möjliga att svara på individuellt. (Tips: uppgift c är troligtvis enklast.)

(a) (5p) Skriv subrutinen `sendpulse` som skickar ut en utsignal på parallelporten som växlar periodiskt mellan noll och ett. Det värde som finns i `D0` bestämmer det gånger som utsignalen ska växla mellan ett och noll. I `D1` finns tiden (mätt i mikrosekunder) som det ska ta innan utsignalen växlar från noll till ett och vice versa. **När din subrutin avslutas måste alla data och adressregister ha samma värden som när subrutinen anropades.** Se figur a för ett exempel.



Figur 2: Utsignalens utseende på parallelporten då `sendpulse` anropas med `D1` satt till 2272 och `D0` satt till 4.

Till din hjälp har du två stycken subrutiner:

- `delay`: Denna subrutin fördröjer programmet. Fördröjningen specificeras i `D1` och anges i mikrosekunder. Subrutinen förstör innehållet i `D1` men ändrar inte på något annat register.
- `setio`: Denna subrutin skickar ut ett värde på parallelporten. Värdet som ska skickas ut anges i register `D2`. (Endast den minst signifikanta biten i `D2` skickas ut, övriga bitar ska ha värdet 0.) Denna subrutin ändrar inte på innehållet i något register.

(b) (4p) Du har fått i uppdrag att snygga upp en kodbas som har ett hundratal subrutiner som ser ut ungefär som nedanstående kodsnutt.

```
manypulses_variant_532:
     move.l #$AD,d0   ; *              move.l #$4FB0,d1
     move.l #$5ED,d1  ; *              jsr delay
     jsr sendpulse    ; *              move.l #$1BE,d0
     move.l #$1A90,d1 ; **             move.l #$6A7,d1
     jsr delay        ; **             jsr sendpulse
     move.l #$AD,d0                    move.l #$4FB0,d1
     move.l #$5ED,d1                   jsr delay
     jsr sendpulse                     move.l #$A7,d0
     move.l #$1A90,d1                  move.l #$5ED,d1
     jsr delay                         jsr sendpulse
     move.l #$AD,d0                    move.l #$22365,d1
     move.l #$5ED,d1                   jsr delay
     jsr sendpulse    ; ***            move.l #$95,d0
     move.l #$1A90,d1                  move.l #$6A7,d1
     jsr delay                         jsr sendpulse
     move.l #$1F4,d0                   move.l #$1A90,d1
     move.l #$5ED,d1                   jsr delay
     jsr sendpulse                     move.l #$404,d0
     move.l #$4FB0,d1                  move.l #$5ED,d1
     jsr delay                         jsr sendpulse
     move.l #$18D,d0
     move.l #$777,d1                   rts
     jsr sendpules
     ; Fortsättning i nästa kolumn
```

Denna kodbas är uppenbarligen skriven på ett tämligen klumpigt sätt. För att göra denna kodbas lättare att underhålla och utveckla är tanken att koda ner informationen om anropen till sendpulse respektive delay i ett mer kompakt format.

Din uppgift i denna deluppgift är att skriva subrutinen `parse_pulsedata` enligt nedanstående specifikationer:

När `parse_pulsedata` anropas är `A0` satt till att peka på en datastruktur som subrutinen ska hantera på två olika sätt, beroende på om det första ordet innehåller värdet ett eller två.

Om det första ordet innehåller värdet ett ska subrutinen `delay` anropas med `D1` satt till innehållet i nästa långord enligt tabellen nedan:

	Förskjutning i datastrukturen	Minnesinnehåll
A0 →	Byte 0-1	1
	Byte 2-5	Nytt värde på D1

Tabell 1: Datastrukturens format då första ordet har värdet ett.

Om det första ordet istället innehåller värdet två ska subrutinen `sendpulse` anropas med `D0` och `D1` satta enligt nedan:

	Förskjutning i datastrukturen	Minnesinnehåll
A0 →	Byte 0-1	2
	Byte 2-3	Nytt värde på D0
	Byte 4-5	Nytt värde på D1

Tabell 2: Datastrukturens format då första ordet har värdet två.

Subrutinen `parse_pulsedata` ska inte ändra i processorns data och adressregister, förutom att `A0` ska räknas upp med 6.

Exempel 1: Antag att minnet på plats $4000 och framåt innehåller värden enligt tabell 3 samt att A0 innehåller värdet $4000 när funktionen `parse_pulsedata` anropas:

```
004000    00 02 00 AD 05 ED 00 01  00 00 1A 90 00 02 00 AD
004010    05 ED 00 01 00 00 1A 90  00 02 00 AD 05 ED 00 01
                      och så vidare...
```

Tabell 3: Exempel på minnesinnehåll för uppgift b och c.

Detta ska medföra att subrutinen `sendpulse` anropas med `D0` satt till $000000AD samt `D1` satt till $000005ED. (Vilket motsvarar de tre rader markerade med `*` i programlistningen ovan.)

Exempel 2: Antag att `A0` istället innehåller värdet $4006. Isåfall ska subrutinen `delay` anropas med `D0` satt till $00001A90. (Vilket motsvarar de två rader markerade med `**` i programlistningen ovan.)

(c) (3p) Skriv en subrutin `send_many_pulses` som tolkar flera, efter varandra liggandes, datastrukturer av den typ som beskrivits i b-uppgiften (med hjälp av funktionen det är tänkt att du ska skriva i uppgift b). `A0` innehåller adressen till den första datastrukturen och `D0` innehåller antalet datastrukturer.

Denna subrutin får ändra på alla register.

Exempel: Om `A0` innehåller värdet $4000 och D0 innehåller värdet 5 och minnesinnehållet ser ut som minnesdumpen i tabell 3 så ska detta motsvara de fem första anropen till `delay`/`sendpulse` i programlistningen ovan. (Slutraden är markerad med `***`.)

Fråga 5: Pipelining(10p)

Antag att du har följande subrutin som ska köras på en MC68000-kompatibel processor som har en pipeline som (under ideala omständigheter) tillåter att en instruktion att startas per klockcykel.

```
dotproduct:
        move.l #40,d0      ; d0 är loopräknaren
        clr.l d1           ; d1 är ackumulator
loop:
        move.w (a0)+,d2    ; Räkna ut d1 = d1 + M[a0++]*M[a1++]
        move.w (a1)+,d3
        muls.w  d2,d3
        add.l  d3,d1

        add.w  #-1,d0      ; Räkna ner vår loopräknare
        bne    loop

        rts
```

I frågorna nedan får du max använda fem meningar per fråga.

(a) (3p) Vad är pipelining och varför vill man använda det i en dator? (För full poäng måste du även rita en förklarande figur!)

(b) (2p) Vad är en datakonflikt och varför uppkommer en sådan i en pipelinead processor?

(c) (1p) Visa på en potentiell datakonflikt i listningen ovan.

(d) (2p) Vad är en styrkonflikt och varför uppkommer en sådan i en pipelinead processor?

(e) (2p) Nämn minst ett sätt att hantera antingen en datakonflikt eller styrkonflikt i en pipelinead processor.

Fråga 6: Cache(5p)

Antag att ett program gör minnesläsningar enligt följande mönster:

	Läsadress				
Läsning 0-127	0	STRIDE	$2 \cdot$ STRIDE	$3 \cdot$ STRIDE	...
Läsning 128-255	1	$1 +$ STRIDE	$1 + 2 \cdot$ STRIDE	$1 + 3 \cdot$ STRIDE	...
Läsning 256-383	2	$2 +$ STRIDE	$2 + 2 \cdot$ STRIDE	$2 + 3 \cdot$ STRIDE	...
...			...		
Läsning 16256-16383	127	$127 +$ STRIDE	$127 + 2 \cdot$ STRIDE	$127 + 3 \cdot$ STRIDE	...

Antag att STRIDE antingen är 1000 eller 1024. Du har mätt upp tiden det tar att köra detta program på två stycken processorer enligt nedan:

Processor	Tid för STRIDE = 1000	Tid för STRIDE = 1024
Processor 1	Cirka 200 μs	Cirka 200 μs
Processor 2	Cirka 200 μs	Cirka 1600 μs

Du vet följande detaljer om processorerna:

- Det enda av vikt som skiljer sig mellan processor 1 och processor 2 är hur datacachen fungerar
- Du vet att datacachens storlek är 64 KiB (dvs 2^{16} bytes) i båda fallen
- Det finns endast en nivå cache i systemet
- I detta exempel får du anta att du ej får några instruktionscachemissar.

Diskutera varför prestandan i detta program kan variera såpass mycket trots att cachens storlek i båda processorerna är lika stora. Använd max två sidor, inklusive eventuella tabeller och figurer.

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: Stack och avbrott

a)

Förutsatt att avbrott är tillåtet (dvs att inkommande avbrott har en högre nivå än aktuell avbrottsnivå, eller att inkommande avbrott är på nivå sju) kommer följande att hända:

- Återhoppsadressen pushas på supervisor-stacken
- Statusregistret pushas på supervisor-stacken
- Supervisor-flaggan sätts till ett (om den inte redan var ett)
- Statusregistret uppdateras med ny avbrottsnivå
- Aktuell avbrottsvektor läses in ifrån avbrottsvektortabellen (som ligger på adress 0 och framåt i minnet) och PC sätts till denna adress

(Det spelar ingen roll om du råkat skriva att SR pushas före PC istället för den korrekta ordningen.)

b)

Här finns det givetvis många olika möjliga svar. Ett par exempel på acceptabla svar:

- Mest uppenbart fall: Du kör instruktionen `trap` för att göra ett operativsystemanrop. Denna instruktion ger alltid ett mjukvaruavbrott.
- Du kör instruktionen `move.l` där du försöker läsa ifrån en udda adress. Detta är inte tillåtet på MC68000 och du kommer att få en address trap error
- Du kör instruktionen `divs` och du försöker dividera med noll.
- Supervisor-flaggan är inte satt och du försöker köra en priviligerad instruktion.

Lösningsförslag, fråga 2: Allmänbildning

a)

För full poäng här behöver det dels framgå att den kan köra flera instruktioner samtidigt och dels att den automatiskt upptäcker vilka instruktioner som den kan köra samtidigt.

b)

Till att börja med så ändras flaggorna bara i samband med att en aritmetisk operation körs (exempelvis addition eller subtraktion). (Ska man vara riktigt petig så ändras inte alltid alla flaggorna, exempelvis så ändrar inte skiftoperationerna på V-flaggan.)

- N: Ettställs om den mest signifikanta biten i en ALU-operation är ett, annars nollställs biten.
- Z: Ettställs om alla bitar i resultatet är noll, annars nollställs biten.
- C: Ettställs om resultatet av en addition av två N bitars tal är större än eller lika med 2^N. (Det vill säga, om carry-out från adderaren i ALU:n var ett.) Ettställs också om den bit vi skiftade ut vid en skiftoperation/rotationsoperation var ett. Nollställs annars.
- V: Ettställs om resultatet av en addition av två N bitars tvåkomplementskodade tal antingen var större än eller lika med 2^{N-1} eller mindre än -2^{N-1}. (Dvs att resultatet hamnade utanför det tillåtna talområdet.)

c)

En tänkbar förklaring är att MMU:n gör det möjlighet att skydda minne från skrivning och/eller läsning. Ett operativsystem kan konfigurera MMU:n så att program enbart har tillgång till just de delar av minnet som de har allokerat. Detta innebär att en bugg i ett program som exempelvis gör så att det börjar skriva på godtyckliga platser i minnet inte kommer att kunna skriva över något annat än sina delar av programmet.

Andra tänkbara förklaringar skulle kunna handla om hur paging hanteras när du har virtuellt minne eller att undvika minnesfragmentering.

d)

För de som gjort lab 5 så ligger det kanske närmast till hands att diskutera möjligheten att ARBURST är 2, så att en cirkulär läsning görs istället för en linjär. Det vill säga, om vi får en cachemiss mitt i en cacheline så kan vi säga till bussen att vi vill läsa den det ord vi faktiskt är intresserade av först. Sedan läser vi in ord fram till slutet av cachelinen och när vi nått slutet på cachelinen läser vi från första ordet i cachelinen fram till ordet precis innan det ord vi fick cachemiss på.

Andra möjliga förklaringar skulle kunna handla om att vi kan göra burstläsningar överhuvudtaget (eftersom det passar bra ihop med hur externa minnen fungerar), samt att bussen är pipelinead (så att vi kan börja skicka exempelvis en läsförfrågan innan vi har fått tillbaka ett svar på en tidigare transaktion).

Mikroprogrammering

a)

```
(Om inget annat anges antas uPC++ på varje rad.)

Adress
 i uM   Mikrokod
0       ASR := PC; PC++      ; Hämtfas
1       IR  := PM
2       uPC := K2(M-fältet)

3       ASR := PC; PC++; uPC := K1(Op-fältet) ; Omedelbar operand

; Två möjliga alternativ finns för att hantera fallet indexerad
; adressering. Det ena fallet är listat i labhäftet för
; mikroprogrammeringslabben och hanterar fallet indexerad adressering
; med förskjutning.  Det skulle alltså kunna hantera instruktioner i
; stil med som MOVE 6(GR3), GR0. Det andra alternativ är att enbart
; hantera indexerad adressering men ej ha stöd för förskjutning
; (offset). Det alternativet listas nedan:

; Notering: Signalen R är en tillökning till MIA-datorn som finns i
; tentan.  Ett annat sätt att lösa detta är att sätta S := 1 och kräva
; att indexerad adressering kodas som M=3.
4       ASR := GRx, R:=1, uPC := K1(Op-fältet) ; Indexerad adressering
                                               ; (utan förskjutning)

5       GRx := PM; uPC := 0       ; MOVE-instruktionen

6       AR := PM                  ; CMP-instruktionen
7       AR := AR - GRx; uPC := 0

8       AR := PM                  ; ADD-instruktionen
9       AR := AR + GRx (ändrar flaggor);
10      GRx := AR; uPC := 0
```

```
11      AR := PC                  ; Mikrokod för att göra ett
12      AR := AR + IR             ; PC-relativt hopp
13      PC := AR, uPC := 0

14      uPC := 11 om Z = 1, annars uPC++   ; BEQ
15      uPC := 0

16      uPC := 11 om Z = 0, annars uPC++   ; BNE
17      uPC := 0
```

b)

K1 är ett minne som används som en uppslagstabell för att avkoda vilken adress mikrokoden för en given instruktion finns på genom att op-kod fältet i maskinkodsinstruktionen skickas in som adress i K1.

K2 är ett minne som används som uppslagstabell för att avkoda vilken adress mikrokoden för ett givet adresseringsläge finns på genom att M-fältet i maskinkodsinstruktionen skickas in som adress i K2.

c)

Klockcykler för de olika instruktionerna, inklusive hämtfas/adressfas: MOVE: 5 CMP: 6 BEQ/BNE: 6 (tas ej), 8 (tas) ADD: 7

MOVE innan loopen tar: $(1 + 4) * 2$ klockcykler En iteration i loopen där BEQ ej tas och BNE tas tar $6 + 6 + 7 + 6 + 8 = 33$ cykler.

Koden tar alltså cirka $10 + 33 \cdot N$ cykler, där N är antalet iterationer innan talet \$10 hittas i minnet. (N är dock maximalt 16.)

Eftersom det står uppskatta så räcker det dock med att exempelvis konstatera att en iteration tar x cykler (hur många cykler det nu handlar om i just ditt fall), vilket gör att det sammanlagt tar $y \cdot x$ cykler, där y är antalet iteration innan värdet \$10 hittas i minnet.

Fråga 4: Assemblerprogrammering

Här finns det givetvis många varianter, här nedan följer mitt förslag på hur detta kan lösas. Eftersom vi även programmerar på papper har jag accepterat att det är rimligt att ett slarvfel smyger sig in i lösningen, exempelvis att

a)

```
sendpulse:
    movem.l d0/d2-d3,-(a7)  ; Spara undan register vi tänker använda
    move.l d1,d3            ; Spara undan delay-värdet
    clr.b  d2         ; Nollställ I/O-värdet
sendloop:
    jsr    delay      ; Återställ d1
    move.l d3,d1      ;

    jsr    setio
    eor.b  #1,d2      ; Toggla I/O-värdet

    add.l  #-1,d0     ; Färdiga?
    bne    sendloop

    movem.l (a7)+,d0/d2-d3   ; Återställ register och avsluta
    rts
```

Notering: Jag är lite förvånad över att nästan ingen kom ihåg att xor kan användas för att byta tecken på en (eller flera) bitar. Detta kan dock vara bra att lägga på minnet inför framtiden eftersom det gör denna typ av operation mycket smidigare än exempelvis följande kodsnutt:

```
    cmp.l #1,d2
    beq toggle_zero
    move.l #1,d2
    bra after_toggle
toggle_zero:
    move.l #0,d2
after_toggle:
```

Det kan också vara intressant att känna till att det finns ett annat sätt att byta mellan två värden som är lite mer generellt, här illustrerat i C, men i just detta fall är det onödigt att krångla till det för mycket.

```
   int value1 = 42;
   int value2 = 87;
   int sum = value1+value2;
   int x = value1;

   while(1){
      printf("Value is %d\n", x);
      x = sum - x;  // Denna rad kommer nu att göra så att x
                    // växlar mellan value1 och value2
   }
```

b)

```
parse_pulsedata:
    movem.l d0-d1,-(a7)

    move.w (a0)+,d0          ; Är det den första typen
    cmp.w  #1,d0             ; av datastruktur?
    beq    calldelay

callsendpulse:
    ; Eftersom det inte var en etta antar vi här att
    ; det var en tvåa.
    clr.l  d0     ; Utan dessa vet vi inte säkert att
    clr.l  d1     ; vi inte har skräp i MSB-delarna
    move.w (a0)+,d0    ; när vi anropar sendpulse.
    move.w (a1)+,d1
    call   sendpulse

    bra    epilogue

calldelay:
    move.l (a0)+,d1
    jsr    delay

epilogue:
    movem.l (a7)+,d0-d1
    rts
```

c)

```
send_many_pulses:
    jsr parse_pulsedata
    add.l  #-1,d0
    bne    send_many_pulses
    rts
```

Notering: Varianten ovan hanterar inte att noll pulser ska sändas ut, men eftersom det står i uppgiften subrutinen ska tolka "flera, efter varandra liggandes, datastrukturer" är det ok att anta att `d0`> 0. Annars skulle följande variant kunna användas:

```
send_many_pulses:
    tst.l  d0
    beq    finish_loop
    jsr    parse_pulsedata
    add.l  #-1,d0
    bra    send_many_pulses
finish_loop:
    rts
```

Bonusuppgift 1

Om du tentapluggar och vill ha en lite mer utmanande uppgift skulle du kunna fundera på hur du skulle kunna implementera detta med hjälp av avbrott istället för med en subrutin. Antag då att du har ett så kallat timerinterrupt som du kan konfigurera till att ge avbrott efter x mikrosekunder genom att skriva in värdet x i timer-hårdvarans konfigurationsregister.

Bonusuppgift 2

Vad gör egentligen programmet i b-uppgiften? (Tips: Tänk högtalare.)

Lösningsförslag, fråga 5: Pipelining

a)

Se kurslitteraturen. Här vill jag allra helst se en diskussion om att pipelining gör så att den kritiska vägen mellan två register minskar, vilket gör så att systemets maximala klockfrekvens ökar. Detta leder normalt sett till att prestandan också ökar. Jag ser helst att det finns en figur på detta tema, men andra typer av figurer går också bra, exempelvis en figur där man ser hur flera instruktioner överlappas i processorn.

b)

Se kurslitteraturen. Enklaste sättet att få full poäng är troligtvis att rita en figur som visar en processors pipeline (eller återanvända en figur ifrån deluppgift a) och visa på hur resultatet av en beräkning är färdigt efter det att detta resultat behövs om ett program körs som vanligt.

c)

Här gäller det att identifiera databeroenden som sannolikt ger upphov till datakonflikter. Ett typiskt exempel skulle vara att det finns ett databeroende mellan `move.w (a1)+,d3` och `muls.w d2,d3` eftersom det vanligtvis tar ett tag att ladda värden ifrån minnet.

d)

Se kurslitteraturen. Precis som i b-uppgiften är det enklast att få full poäng genom att rita en figur som illustrerar en processors pipeline (eller återanvända en figur ifrån en tidigare uppgift). I denna figur kan du visa hur instruktionen efter en hoppinstruktion laddas in innan processorn har en chans att upptäcka att det faktiskt var en hoppinstruktion som laddades in. (Alternativt visa på att det i samband med villkorliga hopp troligtvis tar ännu längre tid att räkna ut om hoppvillkoret är sant eller inte.)

e)

Se kurslitteraturen. Exempelvis skulle man kunna diskutera register forwarding, stall av processorn eller branch prediction.

Lösningsförslag fråga 6: Cache

Den här uppgiften påminner väldigt mycket om fallet rotation med 90 respektive 270 grader i lab 5 samt om exemplet med `mult.c` som jag körde på en föreläsning.

Sammanlagt kommer detta program att begära läsning av 16384 bytes (2^{14}). Eftersom cachen är 2^{16} bytes tycks det vid en första anblick vara så att båda programmen borde gå lika snabbt att köra.

Cachevariant 1: Direktmappad cache

Om vi antar att cachen har associativiteten ett (dvs den är direktmappad) kommer adressen att delas upp enligt följande:

N 16	15 $log_2 M$	$(log_2 M) - 1$ 0
Tag	Index	Byte i cacheline

där $N + 1$ är antalet bitar i adressen och M är antalet bytes i en cacheline. På den första läsningen läser vi då in den cacheline som finns på adress 0 in i cachen. Om vi då stegar med steglängden 1024 (2^{10}) kommer vi efter 64 steg hamna på en adress med samma index men olika taggar. Detta leder till att den gamla cachelinen på denna adress kastas ut och en ny cacheline läses in. Samma sak kommer att upprepas under de nästa 63 läsningarna från minnet.

När läsning 128 sedan ska göras ifrån adress 1, 1025, 2049, och så vidare kommer cachemissar att hända för varje minnesläsning, eftersom de cachelines som tidigare laddats in på adress 0, 1024, 2048, och så vidare, då har kastats ut. På dessa 16384 läsningar kommer då lika många cachemissar ha inträffat.

Om det istället är så att steglängden 1000 används kommer det ta betydligt längre tid innan vi kommer tillbaka till samma index. Exakt hur lång tid varierar beroende på vilken cacheline-storlek som används i cachen, men vi kan göra en rimlighetsuppskattning genom att se vad som händer med de 16 lägst signifikanta bitarna i adressen efter cirka 64 iterationer respektive 128 iterationer:

Iteration	Läsadress	Läsadress mod 2^{16}
64	64000	64000
65	65000	65000
66	66000	464
...	...	...
127	127000	61464

Det vill säga, efter 128 iterationer kommer vi inte ha fått en konflikt på adress 0 (såvida inte en cacheline är 512 bytes lång eller längre, vilket är tämligen orealistiskt). När vi sedan läser från adress 1, 1001, 2001, osv kommer vi således att få cacheträffar.

Variant 2: Fullt associativ cache

Om vi istället antar att cachen är fullt associativ kommer adressen att delas upp enligt följande:

N $log_2 M$	$(log_2 M) - 1$ 0
Tag	Byte i cacheline

I detta fall är associativiteten lika stor som antalet cachelines i cachen. Detta innebär att åtkomstmönstret som syns i detta exempel kommer att kunna hanteras så länge det finns minst 128 cachelines i cachen (vilket kommer att vara fallet såvida inte orimligt långa cachelines används).

Slutsats

Om processor 1 använder en fullt associativ cache och processor 2 använder en direktmappad cache kan detta fenomen uppstå.

En annan variant (överkurs)

För er som är nyfikna på hur en GPU fungerar så kanske det kan vara intressant att känna till att dessa har en speciell cache som är optimerad för att göra textur-uppslagningar. I dessa fall är man vanligtvis mycket intresserad av att man exempelvis kan göra läsningar både horisontellt och vertikalt i texturen (dvs samma typ av problem vi ser i denna uppgift samt i rotationsdelen av cachelaborationen).

I detta fall kan man tänka sig att cachen är konstruerad så att adressen inte delas upp på ett traditionellt sätt utan exempelvis enligt följande idé:

Y-koordinat			X-koordinat		
Tag (MSB)	Index (MSB)	Byte i cacheline (MSB)	Tag (LSB)	Index (LSB)	Byte i cacheline (LSB)

Detta skulle innebära att antingen vi stegar horisontellt eller vertikalt i bilden så kommer vi att ha en mindre risk för att få konflikter än om adressen delas upp på traditionellt sätt. (Med denna uppdelning kommer en cacheline inte heller att representera en del av en rad i bilden utan en liten fyrkant av bilden istället, vilket ofta reducerar antalet cachemissar något i grafiksammanhang.)

Varför ändrar vi inte på textur-storleken?

Vän av ordning kan då fråga sig varför man i en GPU inte väljer att ha en textur-storlek som inte är jämnt delbar med en tvåpotens. Anledningen till detta är att det är vanligt att koordinater i texturen är större än texturens storlek, vilket är bra då en textur ska upprepas flera gånger på en större yta. För att få reda på vilken pixel programmet egentligen vill adressera i detta fall behöver man ta X-koordinaten respektive Y-koordinaten modulo bredd respektive höjd. Om bredden respektive höjden är en tvåpotens är detta trivialt då det bara handlar om att maska bort de översta bitarna i X- respektive Y-koordinaten. Om bredd eller höjd inte är en tvåpotens krävs istället att en modulo-operation görs i hårdvara, vilket inte är önskvärt då en sådan enhet tar relativt stor plats i hårdvara.