Tentamen
Datorteknik Y, TSEA28

Datum	2013-05-28
Lokal	TER1 och TER2
Tid	14-18
Kurskod	TSEA28
Provkod	TEN1
Kursnamn	Datorteknik Y
Institution	ISY
Antal frågor	6
Antal sidor (inklusive denna sida)	12
Kursansvarig	Andreas Ehliar
Lärare som besöker skrivsalen *Telefon under skrivtiden*	Andreas Ehliar
Besöker skrivsalen	Cirka 15 och 17
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(6p)

(a) (1p) Förklara vad skillnaden på ett aritmetiskt och ett logiskt högerskift är.

(b) (3p) Linus har försökt sig på att multiplicera talen -3 och 5 med hjälp av binär tvåkomplementsaritmetik på följande sätt:

```
 0101  (5) = 4 + 1
*1011  (-3) = (-1)*(2+1)
-----
 0101
0101
0000
+0101
--------
0110111  (32+16+4+2+1) = 55 ?!?
```

Förklara för Linus vad han har gjort för fel samt visa hur denna uträkning egentligen ska gå till.

(c) (2p) Förklara hur N, Z, och C-flaggan genereras i samband med en addition i en typisk aritmetisk logisk enhet (ALU).

Fråga 2: Mikroprogrammering(9p)

I figur 1 återfinns en bekant datormodell. Jämfört med den modell ni sett tidigare har följande förändring gjorts: Mikroprogrammeringsordet har utökats med ytterligare en bit (kallad R). Om R=0 fungerar datorn precis som tidigare. Om R=1 sätts styrsignaler som styr multiplexern vid GR0-GR3 till 3 (dvs register GR3 väljs), oavsett vad IR innehåller.

Notera att det är viktigt att du skriver vilken additionsoperation du valt (den som påverkar flaggorna eller den som inte påverkar flaggorna).

(a) (1p) Skriv mikrokoden för hämtfasen.

(b) (4p) Skriv mikrokoden för följande adresseringslägen samt beskriv hur K2 ska fyllas i:[1]

Adresseringsläge	Assemblerexempel
Omedelbar operand (*immediate*)	LOAD #värde,GR1
Postinkrement av GR3	LOAD (GR3)+,GR1
Preinkrement av GR3	LOAD -(GR3),GR1
Indexerad adressering via GR3	LOAD förskjutning(GR3),GR1

(c) (4p) Skriv mikrokoden för följande instruktioner:[2]

Instruktion	Operation	Påverkar flaggor
Ladda från minne	GRx := PM(M)	Z, N
Addition	GRx := GRx + PM(M)	C, Z, O, N
Mättande addition	GRx := MÄTTAD(GRx + PM(M))	C[1] , Z, O, N

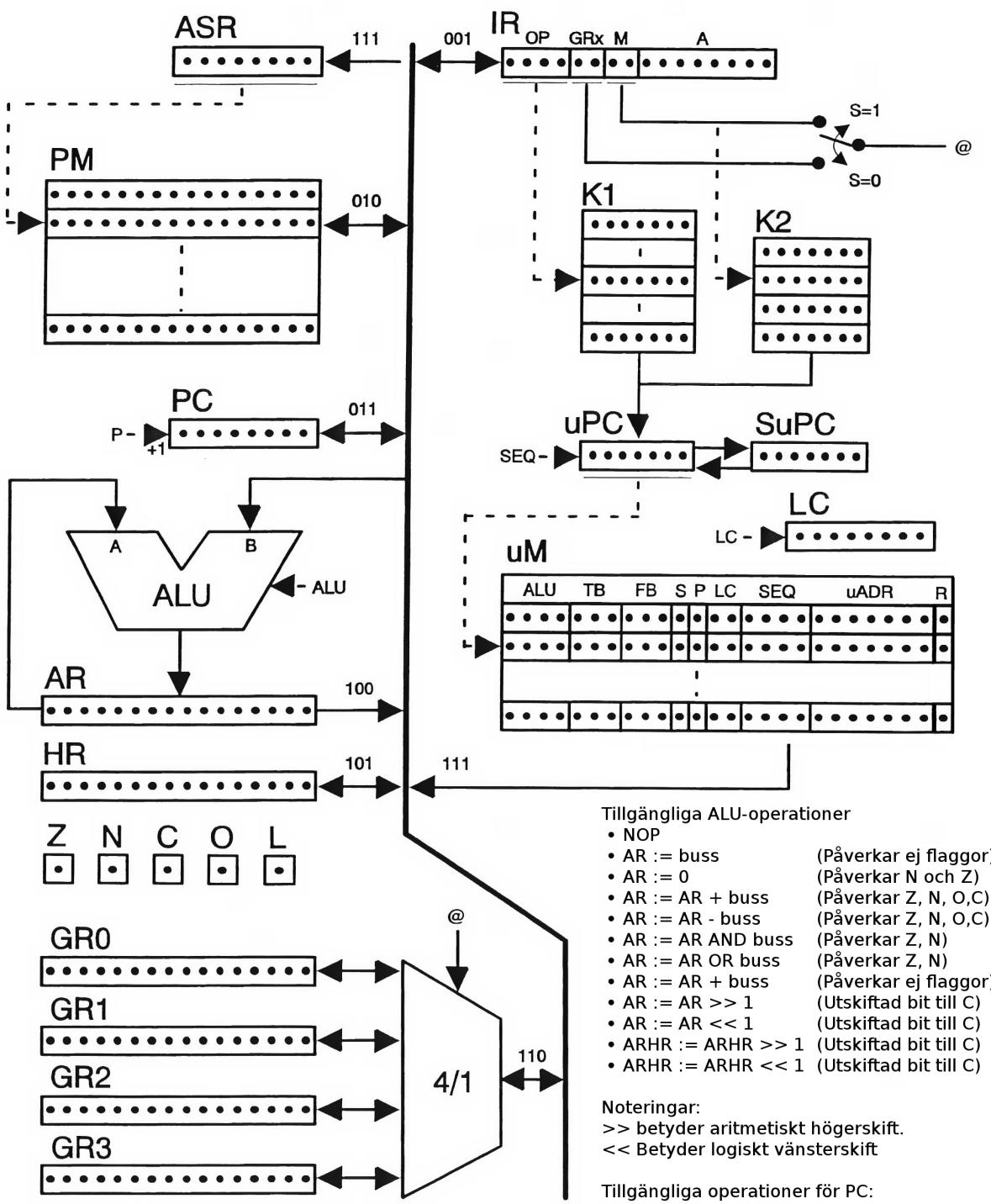
Funktionen MÄTTAD (*saturated*) fungerar så att den i samband med att en tvåkomplementsoperation hamnar utanför talområdet ser till så att svaret blir så nära det korrekta resultatet som möjligt. Det vill säga, i detta fall mättar den resultatet till antingen det största eller det minsta värde som går att representera med ett 16 bitars tvåkomplementstal. Mer formellt kan den definieras på följande sätt:

$$\text{MÄTTAD}(x) = \begin{cases} 2^{15}-1 & \text{om } x \geq 2^{15} \\ -2^{15} & \text{om } x \leq -2^{15} \\ x & \text{annars} \end{cases}$$

[1] Egentligen vill vi inte ändra på C-flaggan här, men datormodellen tillåter inte att vi på ett enkelt sätt ändrar på enbart Z, O, och N. På grund av detta får du för instruktionen mättande addition sätta C-flaggan på valfritt sätt.

[1]Errata: Uppgiften är tyvärr ambivalent huruvida det handlar om preinkrement eller predekrement. Som jag sade på tentan så godkänns båda varianterna.

[2]Errata: Alla hopp på flaggor har ej listats, men som jag sa i samband med tentatillfället får alla typer av hopp användas.



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 C är 1, annars uPC+1
- uPC := uADR om L är 1, 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 3: Allmän teori(12p)

(a) (1p) Varför består primärminnen idag nästan uteslutande av DRAM?

(b) (2p) Förklara konceptet sidindelat primärminne samt nämn minst en anledning till varför man vill ha det i ett givet datorsystem.

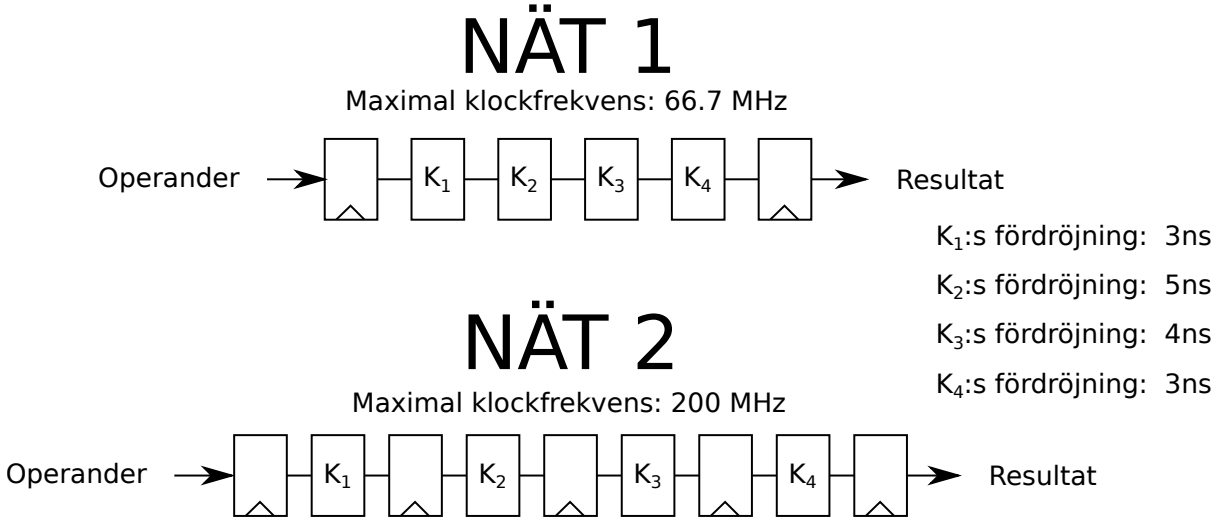
(c) (1p) Om du kör följande programsnutt på TUTOR-systemet kommer du att få en address trap error. Varför?

```
    move.l #$8000,a0
    move.l #$0,d0
loop:
    add.l (a0),d0
    add.l #1,a0
    cmp   #$8100,a0
    bne   loop
```

(d) (4p) Förklara hur en DSP-processor använder sig av flera dataminnen och en MAC-enhet (*multiply and accumulate*) för att utföra operationer såsom filtrering och skalärprodukter på ett effektivt sätt jämfört med en processor där dessa finesser inte finns. Rita väldigt gärna en enkel figur för att förtydliga din förklaring. Diskutera hur mycket snabbare en skalärprodukt kan utföras på en sådan processor jämfört med en processor som inte har dessa finesser (exempelvis en hypotetisk MC68008-kompatibel processor som kan köra en instruktion per klockcykel).

(e) (2p) Diskutera minst en fördel samt en nackdel med en A/D-omvandlare implementerad som en flash-omvandlare respektive en A/D-omvandlare implementerad med hjälp av successiv approximation.

(f) (2p) I figur 2 nedan syns två digitala kretsar samt de fördröjningar olika delar av näten ger upphov till. I det nedre nätet har en viss metod använts för att öka systemets klockfrekvens. Diskutera under vilka omständigheter denna metod faktiskt ökar systemets totala prestanda.



Figur 2: Två digitala nät med samma funktion men implementerade med olika fördröjningar.

Fråga 4: Cache(10p)

En viss cache innehåller 8192 (2^{13} bytes). Den är gruppassociativ och har två vägar. En cacheline är 16 bytes. Du kan anta att cachen är tömd (flushad) i början på varje fråga nedan. 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, etc.

(a) (3p) Ett visst program läser 20 bytes ifrån minnet på följande adresser:

Läsning nummer 0–4:	$1050	$1055	$105a	$105f	$2000
Läsning nummer 5–9:	$2005	$200a	$200f	$2160	$6ffc
Läsning nummer 10–4:	$6ffd	$6ffe	$6fff	$1053	$1AB0
Läsning nummer 15–19:	$1BB0	$7000	$7001	$7002	$7003

Beskriv tillståndet hos alla cachelines i cachen efter att dessa byteläsningar har utförts.

(b) (3p) Förklara vad som händer på systembussen som är kopplad mellan cachen och primärminnet för varje läsning som sker ovan.

(c) (4p) Linnea har fått i uppdrag att modernisera ett program som tidigare har körts på en äldre arbetsstation utrustad med en fullt associativ cache. Tyvärr visar det sig att detta program i vissa lägen går mycket långsammare än väntat när det körs på en modern dator med cacheparametrar enligt ovan. Speciellt långsamt går det i samband med att ett minnesåtkomstmönster i stil med nedan används:

Läsning nummer 0–99	$0000	$0400	$0800	$0c00	$1000	$1400	$1800	...
Läsning nummer 100–199	$0004	$0404	$0804	$0c04	$1004	$1404	$1804	...
Läsning nummer 200–299	$0008	$0408	$0808	$0c08	$1008	$1408	$1808	...
Läsning nummer 300–399	$000c	$040c	$080c	$0c0c	$100c	$140c	$180c	...
Läsning nummer 400–499	$0010	$0410	$0810	$0c10	$1010	$1410	$1810	...
Läsning nummer 500–599	$0014	$0414	$0814	$0c14	$1014	$1414	$1814	...
		Och så vidare...						

Förklara för Linnea vad orsaken är att dessa minnesläsningar går mycket långsammare när det körs på en dator med en gruppassociativ cache än på en dator med fullt associativ cache.

Fråga 5: Assemblerprogrammering(8p)

Skriv en subrutin som implementerar följande skalärprodukt av vektorerna x och y som vardera innehåller N stycken 16-bitars tvåkomplementsrepresenterade heltal.

$$S = \sum_{i=0}^{N-1} x_i \cdot y_i \tag{1}$$

Subrutinen använder register på följande sätt:

- Subrutinen anropas med A0.L satt till adressen för första värdet i x
- Subrutinen anropas med A1.L satt till adressen för första värdet i y
- Subrutinen anropas med D2.L satt till N
- Resultatet S ska hamna i register D0.L (som ett 32-bitars tvåkomplementstal)
- Om inget fel uppstod under beräkningen ska D1.B innehålla värdet 0. Om däremot S någon gång under beräkningen hamnar utanför det talområde som ryms i D0.L ska D1.B innehålla värdet 1. I detta fall är värdet i D0.L oväsentligt.

Det kan vara bra att känna till följande detaljer om MC68000:

- Multiplikation av två sexton bitars tvåkomplementstal görs med hjälp av instruktionen `MULS`. Exempel: Instruktionen `MULS.W D0,D1` multiplicerar D0.W och D1.W och lägger resultatet som ett 32-bitars tvåkomplementstal i D1.
- Hoppinstruktionen `BVS` hoppar om V-flaggan är satt.
- Hoppinstruktionen `BVC` hoppar om V-flaggan är nollställd.

Fråga 6: Stack och programförståelse(5p)

Detta är en disassemblering av ett program som kan köras på TUTOR-systemet.

```
001000    4EBA000A            JSR     $0000100C(PC)
001004    5888                ADDQ.L  #4,A0
001006    B3C8                CMP.L   A0,A1
001008    66F6                BNE.S   $001000
00100A    4E75                RTS

00100C    2F00                MOVE.L  D0,-(A7)
00100E    2010                MOVE.L  (A0),D0
001010    4A80                TST.L   D0
001012    6704                BEQ.S   $001018
001014    81C2                DIVS.W  D2,D0
001016    2080                MOVE.L  D0,(A0)
001018    201F                MOVE.L  (A7)+,D0
00101A    4E75                RTS
```

Antag att följande gäller när subrutinen på adress $1000 precis anropats (men innan den första instruktionen i subrutinen börjat utföras):

- A0 innehåller värdet $4000
- A1 innehåller värdet $4008
- A7 innehåller värdet $6800
- D2 innehåller värdet $0000000f
- D0 innehåller värdet $11223344
- På adress $4000 och framåt finns följande hexadecimala värden: 00 00 05 30 FF FF F0 30 00 00 11 90

Beskriv vad som händer på stacken (och varför) från och med det att subrutinen på adress $1000 anropas tills dess att denna subrutin avslutas.

Kort repetition av M68000

En M68008 har följande register: D0-D7, A0-A7 samt SR. Ibland används bara delar av dessa register. I dessa fall syftar D0.B på de 8 minst signifikanta bitarna, D0.W på de 16 minst signifikanta bitarna och D0.L syftar på hela registret.

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

1a)

Se lärobok

1b)

Det första felet som gjorts är att Linus har representerat -3 med tecken-belopp istället för tvåkomplement. (Sedan är även multiplikationen felaktigt utfört eftersom Linus inte tagit hänsyn till tecknet på något sätt.)

```
       0101  (5) = 4 + 1
      *1101  (-3) = -8 + 4 + 1
      -----
       0101
      0000
     0101
  +11011    (dvs -0101 samt teckenutökat till 8 bitar)
   --------
   11110001  (-128 + 64 + 32 + 16 + 1) = -15
```

1c)

Se lärobok. Notera att det enbart är operationen addition som efterfrågas uppgiften. (Till skillnad från exempelvis logiskt skift.)

Fråga 2

2a)

Se labmanualen för mikrokodningslaborationen.

2b)

Se labmanualen för omedelbar operand och indexerad adressering.

```
Postinkrement:

    uPC antas räknas upp med ett om inget annat anges:
    R:=1, ASR := GRx, AR := GRx
    AR := AR + 1 (ettan ifrån uM, additionen ändrar ej flaggor)
    R:=1, GRx := AR, uPC := K1(OP)

Predekrement:
    R:=1, AR = GRx
    AR := AR + \$FFFF (-1 ifrån uM, additionen ändrar ej flaggor)
    R:=1, GRx = AR
    ASR = AR, uPC := K1(OP)
```

En fiffig variant som en tentand hittade på i samband med postinkrement och predrekement var att anta att adress-fältet i instruktionen innehöll inkrementet, vilket ökar flexibiliteten hos dessa adresseringslägen.

Kommentar: Tyvärr har alldeles för många glömt att ange om flaggorna ändras eller inte i samband med additionsoperationen.

2c)

```
Ladda (samt sätta flaggorna Z och N):
    AR := 0
    AR := AR or PM, GRx := PM, uPC := 0

Addition:
    AR := PM
    AR := AR + GRx
    GRx := AR, uPC := 0

Mättande addition:
    AR := PM
    AR := AR + GRx
    uPC := hantera_overflow om O = 1, annars uPC++
    GRx := AR, uPC := 0
hantera_overflow:
    AR := 0, uPC := positivt om N = 1, annars uPC++
    AR := AR or \$8000 (från uM)
    GRx := AR, uPC := 0
positivt:
    AR := AR or \$7fff (från uM)
    GRx := AR, uPC := 0
```

Kommentar: Det står inte uttryckligen i uppgiften exakt hur flaggorna ska sättas, men för att dessa ska sättas vettigt även vid overflow är det nödvändigt att använda en operation som sätter flaggorna på ett rimligt sätt när AR laddas in med ett värde ifrån bussen. (Det här var nog den del som flest missade på, men eftersom det inte uttryckligen står i uppgiften hur flaggorna ska sättas så har detta ignorerats i rättningen.)

Fråga 3

3a)

För att DRAM utgör en bra kompromiss mellan snabbhet och lagringstäthet. Framförallt är det betydligt billigare än SRAM.

3b)

Se lärobok.

3c)

För att programmet försöker läsa ett 4 bytes ord ifrån en udda adress (adress $8001).[3]

Kommentar: Ett vanligt fel här är att påpeka att RAM i TUTOR-systemet endast sträcker sig till $7fff. Från $8000 och upp till $ffff finns det istället ROM. Detta är dock inte anledningen till avbrottet eftersom det går alldeles ypperligt att läsa ifrån ROM.

[3]För er som tycker att det här är en relativt nischad fråga så är detta faktiskt en relativt vanlig fallgrop när man i ett högnivåspråk vill tolka en binär datastruktur, exempelvis ett paket som kommit in ifrån nätverket. Enligt exempelvis C-standarden är det ej tillåtet att läsa ett 16-bitars ord ifrån en udda adress samt ett 32-bitars ord ifrån en adress som ej är jämnt delbar med 4. Om man ändå skulle göra detta är det upp till vilken processor (och kompilator) man använder om det fungerar eller inte. På X86 kommer det antagligen att fungera (men gå långsamt) men på en del andra processorer (främst processorer avsedda för inbyggda system) kommer du att få en exception istället. Slutligen finns det vissa system som erbjuder utvecklaren timtals med rolig felsökning eftersom load-instruktionen i detta fall helt enkelt returnerar fel värde ...

3d)

För full poäng här ska det framgå tydligt att DSP-processorn har dataminnen direktkopplade till MAC-enheten så att det är möjligt att få en genomströmning på en MAC-instruktionen per klockcykel. Detta görs lättast genom en illustration där dataminnen, multiplicerare, adderare, samt ackumulatorregister ingår.

Det ska också framgå tydligt vilken uppsnabbning som de efterfrågade finesserna ger, exempelvis genom att diskutera skillnaden mellan den inre loopen på en MC68008-liknande processor och en DSP-processor. På MC68008 behöver exempelvis följande saker göras i den inre loopen i samband med en skalärmultiplikation, vi antar också att varje rad tar en klockcykel att utföra:

MAC Ladda från vektor 1

MAC Ladda från vektor 2

MAC Multiplicera elementen från vektorerna

MAC Ackumulera resultatet från multiplikationen till lämpligt ackumulatorregister

- Räkna ner loopräknare
- Villkorligt hopp till början av den inre loopen

På en typisk DSP-processor kan raderna som är markerade med MAC ersättas med endast en instruktion. Detta medför att antalet instruktioner som körs i den inre loopen minskar till hälften och funktionen blir då ungefär dubbelt så snabb.[4]

3e)

Se kurslitteraturen. Ett vanligt missförstånd (som dock ej gett poängavdrag) tycks för övrigt vara att successiv approximation med nödvändighet måste göras i mjukvara. I själva verket är det vanligast att även successiv approximation är implementerat i hårdvara.

3f)

Här vill jag att det tydligt ska framgå att pipelining endast lönar sig i de fall där inkommande operander är oberoende av resultatet ifrån beräkningar som påbörjats i pipelinen men ännu ej slutförts.

[4]Detta är en sanning med modifikation, då vår fiktiva MC68008 enligt uppgiften kan köra en instruktion per klockcykel och således alltså kan köra instruktioner som `MULT.W (A0)+,D0` samt `DBRA` på endast en klockcykel. Å andra sidan har en DSP-processor vanligtvis en loop-instruktion också för att snabba upp den inre loopen ytterligare. Det kom också in några svar som antog att det fanns fler än 2 dataminnen, vilket givetvis kan ge ytterligare uppsnabbning. Det finns således lite varianter här på hur stor uppsnabbning som sker, beroende på vilka antaganden som görs om MC68008 respektive DSP-processorn, men så länge antaganden är rimliga så är det ok.

Fråga 4

4a)

Då en cacheline är 16 byte och det finns 2 vägar är det sammanlagt $8192/16/2 = 256 = 2^8$ cachelines per väg. Detta medför att adressen kan delas upp på följande sätt:

Bit nummer	?–12	11–4	3–0
Användning	Tag	Index	Byte

(De flesta har antagit att adressen är 16 bitar bred, men uppgiften ändras inte om man antar att adressen är bredare än så, exempelvis 32 bitar.)

Index	Tag	Innehåll från adress	Tag	Innehåll från adress
$00	$2	$2000–$200F	$7	$7000-$700f
$05	$1	$1050–$105F	-	*Empty/Invalid*
$16	$2	$2160–$216F	-	*Empty/Invalid*
$AB	$1	$1AB0–$1ABF	-	*Empty/Invalid*
$BB	$1	$1BB0–$1BBF	-	*Empty/Invalid*
$FF	$6	$6FF0–$6FFF	-	*Empty/Invalid*

4b)

Läsning nr.	Aktivitet på systembussen
0	$1050–$105F begärs från primärminnet
1	—
2	—
3	—
4	$2000–$200F begärs från primärminnet
5	—
6	—
7	—
8	$2160–$216F begärs från primärminnet
9	$6FF0–$6FFF begärs från primärminnet
10	—
11	—
12	—
13	—
14	$1AB0–$1ABF begärs från primärminnet
15	$1BB0–$1BBF begärs från primärminnet
16	$7000–$1700 begärs från primärminnet
17	—
18	—
19	—

Ett relativt vanligt missförstånd här är att man i allmänna ordalag har beskrivit vad som händer på bussen utan att specifikt ange vad som händer på bussen för varje läsning i 4a. Detta är dock inget större problem eftersom de mellansteg som behövs för att lösa 4a duger som redovisning av 4b.

4c)

För full poäng vill jag att det ska framgå att anledningen till att det går långsamt är att det blir cachemissar hela tiden när detta program körs. Eftersom cachelines med samma index ($00, $40, $80 och $C0) fast med olika tag används under läsning 0-99 kommer den data som läses in vid läsning 0 bli utkastad innan den behövs igen vid läsning 100, 200 samt 300. Samma sak vid läsning 1 och läsning 101, 201 samt 301, och så vidare. Det ska också framgå att en fullt associativ cache (givet att den är tillräckligt stor) inte har detta problem eftersom data som hör till en viss adress i primärminnet kan läggas i godtycklig cacheline. Ett alternativt sätt att se detta på är att en fullt associativ cache har enbart ett index men N vägar (där N är antalet cachelines i cachen).

Kommentar: Se även lab 5 där exakt samma fenomen gör att bildrotation går långsamt vid 90 respektive 270 grader.

Fråga 5

```
    clr.l  d0
loop:
    move.w (a0)+,d1          ; Alternativt   move.w (a0)+,d3
    move.w (a1)+,d3          ; muls.w (a1)+,d3
    muls.w d1,d3             ;
    add.l  d3,d0
    bvs    fick_overflow
    add.l  #-1,d2
    bne    loop
    move.b #0,d1
    rts
fick_overflow
    move.b #1,d1
    rts
```

Kommentar: Det vanligaste felet är antagligen att move.l har använts istället för move.w. Eftersom vi programmerar på papper istället för på ett datorsystem är det dock ok att ha ett fel av slarvkaraktär utan att några poäng dras. (Är det mer än ett slarvfel dras dock poäng som vanligt.)

Fråga 6

PC	Stackpekarens nya värde	Vad som händer	Orsak
`$1000`	`$67fc`	$00001004 pushas på stacken	Återhoppsadressen för JSR
`$100C`	`$67f8`	$11223344 pushas på stacken	D0 skrivs till stacken
`$1018`	`$67fc`	$11223344 popas från stacken	Återställer D0:s värde
`$101A`	`$6800`	$00001004 popas från stacken	RTS läser återhoppsadressen
(`$1008`)	Operationerna ovan upprepas ytterligare en gång eftersom A0 i detta läge ej nått $4008		
`$100A`	`$6804`	32 okända bitar popas från stacken	RTS läser återhoppsadressen

Kommentar: Uppgiften var noga specificerad så att divisionsinstruktionen ej får in en nolla, vare sig i nämnaren eller täljaren. (Uppgiften blir betydligt mer spännande om det kommer in en nolla i nämnaren eftersom vi då kommer att få ett avbrott.)

Statistik

Betygsfördelningen såg ut såhär:

- U: 34
- 3: 43
- 4: 29
- 5: 9 (bästa resultat var 47 poäng)

Medelpoäng per uppgift:

1: 4.6 (av 6)

2: 2.7 (av 9)

3: 5.9 (av 12)

4: 3.6 (av 10)

5: 6.5 (av 8)

6: 2.4 (av 5)