Tentamen
Datorteknik Y, TSEA28

Datum	2011-08-16
Lokal	
Tid	8:00-12:00
Kurskod	TSEA28
Provkod	TEN 1
Kursnamn	Datorteknik Y
Institution	ISY
Antal uppgifter	9
Antal sidor (inklusive denna sida)	7
Kursansvarig	Andreas Ehliar
Lärare som besöker skrivsalen	Olle Seger
Telefon under skrivtid	013 - 28 2159
Besöker skrivsalen ca kl	9 och 11
Kursadministratör	Ylva Jernling, 013-282648, ylva@isy.liu.se
Tillåtna hjälpmedel	Inga
Betygsgränser	**Poäng** **Betyg** 41-50 5 31-40 4 21-30 3 0-20 U

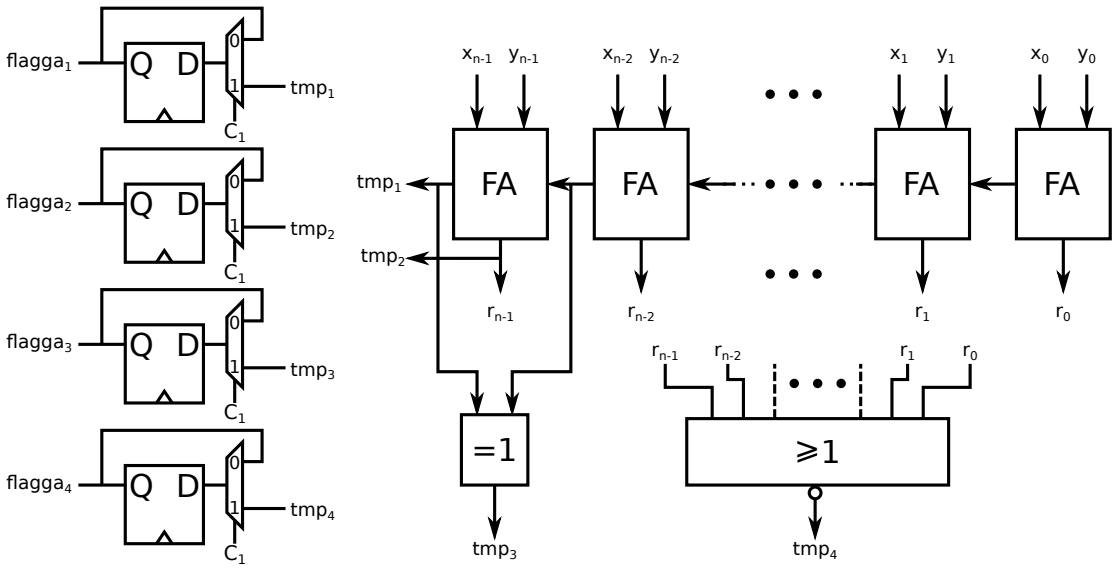
Viktig information

- Normalt rättas en tenta inom 10 arbetsdagar. Den här gången är det dock sannolikt att det kommer att ta lite längre tid att rätta tentan på grund av föräldraledighet. *För mer info om när tentan beräknas vara färdigrättad, se kurshemsidan.*
- 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!

Uppgift 1 (7p)

Figur 1 är hämtad ifrån ett kretsschema på en ALU med n bitar. Denna enhet innehåller de flaggor som vanligtvis brukar vara synliga för en programmerare.



Figur 1: En ALU med några mystiska flaggor.

a) Vilken eller vilka matematiska operationer utför denna enhet på signalerna x och y? Vilken funktion har signalen C_1? (1p)

b) Tyvärr har personen som ritat kretsschemat gått på semester innan dokumentationen blev färdigskriven. Namnge flagga_1 till flagga_4 och beskriv med en mening per flagga dess huvudsakliga syfte. (4p)

c) Antag att den aritmetiska enheten i figuren ovan är 8 bitar bred (dvs n är 8). $x = 11011011$, $y = 10011001$ och $C_1 = 1$. Vad blir r och vilka värden får de olika flaggorna? (2p)

Uppgift 2 (6p)

a) Förklara kortfattat konceptet pipelining (*överlappning i Roos*). Rita en figur och använd max 5 meningar för att förklara varför pipelining i många fall kan öka prestandan på ett digitalt system. (2p)

b) Ett problem som kan uppstå när man använder pipelining i en dator är en så kallad strukturell konflikt (*structural hazard*). Förklara vad detta är med max 5 meningar. Rita gärna en figur. (2p)

c) Ett annat problem som kan uppstå när man använder pipelining är en så kallad styrkonflikt (*control hazard*). Förklara vad detta är med max 5 meningar. Rita gärna en figur. (2p)

Uppgift 3 (10p)

I figur 2 på nästa sida återfinns arkitekturen för den dator ni använde i mikroprogrammeringslaborationerna. Ni får använda alla operationer som finns med i labdatorn, men ni bör inte behöva några fler operationer utöver de som är listade i figuren.

a) En programmerare behöver på ett effektivt sätt räkna hur många binära nollor det finns i ett 16 bitars tal. Han vill gärna ha följande instruktion:

```
COUNT0 GRx, Operand   ; Count zeroes
```

Denna instruktion ska räkna antalet nollor i operanden och skriva in resultatet i GRx. Ett par exempel på hur instruktionen ska fungera följer nedan:

```
COUNT0 GR0, #$f001    ; Returnerar elva i GR0
COUNT0 GR0, #$ffff    ; Returnerar noll i GR0
COUNT0 GR0, #$0000    ; Returnerar sexton i GR0
```

Skriv mikrokoden för denna instruktion! (Du behöver ej skriva hämtfas och adresseringsfas.) (4p)

b) En annan programmerare behöver på ett effektivt sätt räkna hur många konsekutiva binära nollor det finns i den minst signifikanta delen av ett 16 bitars tal. Hon vill gärna ha följande instruktion:

```
FL1 GRx, Operand    ; Find last one
```

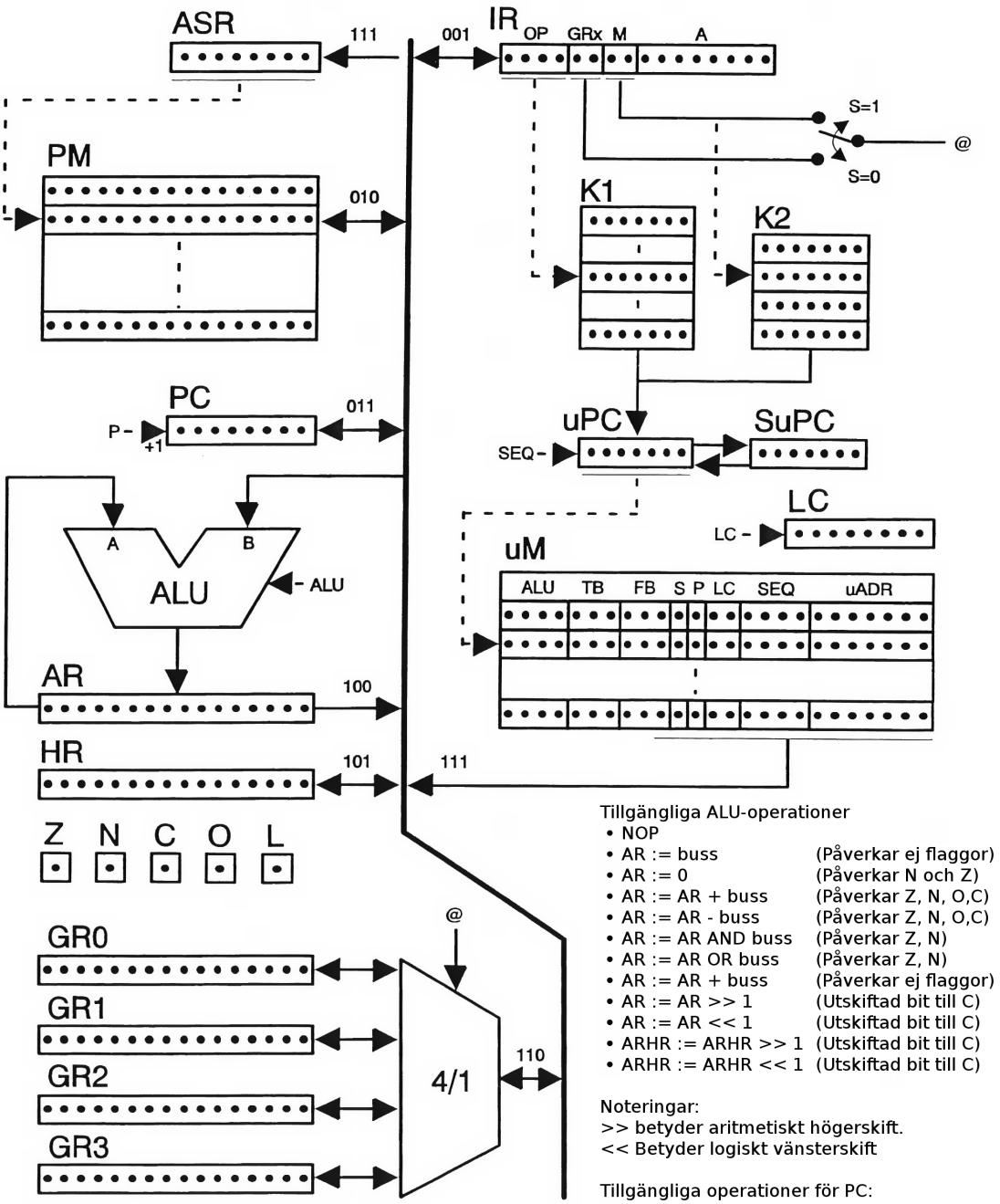
Denna instruktion ska börja på den minst signifikanta biten i operanden och räkna hur många binära nollor som finns lagrade där innan den första ettan dyker upp. Några exempel på hur instruktionen ska fungera följer nedan:

```
FL1 GR0, #$f001     ; Returnerar 0 i GR0 då det är noll stycken
                    ; nollor i LSB-delen av operanden
FL1 GR0, #$0090     ; Returnerar 4 i GR0 då det är fyra nollor
                    ; i LSB-delen av operanden
FL1 GR0, #$f020     ; Returnerar 5 i GR0 då det är fem nollor i
                    ; LSB-delen av operanden
FL1 GR0, #$0000     ; Returnerar 16 i GR0 då det är sexton
                    ; nollor i LSB-delen av operanden
```

Skriv mikrokoden för denna instruktion! (Du behöver ej skriva hämtfas och adresseringsfas.) (6p)

Du behöver inte överföra mikrokoden till binär form, det räcker att skriva mikrokoden enligt följande exempel:

Adress 54: Buss := IR, ASR := Buss, PC := PC + 1, AR := AR – buss, uPC := uPC + 1
Adress 55: Buss := AR, GRx := Buss, uPC := 0



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

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.

Figur 2: En välkänd mikroprogrammerad dator. (Björn Lindskog 1981)

Uppgift 4 (5p)

Din chef har i sin oändliga vishet beslutat sig för att använda datormodellen i figur 2 på föregående sida till sitt nästa stora projekt.[1] Tyvärr märkte din chef aldrig att datormodellen i figur 2 inte har någon möjlighet till in- eller utmatning.

Din uppgift är nu att lägga till dessa I/O-portar till detta system:

- **PORTU:** En parallellport med 16 utsignaler från systemet
- **PORTI:** En parallellport med 16 synkroniserade insignaler till systemet

a) Ett sätt att implementera detta på är genom att lägga till två instruktioner till systemet:

```
OUT Operand          ; PORTU sätts till värdet på operanden
IN GRx               ; GRx sätts till värdet som ligger på PORTI
```

Beskriv hur du skulle modifiera datormodellen i figur 2 för att lägga till dessa instruktioner. Rita en figur/blockschema! Skriv mikrokoden för OUT respektive IN.(2p)

b) Ett annat sätt att implementera detta på är genom att använda minnesmappad (*memory mapped*) I/O. Beskriv hur du skulle modifiera datormodellen i figur 2 för att lägga till **PORTU** och **PORTI** med hjälp av minnesmappad I/O. Rita en figur/blockschema! Ange alla modifikationer du behöver göra på mikrokoden. (3p)

Uppgift 5 (3p)

Multiplicera talen -6 och -10 med varandra med hjälp av valfri algoritm för att multiplicera binära tal. Negativa tal ska representeras med hjälp av tvåkomplement!

Uppgift 6 (4p)

Beskriv hur en A/D-omvandlare som arbetar enligt principen successiv approximation fungerar. Rita ett blockschema och ett flödesschema!

Uppgift 7 (3p)

a) Vilken typ av cache är sannolikt bäst för prestandan i ett datorsystem utav en direktmappad, flervägsassociativ eller fullt associativ cache? (Under förutsättningen att cachearna i övrigt har samma parametrar, det vill säga, de rymmer lika mycket data, de har samma klockfrekvens, och så vidare.) (1p)

b) En modern processor som fungerar enligt RISC-principen har vanligtvis både en instruktionscache och en datacache. Vilken av dessa typer av cache:ar är viktigast för prestandan? Motivera ditt svar! (2p)

[1] Sannolikt är en slarvig kravspecifikation i kombination med en pubrunda orsaken till detta beslut...

Uppgift 8 (4p)

Du håller på att köpa in ett nätverkskort och ett grafikkort ifrån företaget *Dyra Komponenter AB*. Du har råd med ett nätverkskort och grafikkort som har stöd för antingen avbrott eller DMA. Tyvärr har du inte råd med lyxvarianterna av dessa kort som har stöd för båda dessa finesser.

a) Diskutera ett scenario där stöd för avbrott gör mest nytta för systemprestandan. (2p)

b) Diskutera ett scenario där stöd för DMA gör mest nytta för systemprestandan. (2p)

Uppgift 9 (8p)

Du har tröttnat på alla mail som examinator för TSEA28 skickar ut till kursmailinglistan. Du vill således sortera bort dessa mail på ett smidigt sätt. Till din hjälp så har du (av lite oklara anledningar) valt att använda en dator baserad på Motorola 68000. (Se nästa sida för en ASCII-tabell samt en kort repetition av de instruktioner som finns på Motorola 68000.)

Tips: *Om du skriver subrutinerna i deluppgift **a** på ett smart sätt så behöver du inte skriva särskilt mycket ny kod i deluppgift **b**.*

a) Skriv en subrutin i som arbetar enligt följande specifikation: (6p)

- I register `a0` finns en pekare till början på en text i ASCII-format.
- I register `a1` finns en pekare till det sista tecknet i denna text.
- I register `d0` ska du returnera 1 om du hittade ordet "TSEA28" *någonstans* i texten, annars ska du returnera 0. Du kan anta att examinator alltid använder stora bokstäver för att skriva "TSEA28".

Du kan ha nytta av att dela upp problemet så att du exempelvis anropar en subrutin (som du själv måste skriva!) som fungerar ungefär på följande sätt:

- I register `a2` finns en pekare till början på en text i ASCII format.
- I register `d0` returneras 1 om denna text *börjar* med texten "TSEA28", annars returneras 0 i `d0`.

b) Du kommer på att du gärna vill de tips inför tentan som examinator kan tänkas skicka ut. Skriv en ny subrutin som fungerar som subrutinen i deluppgift **a**, med undantag för att den ska returnera 0 i `d0` om den hittar både ordet "TSEA28" och ordet "tenta" i texten. (Du kan anta att tenta alltid är skrivet med små bokstäver.) (2p)

Utdrag ur en ASCII-tabell

Kodning	Tecken	Kodning	Tecken	Kodning	Tecken	Kodning	Tecken
48	0	67	C	86	V	105	i
49	1	68	D	87	W	106	j
50	2	69	E	88	X	107	k
51	3	70	F	89	Y	108	l
52	4	71	G	90	Z	109	m
53	5	72	H	91	[	110	n
54	6	73	I	92	\	111	o
55	7	74	J	93	]	112	p
56	8	75	K	94	^	113	q
57	9	76	L	95	_	114	r
58	:	77	M	96	‘	115	s
59	;	78	N	97	a	116	t
60	<	79	O	98	b	117	u
61	=	80	P	99	c	118	v
62	>	81	Q	100	d	119	w
63	?	82	R	101	e	120	x
64	@	83	S	102	f	121	y
65	A	84	T	103	g	122	z
66	B	85	U	104	h		

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
    LEA $2000,A0
    LEA $3000,A1
    MOVE.B #50,D0
loop
    MOVE.L (A0)+,(A1)+
    ADD.B #-1,D0
    BNE loop
```

Lösningsförslag

Uppgift 1

a) Enheten utför addition av x och y. Signalen C1 används som load enable för statusflaggorna.

b)

flagga1: Carry - Ettställs för att markera att resultatet av en addition av två binära tal utan tecken blivit för stort för att rymmas i talområdet.

flagga2: Negative - Ettställs för att markera att resultatet är negativt om det tolkas som ett tvåkomplementsrepresenterat tal

flagga3: Overflow - Ettställs om resultatet av en addition av två stycken tvåkomplementsrepresenterade tal blivit antingen för stort eller för litet för att rymmas i talområdet

flagga4: zero - Ettställs om resultatet är noll

c) `r = 11011011 +10011001 = (1)01110100, c=1, n=0, v=1, z=0`

Uppgift 2

a) Pipelining används ofta för att snabba upp prestandan i processorer. Istället för att hämta, avkoda, utföra och skriva tillbaka resultatet för varje instruktion i taget kan man överlappa dessa enligt figuren nedan. Detta innebär att genomflödet av assemblerinstruktioner i datorn teoretiskt kan öka med fyra, under förutsättningen att alla dessa fyra moment tar lika lång tid att utföra.

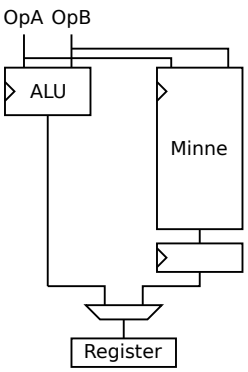
Utan pipelining:	hämta 1	avkoda 1 utför 1		skriv 1	hämta 2	utför 2	...
Med pipelining:	hämta 1	avkoda 1	utför 1	skriv 1			
		hämta 2	avkoda 2	utför 2	skriv 2		
			hämta 3	avkoda 3	utför 3	skriv 3	
				hämta 4	avkoda 4	utför 4	...

b) En strukturell konflikt innebär att processorn försöker använda samma enhet/resurs ifrån olika pipelinesteg. Ett exempel på när detta kan inträffa ses i figuren nedan där följande program kommer att orsaka en strukturell konflikt:

Exempelprogram:

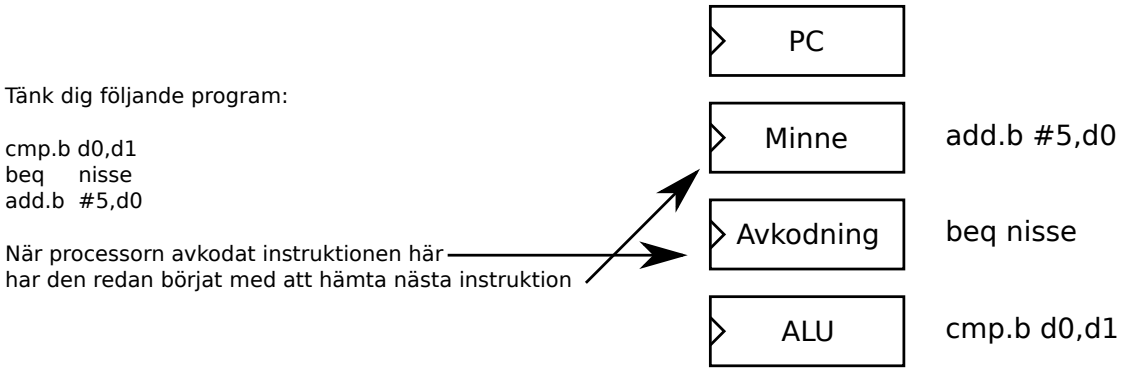
move.b (a0),d1
add.b #0,d0

Båda instruktionerna kommer att vilja skriva till registerfilen samtidigt då minnesläsningen precis hunnit klart samtidigt som ALU-operationen blir klar.



c) En styrkonflikt uppstår på grund av att processorn inte känner till att en viss instruktion är en hoppinstruktion förräns den har hämtats in och avkodats. (Alternativt känner processorn

inte till om hoppvillkoret är sant eller inte.) I detta läge har ett antal instruktioner efter hoppinstruktionen redan hämtats in i processorns pipeline trots att processorn eventuellt måste utföra ett villkorligt hopp, vilket innebär att add-instruktionen i figuren eventuellt inte ska utföras.



Uppgift 3

Den här uppgiften blev aningens klurigare än det var tänkt. I uppgiften står det att man får använda alla operationer som ni använt i laborationerna. Samtidigt står det att man inte bör behöva fler operationer än de som är listade i figur 2. Olyckligtvis så råkade följande operationer utebli ifrån figur 2 för uPC:

- uPC := uADR om C är ett, annars uPC + 1
- uPC := uADR om L är ett, annars uPC + 1

Det var dock ingen som anmärkte på detta i samband med att tentan gick och de flesta som svarat på uppgiften har kommit ihåg att dessa instruktioner finns med i labdatorn. Vi har också varit generösa i vår bedömning av uppgiften och exempelvis inte gjort några avdrag om man tror att det finns en möjlighet att hoppa om C är noll (det vill säga, man har vänt på hoppvillkoret).

Till att börja med så presenterar vi två lösningar som använder dessa instruktioner. Eftersom 3b är värd lite mer har vi varit extra noga med att se till så att det inte blir några så kallade "off by one" fel i era lösningsförslag i 3b.

```
; Alla mikrokodsrader nedan antas innehålla uPC := uPC + 1 om inget
; annat anges för uPC! Notera också att det finns två olika
; additionsinstruktioner, men i dessa program spelar det ingen roll
; vilken som används!
count0_microcode:
    Buss := PM(ASR), HR := Buss, AR := 0, LC := 16    ; Initiera

loop:
    AR := AR << 1 ; Skifta HR höger utan att påverka AR...
    ARHR := ARHR >> 1, uPC := slutloop om L = 1, annars uPC := uPC + 1

    uPC := loop om C = 1, annars uPC := uPC + 1
    Buss := 1, AR := AR + Buss
    uPC := loop, LC := LC - 1
slutloop:

    Buss := AR, GRx := Buss, uPC := 0
```

```
fl1_microcode:
    Buss := PM(ASR), HR := Buss, AR := 0, LC := 16    ; Initiera

loop:
    AR := AR << 1 ; Skifta HR höger utan att påverka AR...
    ARHR := ARHR >> 1, uPC := slutloop om L = 1, annars uPC := uPC + 1

    uPC := slutloop om C = 1, annars uPC := uPC + 1
    Buss := 1, AR := AR + Buss
    uPC := loop, LC := LC - 1

slutloop:
    Buss := AR, GRx := Buss, uPC := 0
```

Uppgift 3a utan villkorliga hopp

Det är faktiskt möjligt att, utan några större besvär, lösa uppgift 3a utan att använda några villkorliga hopp överhuvudtaget. Det största problemet är att programmet blir mycket längre eftersom vi måste "rulla ut" loopen eftersom vi inte har någon nytta av loopräknaren längre.

```
; Alla mikrokodsrader nedan antas innehålla uPC := uPC + 1 om inget annat
; anges för uPC!

count0_microcode_without_branches:
    Buss := PM(ASR), HR := Buss, AR := 0    ; Läs operanden

count0_after_operand_read:                  ; För uppgift 3b
    ; Det tycks lättare att räkna ettor än nollor, så vi
    ; inverterar operanden innan vi gör något mer.
    AR := AR - Buss, Buss := HR
    AR := AR - Buss, Buss := 1

count0_after_inversion:                      ; Se uppgift 3b längre ner...
    Buss := AR, HR := Buss, AR := 0


iteration1:
    ARHR := ARHR << 1 ; Skifta operanden så att MSB-biten hamnar i LSB-delen av AR
    AR := AR + Buss; Buss := 1
    AR := AR >> 1     ; Om MSB-biten av operanden var ett har vi ökat på AR med 1
                          ; Om MSB-biten av operanden var noll har vi inte påverkat AR
slut_iteration1:
    ; Sedan upprepar vi dessa tre mikroinstruktioner mellan
    ; iteration1 och slut_iteration1 15 gånger till i
    ; mikrokodsminnet (eftersom vi inte har någon
    ; loopräknaren vi kan använda)

    GRx := Buss, Buss := AR, uPC := 0 ; Och så är vi klara
```

Uppgift 3b utan villkorliga hopp

b-uppgiften är inte riktigt lika enkel att lösa utan dessa mikroinstruktioner, men det är möjligt att göra detta också enligt nedan:

```
; Alla mikrokodsrader nedan antas innehålla uPC := uPC + 1 om inget annat anges
; för uPC!
fl1_microcode_without_branches:

    AR := 0

iteration:
    Buss := PM(ASR), AR := AR OR Buss
    ARHR := ARHR >> 1
slut_iteration

    ; På samma sätt så upprepar vi här de instruktioner som finns
    ; emellan iteration och slut_iteration 15 gånger till så att vi
    ; sammanlagt "orar" och skiftar AR 16 gånger.
    ;
    ; Då har vi i HR fått en mask som är ett från och med den minst
    ; signifikanta ettan i talet till den mest signifikanta ettan i
    ; talet. Denna kan vi sedan räkna nollorna i med hjälp av count0.

    uPC := count0_after_operand_read, AR := 0
```

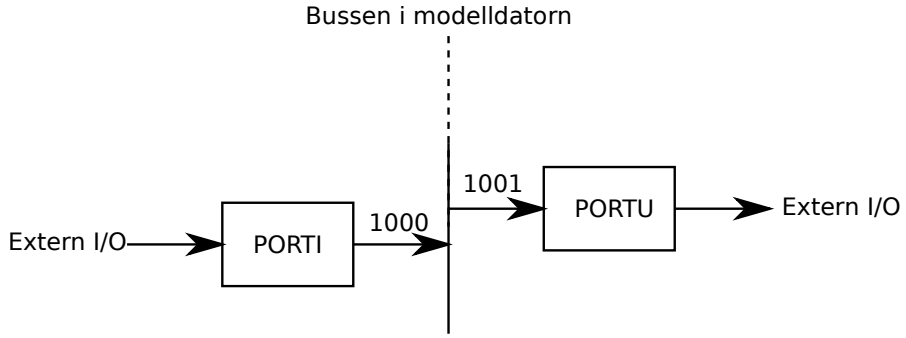
Klurig lösning på uppgift 3b

En lite mer överkursaktig lösning på detta problem följer nedan. Eftersom det är en ganska trevlig övning i binär aritmetik att lista ut varför detta fungerar så uppmanar jag läsaren att försöka reda ut detta själv. (För de som klurat färdigt och vill kontrollera sitt svar så har jag inkluderat en förklaring till detta sist i detta dokument.)

```
better_fl1_microcode_without_branches:
        Buss := PM(ASR), AR := 0, HR := Buss, uPC := uPC + 1
        Buss := HR, AR := AR - Buss, uPC := uPC + 1
        Buss := HR, AR := AR AND Buss, uPC := uPC + 1
        Buss := 1, AR := AR - Buss, uPC := uPC + 1
         uPC := count0_after_inversion
```

Uppgift 4

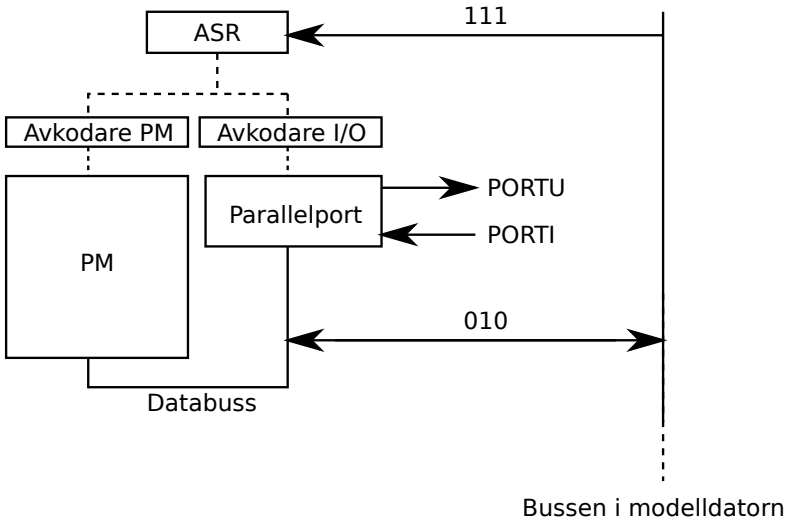
a) Vi lägger till PORTI och PORTU på bussen enligt figuren. (Vi måste sannolikt också ändra på TB och FB-fältet i mikrokoden så att det blir en bit bredare för att få plats med alla enheter...)




```
out_microcode:
        Buss := PM(ASR), PORTU := Buss, uPC := 0

in_microcode:
        Buss := PORTI, GRx := Buss, uPC := 0
```

b) Vi modifierar PM så att PORTI och PORTU hamnar i minnesrymden enligt figur nedan. Mikrokoden behöver ej förändras.



Uppgift 5

					1	0	1	1	0	(-10)
			*			1	0	1	0	(-6)
	0	0	0	0	0	0	0	0	0	$(0 \cdot 2^0)$
+	1	1	1	1	0	1	1	0		$(-10 \cdot 2^1)$
+	0	0	0	0	0	0	0			$(0 \cdot 2^2)$
-	1	1	0	1	1	0				$-(-10 \cdot 2^3)$
	0	0	0	1	1	1	1	0	0	(60)

Uppgift 6

Se kurslitteratur/labkompendium.

Uppgift 7

a En fullt associativ cache är sannolikt bäst för prestandan givet att alla andra parametrar är likvärdiga. (I praktiken är hårdvarukostnaden för en fullt associativ cache nästan alltid för hög för att det ska vara aktuellt att använda en sådan dock.)

b) En instruktionscache är betydligt viktigare för prestandan eftersom den kommer att utnyttjas vid varje instruktion till skillnad ifrån datacachen som bara används i samband med load/store.

Uppgift 8

a) Avbrott gör mest för prestandan i exempelvis ett scenario där ett system utför både beräkningar samt behöver svara snabbt på enstaka I/O-händelser. Det är då inte särskilt effektivt att beräkningsprogrammet hela tiden ska "polla" I/O-enheterna. Exempel är en beräkningsserver som inte skickar så mycket data över nätverket, men som måste svara snabbt på de förfrågningar som faktiskt kommer in till nätverkskortet.

b) DMA gör mest nytta för prestandan i ett system som utför både beräkningar samtidigt som stora mängder data ska in och ut ur systemet. (Ett typexempel på en enhet som nästan alltid använder DMA är ett grafikkort. Genom att DMA används kan processorn göra annat medans bilden läses ut ifrån minnet och ritas ut.)

Uppgift 9a

```
uppgift9a
        add.l   #1,a1             ; Det blir enklare att konstruera hitta_text
                                  ; om a1 pekar på en bokstav efter sista tecknet
loop
        cmp.l   a0,a1
        beq     ejhittad

        move.l  a0,a2
        jsr     hitta_tsea28
        add.l   #1,a0

        cmp.b   #1,d0
        bne     loop              ; Vi hittade inte TSEA28, fortsätt

        rts                       ; d0 kommer att vara 1 när vi kommer hit

ejhittad
        move.l #0,d0
        rts
```

Subrutin som används av 9a

Jag har här förberett subrutinen så att man i uppgift 9b kan hoppa direkt till `hitta_text`. För att göra det lite lättare för mig att skriva huvudsubrutinerna så använder jag a1 i subrutinerna också för att hålla koll på att jag inte hamnar utanför texten.

```
hitta_tsea28
        lea     tsea28_text,a3
        move.b  #6,d1             ; Antal tecken i tsea28
hitta_text
        cmp.l   a2,a1
        beq     returneranoll            ; Vi har gått utanför texten här!

        cmp.b   (a3)+,(a2)+
        bne     returneranoll
        add.b   #-1,d1
        bne     hitta_text

        move.l  #1,d0
        rts
returneranoll
        move.l  #0,d0
        rts
tsea28_text
        byte 84, 83, 69, 65, 50, 56  ; Här står det TSEA28 i ASCII
```

Uppgift 9b

Den här subrutinen fungerar nästan exakt som 9a. Först söker jag igenom texten efter ordet "tenta". Hittar jag det är saken klar. Om inte så ska jag utföra samma sak som i uppgift 9a och hoppar därför till denna subrutin istället

```
uppgift9b
       move.l   a0,a4         ; Kom ihåg början på texten i a4
       add.l    #1,a1

loop_uppgift9b
       cmp.l    a0,a1
       beq      hittade_ej_tenta

       lea      tenta_text,a3
       move.b   #5,d1
       move.l   a0,a2
       jsr      hitta_text
       add.l    #1,a0
       cmp.b    #0,d0
       beq      loop_uppgift9b
       bra      returneranoll ; Vi hittade ordet tenta!
hittade_ej_tenta
       move.l   a4,a0
       add.l    #-1,a1        ; Kompensera för att vi lade på ett på a1 tidigare
       bra      uppgift9a
tenta_text
       byte 116,101,110,116,97  ; Här står det tenta i ASCII
```

Alternativ lösning på Uppgift 9

Många studenter hade en annan typ av lösning på uppgift 9 än den som nämndes ovan. Jag inkluderar en variant på denna typ av lösning här eftersom jag tycker att det är ganska elegant att skriva en lösning där så lite som möjligt kan gå fel. Det blir lite mer att skriva på tentan om man löser den på följande sätt, men det är kanske inget fel med det...

```
alternativ_uppgift9
   ; Spara a0 till kolla_tsea28
    move.l a0,a2

loop_kolla_tenta

    ; Spara undan textpekaren
    ; Annars kan vi inte hantera
    ; fallet tententa exempelvis
    move.l a2,a3

    cmp.b  #116,(a3)+
    bne    ej_tenta
    cmp.l  a1,a3
    beq    ej_tenta

    cmp.b  #101,(a3)+
    bne    ej_tenta
    cmp.l  a1,a3
    beq    ej_tenta

    cmp.b  #110,(a3)+
    bne    ej_tenta
    cmp.l  a1,a3
    beq    ej_tenta

    cmp.b  #116,(a3)+
    bne    ej_tenta

    cmp.b  #97,(a3)
    bne    ej_tenta

    move.l #0,d0
    rts

ej_tenta
    cmp.l  a2,a1
    beq    kolla_tsea28

    add.l  #1,a2
    bra    loop_kolla_tenta
```

```
kolla_tsea28
    move.l a0,a3

    cmp.b  #84,(a3)+
    bne    ej_tsea28
    cmp.l  a1,a3
    beq    ej_tsea28

    cmp.b  #83,(a3)+
    bne    ej_tsea28
    cmp.l  a1,a3
    beq    ej_tsea28

    cmp.b  #69,(a3)+
    bne    ej_tsea28
    cmp.l  a1,a3
    beq    ej_tsea28

    cmp.b  #65,(a3)+
    bne    ej_tsea28
    cmp.l  a1,a3
    beq    ej_tsea28

    cmp.b  #50,(a3)+
    bne    ej_tsea28

    cmp.b  #56,(a3)
    bne    ej_tsea28

    move.l #1,d0
    rts

ej_tsea28
    cmp.l  a0,a1
    beq    ej_hittad
    add.l  #1,a0
    bra    kolla_tsea28

ej_hittad
    move.l #0,d0
    rts
```

Förklaring till better_fl1_microcode_without_branches

Mikrokoden börjar med att beräkna $y = (xAND - x)$. Efter denna beräkning kommer endast den minst signifikanta ettan att finnas kvar i talet. (Om x är noll så är även resultatet av denna

beräkning noll.) Slutligen så beräknas $y - 1$, vilket resulterar i ett tal där alla bitar till höger om den minst signifikanta ettan i y har blivit ett medans resterande bitar i talet är noll. Detta innebär att vi har efter denna beräkning har reducerat problemet till att enbart räkna alla ettor i talet, vilket vi i princip redan har löst i deluppgift 3a.