Tentamen
Datorteknik Y, TSEA28

Datum	2012-08-14
Lokal	TER2
Tid	8-12
Kurskod	TSEA28
Provkod	TEN1
Kursnamn	Datorteknik Y
Institution	ISY
Antal frågor	6
Antal sidor (inklusive denna sida)	7
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: Assemblerprogrammering(6p)

På adress $4000 fram till $40ff finns en array med 8-bitars värden (x_0 till x_{255} enligt tabell 1 nedan (vänstra delen). Din uppgift är att skriva en subrutin som räknar ut medelvärdet av två efterföljande värden i denna array (se högra delen av tabell 1). (Arrayen innehåller positiva heltal inom talområdet 0 till och med 255.)

Tabell 1: Innehåll på adress $4000 före (till vänster) respektive efter (till höger) det att subrutinen körts

$4000	x_0
$4001	x_1
$4002	x_2
$4003	x_3
$4004	x_4
$4005	x_5
$4006	x_6
$4007	x_7
$4008	x_8
$4009	x_9
$400a	x_{10}
...	...
$40ff	x_{255}

$4000	x_0
$4001	$(x_1 + x_0)/2$
$4002	$(x_2 + x_1)/2$
$4003	$(x_3 + x_2)/2$
$4004	$(x_4 + x_3)/2$
$4005	$(x_5 + x_4)/2$
$4006	$(x_6 + x_5)/2$
$4007	$(x_7 + x_6)/2$
$4008	$(x_8 + x_7)/2$
$4009	$(x_9 + x_8)/2$
$400a	$(x_{10} + x_9)/2$
...	...
$40ff	$(x_{255} + x_{254})/2$

Fråga 2: Mikroprogrammering(6p)

Du har fått i uppdrag att skriva mikrokod för instruktionen `REVERSE GRx` till den modelldator ni använt i labbarna (se figur 1). Denna instruktion ska ändra ordning på alla bitar i GRx så att den första biten och sista biten byter plats, den näst första biten och den näst sista biten byter plats, och så vidare.

Exempel:

```
; GR0 innehåller det binära värdet 1001001100101110
REVERSE GR0
; Efter att reverse-instruktionen körts kommer GR0 att
; innehålla det binära värdet 0111010011001001
```

Fråga 3: Diverse teorifrågor(15p)

Tips: Du bör kunna svara på dessa frågor med cirka två till fyra meningar per fråga. Du får också gärna rita figurer för att förtydliga dina förklaringar.

(a) (2p) Förklara kortfattat vad en MMU är samt nämn åtminstone en fördel med att ha en MMU i en dator.

(b) (2p) Förklara kortfattat vad DMA är och beskriv varför det kan vara en fördel om exempelvis ett nätverkskort eller grafikkort har stöd för DMA.

(c) (2p) Förklara hur subrutiner fungerar på MC68008 med fokus på hur datorn håller rätt på var processorn ska fortsätta körningen när subrutinen avslutas.

(d) (2p) Nämn två principiella skillnader mellan en subrutin och ett avbrott.

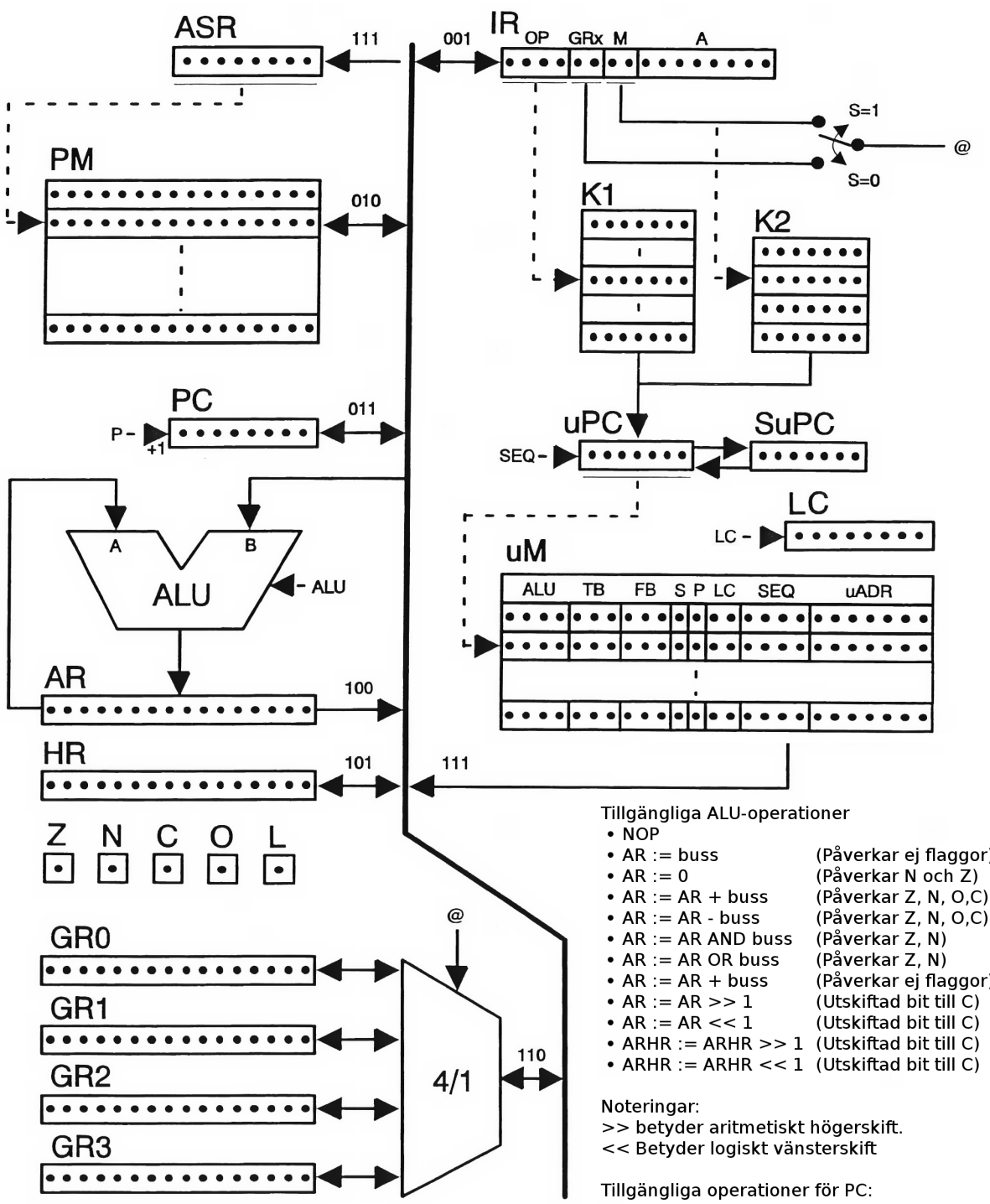
(e) (2p) Nämn minst två saker som särskiljer en DSP-processor jämfört med den MC68008 som ni använt i laborationerna.

(f) (2p) Förklara varför pipelining kan göra ett digitalt system (som exempelvis en dator) snabbare.

(g) (3p) Vad är skillnaden på SRAM och DRAM? Till vilken minnestyp är det viktigast att ha en cache och varför?

Fråga 4: A/D-omvandling(6p)

Beskriv kortfattat hur två olika typer av A/D-omvandlare fungerar samt jämför dessa två A/D-omvandlare med varandra och diskutera en viktig skillnad mellan dessa typer av omvandlare.



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.

Slutligen så används signalen S för att välja om M eller IR-fältet ska användas till att adressera GR0-GR3.

Figur 1: En välkänd mikroprogrammerad dator. (Björn Lindskog 1981)

Fråga 5: Förståelse av assemblerprogram(11p)

Du jobbar på ett företag som bygger en robot där styrmodulen bygger på ett MC68008-baserat Tutorkort. Till detta tutorkort har en avståndssensor kopplats in till PIA:n på följande sätt: Så snart det uppmätta avståndet ifrån sensorn **ändrar sig samt är större än 0** så genereras ett avbrott. Det uppmätta avståndsvärdet går då att avläsa ifrån PIA:n på adress $10080.

Tyvärr så märks det ganska snabbt att mjukvaran som hanterar denna avståndssensor ej fungerar helt problemfritt och det är upp till dig att reda ut denna situation.[1]

```
subrutin:
    move.l a0,-(a7)

    or.w   #$0700,sr

    move.w $3004,d0
    tst.b  d0
    beq    skip

    sub.w  #1,d0
    move.w d0,$3004

    move.l $3000,a0
    move.b -(a0),d0
    move.l a0,$3000

skip
    and.w  #$f8ff,sr
    rts
```

```
avbrott:
    move.l d0,-(a7)
    move.l a0,-(a7)
    move.w $3004,d0
    add.w  #$1,d0
    move.w d0,$3004

    move.l $3000,a0
    move.b $10080,(a0)+
    move.l a0,$3000
    move.l (a7)+,d0
    move.l (a7)+,a0
    rte
```

(a) (5p) Förklara noga hur subrutinen respektive avbrottsrutinen hanterar. Utöver en översiktlig beskrivning av vad subrutinen/avbrottsrutinen gör så vill jag se en diskussion av följande i ditt svar:

- Vad använder subrutinen/avbrottsrutinen innehållet på adress $3000 och $3004 till?
- Varför behövs instruktionerna `and.w` och `or.w`?
- Förklara hur returvärdet i subrutinen kan användas av huvudprogrammet.

(b) (6p) Det finns åtminstone tre buggar i detta system som kan få huvudprogrammet att krascha på olika mer eller mindre lättupptäckta sätt. Hitta åtminstone tre buggar och förklara följande för varje bug:

- Vad är det för bugg och hur kan du fixa till den?
- Hur påverkas resten av systemet av buggen?

Du kan anta att de subrutiner som ställer in avbrottsvektorer, PIA:n och så vidare är korrekta.

Fråga 6: Binär aritmetik(6p)

(a) (1p) Hur skriver du det decimala talet 25 binärt respektive hexadecimalt?

(b) (2p) Beskriv hur du utför operationen abs(x), där x är ett tvåkomplementsrepresenterat tal.x

(c) (3p) Beskriv hur N, C, samt Z-flaggan genereras i en typisk ALU (exempelvis den ALU som finns i modelldatorn i figur 1) i samband med en addition.

[1] Personen som skrev mjukvaran har dessutom, vis av tidigare erfarenhet, tagit en lång semester lagom tills dess att koden ska testas.

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 fråga 1

```
    move.l #$4000,a0
    move.l #$4100,a1
    clr.w  d0
    clr.w  d1
    move.b (a0)+,d0
loop
    move.b (a0),d1
    move.b d1,d2
    add.w  d0,d1
    lsr.w  #1,d1
    move.b d1,(a0)+
    move.b d2,d0
    cmp.l  a0,a1
    bne    loop
    rts
```

Det är värt att kommentera att det är viktigt att `d0` och `d1` adderas med `add.w` här eftersom det annars finns risk för overflow! Detta innebär också att det är nödvändigt att nollställa de 16 nedersta bitarna i d0 respektive d1 för att undvika att okända värden skiftas ner med `lsr.w` och sedan skrivs ut till minnet.

Lösningsförslag fråga 2

```
    ; Kommentar: För alla instruktioner gäller att uPC räknas upp med
    ; ett om inget annat anges.
    HR := PM(ASR)  ; Ladda operand
    LC := 16
    AR := 0

loop:
    uPC  := finished om L = 1
    AR   := AR << 1
    ARHR := ARHR >> 1
    AR   := AR << 1; uPC  := withcarry om C = 1, annars uPC++
    uPC := loop

withcarry:
    AR   := AR + Buss; Buss = 1
    uPC  := loop

finished:
    GRx  := AR; uPC := 0
```

Lösningsförslag fråga 3

Se kurslitteraturen för dessa frågor.

c) Kommentar: Tämligen många förklarade aldrig var återhoppsadressen kommer ifrån (dvs att det är adressen till instruktionen som följer efter subrutinshoppet).

Lösningsförslag fråga 4

Se kurslitteraturen här också. För full poäng så vill jag att det ska framgå tydligt hur A/D-omvandlarna fungerar. Detta görs enklast genom att rita scheman, men även förklaringar i löpande text har accepterats.

Lösningsförslag fråga 5

a)

Avbrottshanteraren pushar inkommande värde på en stack och subrutinen popar värden från samma stack. $3004 används för att hålla reda på hur många element som ligger på stacken och $3000 används som stackpekare. `and.w` och `or.w` behövs för att subrutinen inte ska avbrytas medans den läser från adress $3000 och $3004 (om vi exempelvis får ett avbrott precis innan den sista move.l-instruktionen så kommer $3000 och $3004 ej att vara i fas längre.

Vad gäller returvärdet så kommer de 8 lägsta bitarna i d0 att vara noll om stacken var tom. Annars innehåller dessa bitar det mätvärde som lästs ifrån stacken.

b)

Det finns några olika buggar här som påverkar systemet på mer eller mindre uppenbara sätt:

- Subrutinen återställer aldrig a0 med `move.l (a7)+,a0` innan rts vilket innebär att rts kommer att returnera till något vansinnigt ställe. Buggen påverkar resten av systemet så att applikationen (troligtvis) kraschar när subrutinen anropas. Fixas genom att lägga till denna instruktion.
- Avbrottsrutinen sparar och återställer alla register som används. Tyvärr så återställs d0 och a0 i fel ordning. Detta är en mer lömsk bugg som kommer att leda till att godtyckliga delar av programmet kommer att få a0 och d0 förväxlade på ett icke-deterministiskt sätt. Fixas genom att byta ordning på `move.l` instruktionerna innan rte.
- Avbrottsrutinen har ej någon koll på att stacken inte växt för långt. Om subrutinen inte anropas tillräckligt ofta kommer stacken att växa tills någon annan del av primärminnet skrivs över och troligtvis göra att applikationen kraschar på obskyra sätt. Fixas genom exempelvis följande kod efter `add.w` i avbrottet:
  ```
  cmp.w #100,d0
  beq failure ; Labeln failure finns vid första move.l innan rte
  ```
- Det är mycket underligt att `tst.b` används i subrutinen när alla andra operationer på värdet i $3004 använder `.w` istället för `.b`.
- Som en tentand påpekade så är det slutligen också mycket tveksamt om en stack är rätt sätt att lagra denna typ av data på, troligtvis skulle en kö (FIFO) vara bättre, förslagsvis implementerat som en cirkulär buffer.

Lösningsförslag uppgift 6

a) $25_{10} = 16_{10} + 8_{10} + 1_{10} = 19_{16} = 11001_2$ **b)** Om MSB-biten i x är noll: $\text{abs}(x) = x$. Om MSB-biten är ett: $\text{abs}(x) = \text{invert}(x) + 1$ (där invert() inverterar alla bitarna i argumentet) **c)** N-flaggan sätts om MSB-biten i resultatet är en etta. C-flaggan sätts om carry-ut ifrån adderaren är en etta. Z-flaggan sätts om resultatet är noll.