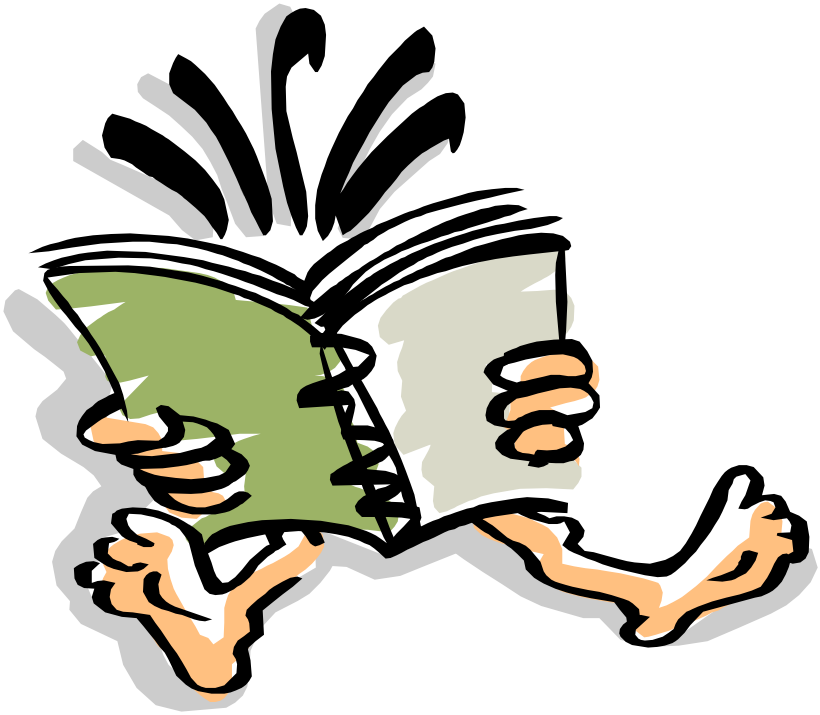


Tentamen
Mikrodatorteknik, 4p/5p

2001-03-21

Skrivtid:	5 timmar
Preliminära gränser 5p:	G: 52-79, VG: 80- (Max: 100p)
Preliminära gränser 4p:	G: 48-79, VG: 80- (Max: 100p)
Ansvariglärare:	Mattias O'Nils
Telefon:	070-695 7668
Hjälpmedel:	The 68000's Instruction Set Miniräknare
Program:	Systemteknikprogrammet, ST3

OBS!
Skriv tydligt!
Skriv namn och kod på varje blad.
Skriv bara på en sida av varje blad.
Redovisa alla steg i beräkningar och konstruktioner.
Rita snygga figurer!

Lycka till,
/Mattias

1. Vilka register finns det i processorn och vad används dessa till? **6p**

2. Nämn och beskriv kortfattat två sätt att adressera data i ett cacheminne. **8p**

3. Vilken uppgift har UDS* och LDS* ? **5p**

4. Vad står DMA för samt beskriv hur DMA fungerar? **6p**

5. Antag att register D1 innehåller $00000099. Ange D1, C-flaggans samt Z- flaggans värde efter instruktionerna nedan har exekverats. Observera att a till f är enskilda instruktioner och inte ett program. **6p**

a)	EORI.B	#$66,D1
b)	ADDI.B	#$66,D1
c)	ADDI.B	#$67,D1
d)	SUBI.B	#$66,D1
e)	ASR.B	#2,D1
f)	ROL.B	#4,D1

6. Funktionsanrop i assembler. En funktion i C är deklarerade som följande:

```
void printChar(int a, int b);
```

Parametrarna a och b ska skickas till funktionen via stacken.

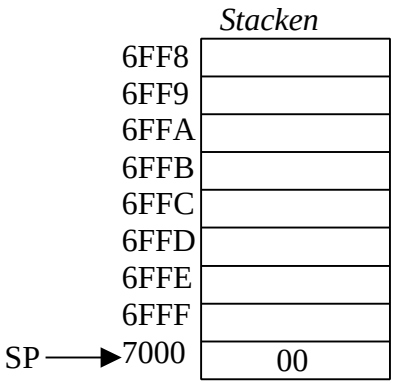
a) Rita stackens innehåll när subrutinen printChar börjar exekveras om stackpekaren pekar på adress $7000 innan anropet. Anropet till funktionen i assembler görs enligt följande:

Funktionsanrop i C

```
PrintChar(a,b);
```

Funktionsanrop i assembler

```
MOVE.W a,-(A7)
MOVE.W b,-(A7)
BSR printChar
ADDA.L #4,A7
```



Rita inte på detta paper!!!

6p

b) Vilken funktion har instruktionen ADDA.L #4,A7 ?

3p

7. Adresseringsmetoder: Vad kommer D2 och A3 att innehålla efter att instruktionen har körts. Innehållet i minnet är specificerat av tabellen på nästa sida. Instruktionerna är oberoende av varandra och D2 är $00000000 samt A3 = $00004004 före varje instruktion exekveras.
    a) MOVE.L $4004, D2
    b) MOVE.W 4(A3), D2
    c) MOVE.L -(A3), D2
    d) MOVE.B -(A3), D2
    e) MOVE.W -2(A3), D2
    f) MOVE.W #$4004, D2
    g) MOVE.B (A3)+,D2
    h) MOVE.L (A3)+,D2

8p

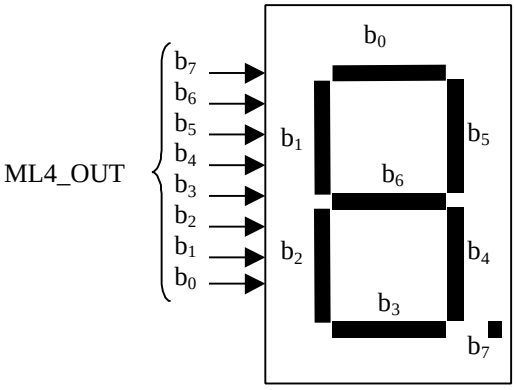
Adress (Hexadecimal)	Innehåll (Hexadecimal)
4000	77
4001	21
4002	C3
4003	FD
4004	00
4005	14
4006	99
4007	74
4008	A6
4009	AA

8. Assemblerprogrammering: Skriv en subrutin i 68000-assembler som skriver ut ett tal (0-15) till en display. På displayen presenteras talet som en hexadecimal siffra (0-F). Displayen är av samma typ som på labbkorten, dvs. en 7-segment display.

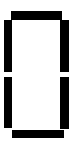
    Displayen är ansluten till ML4_OUT (porten är definierad). Varje bit i ML4_OUT är ansluten till en lysdiod i 7-segment displayen. Kopplingen mellan bitarna i ML4_OUT och lysdioderna är definierad i figuren nedan (T ex. b_0 är kopplad till den översta vågräta dioden). b_7 är kopplad till punkten på 7-segmentdiesplayen, så den sätter ni alltid till noll.

    In: ML4_OUT = Talet som ska skrivas ut till 7-segmentdisplayen (tal mellan 0-16)
    Ut: Inget ska returneras från subrutinen.

    Exempel: Nedre byten på D0 = %00110000 ska ge en etta på displayen och ML4_OUT = %00111111 ger en nolla.



Ex. för en nolla
ML4_OUT = %00111111



b_7=0
b_6=0
b_5=1
b_4=1
b_3=1
b_2=1
b_1=1
b_0=1

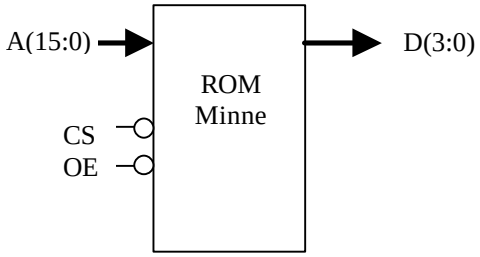
12p

9. Programmet nedan består av 16 stycken instruktioner, din uppgift är att tala om vilka instruktioner som kommer att exekveras och varför. (Det är inget fel på programmet, ni ska bara fundera ut vilka instruktioner som exekveras och varför)

```
            ORG         $11000                  * Instruktions nummer:
            MOVE.B      #$F0,D0                 * 1
            LSL.B       #3,D0                   * 2
            BPL         Label_1                 * 3
            ORI.B       #1,D0                   * 4
Label_1     EORI.B      #$81,D0                 * 5
            BEQ         Label_2                 * 6
            EORI.B      #1,D0                   * 7
            BEQ         Label_3                 * 8
Label_2     SUB.B       #1,D0                   * 9
            ANDI.B      #$80,D0                 * 10
            BPL         Label_4                 * 11
Label_3     ANDI.B      #$7F,D0                 * 12
            BEQ         Label_4                 * 13
            ANDI.B      #$7F,D0                 * 14
            BEQ         Label_4                 * 15
Label_4     BRA         Label_4                 * 16
```

8 p

10. Sättsamman ett ROM-minne på 512 kbyte (16-bitar bred), med hjälp av minneskomponenter 64kx4bit (se komponenten). Ni för gärna använda en färdig avkodare för att generera CS-signalerna.
Obs! Generera bara CS signalen. **10p**



11. Konstruera en partiell minnesavkodare med följand minneskarta. Använd en eller flera 4-16 avkodaren 74HC154 beskriven på nästa sida. (Dvs. generera CS signaler för 68000)
10p

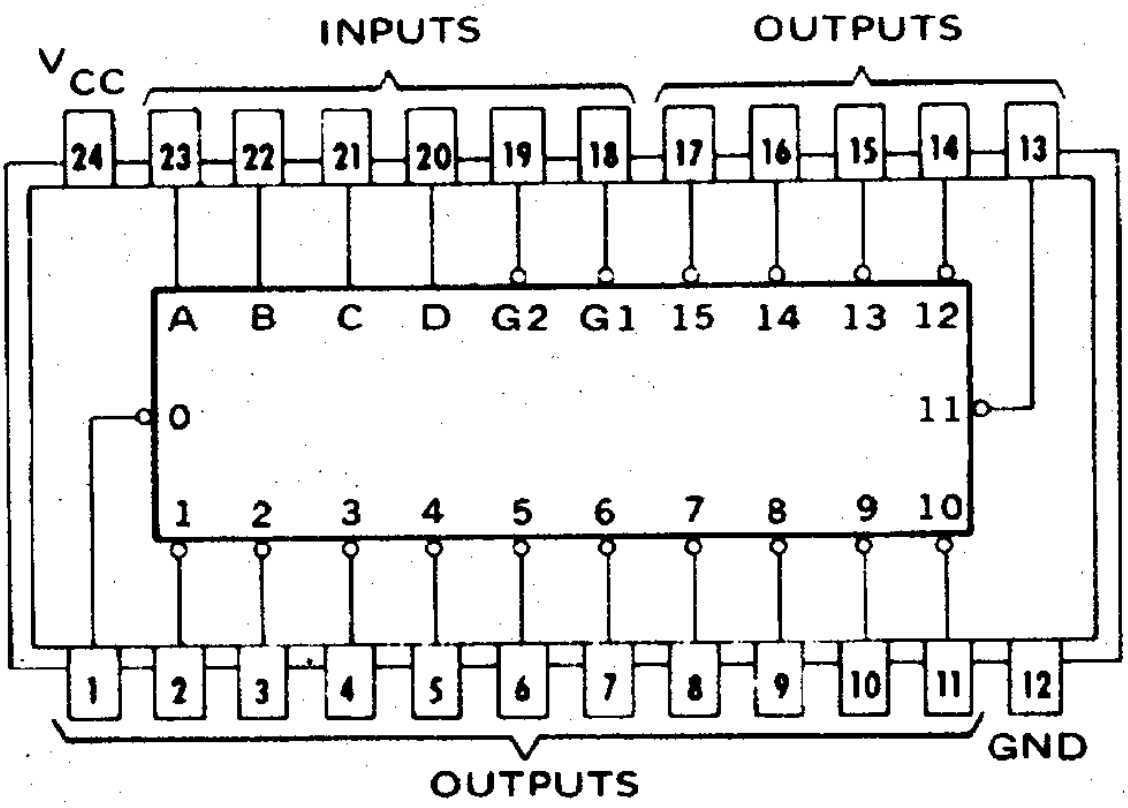
ROM1	00 0000 -	03 FFFF
ROM2	04 0000 -	04 FFFF
RAM1	05 0000 -	05 FFFF
RAM2	06 0000 -	07 FFFF
IO_0	08 0000 -	08 0FFF
IO_1	08 1000 -	08 1FFF
IO_2	08 2000 -	08 2FFF
IO_3	08 3000 -	08 3FFF
IO_4	09 4000 -	08 4FFF

12. Skriv en subrutin som konverterar ett 32-bitars binärttal till en textsträng med nollor och ettor.

In: D0 = Talet som ska omvandlas. A0 pekar på den adress där strängen ska ligga.
Ut: En teckensträng med ettor och nollor och avslutats med en binär nolla på 33 positionen för att indikera strängslut.

Exempel: Talet $4323 Ska ge textsträngen "00000000000000000100001100100011",0.

Ledning: Att omvandla från en siffra till ASCII tecknet för siffran görs genom att addera ASCII-koden för en nolla ('0' = $30). **12p**


INPUTS
OUTPUTS
VCC
24
23
22
21
20
19
18
17
16
15
14
13
A
B
C
D
G2
G1
15
14
13
12
0
11
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
11
12
OUTPUTS
GND