Lösningsförslag tenta 2010-12-13

1. X = 01011010; Y = 10010101 (8 bitars ordlängd)

a) $R = X+Y_{1k}+1$

```
876543210      bitnummer
011110101   1  carry
 01011010   X
+01101010   Y1k
 11000101 = R
```
(1p)

b) $N = r_7 = 1$;
$Z = 0$ ($R \neq 0$);
$V = x_7*y_7*r_7' + x_7'*y_7'*r_7 = 0*0*1' + 0'*0'*1 = 1$; (Vid ”subtraktion” är y_7 motsvarande bit i Y_{1k}.)
$C = c_8' = 0' = 1$ **(1p)**

c) $\underline{R} = 11000101_2 = C5_{16} = \underline{197}$;
$\underline{X} = 01011010_2 = 5A_{16} = \underline{90}$;
$\underline{Y} = 10010101_2 = 95_{16} = \underline{149}$;
Resultatet R är felaktigt (C = 1). Korrekt resultat om C = 0. **(1p)**

d) ($r_7 = 1$, neg) $R_{2k} = 2^8 - 197 = 59$ $\underline{R}$ motsvarar $\underline{-59}$
($x_7 = 0$, pos) $\underline{X} = \underline{90}$
($y_7 = 1$, neg) $Y_{2k} = 2^8 - 149 = 256 - 149 = 107$ $\underline{Y}$ motsvarar $\underline{-107}$
Resultatet R är felaktigt (V = 1). Korrekt resultat om V = 0. **(1p)**

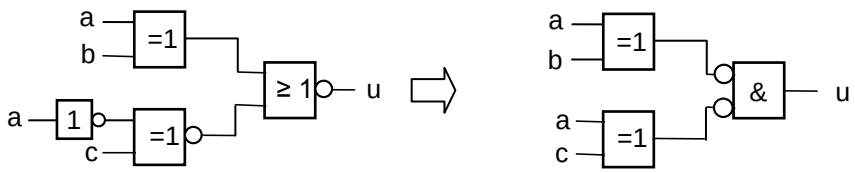
e) Resultatet av X – Y skall vara negativt eller noll. $f = (N \oplus V) + Z$ **(2p)**

f) $N_{flyt} = C3DA7000_{16} =$ 1/100 0011 1/101 1010 0111 0000 0000 0000
s c f

s = 1 (-)
c = 135; exp = 135 – 127 = 8;
m = 1.f = 1.10110100111000000000000

$\underline{N_2} = -1.10110100111000000000000\underline{*2^8} = -(256 + 128 + 32 + 16 + 4 + .5 + .25 + .125) = \underline{-436{,}875}$
(2p)

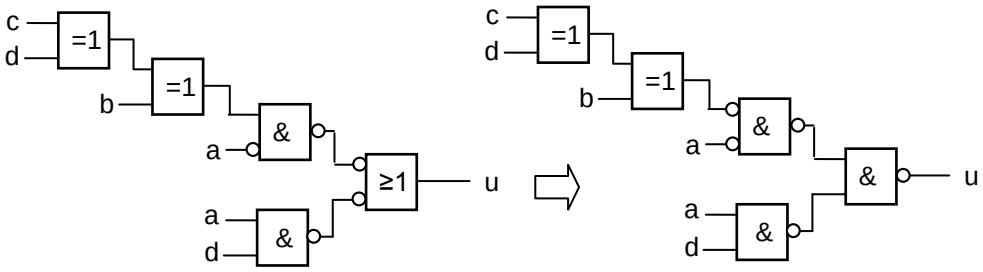
g)



$\underline{u} = (a \oplus b)'(a \oplus c)' = (a'b' + ab)(a'c' + ac) = a'b'a'c' + a'b'ac + aba'c' + abac = \underline{a'b'c' + abc}$
(3p)

2.

ab \ cd	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	0	1	1	0
10	0	1	1	0



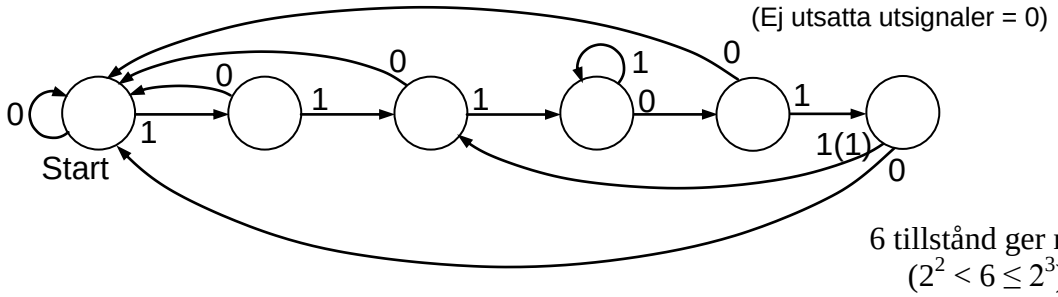
(Ring på ingångar i figuren ovan betyder att signalen inverteras innan den når grinden.)

$\underline{u} = ad + a'b'c'd' + a'b'cd + a'bc'd + a'bcd' = ad + a'(b'c'd' + b'cd + bc'd + bcd') =$
$= ad + a'[b'(c'd' + cd) + b(c'd + cd')] = ad + a'[b'(c \oplus d)' + b(c \oplus d)] = \underline{ad + a'[b \oplus (c \oplus d)]'}$

(4p)

3.

a)



6 tillstånd ger minst 3 vippor
$(2^2 < 6 \leq 2^3)$

(4p)

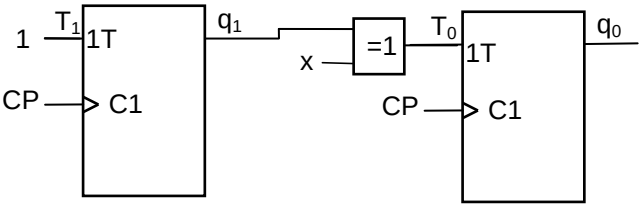
b)

x	q_1	q_0	q_1^+	q_0^+	T_1	T_0
0	0	0	1	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	1	1
0	1	1	0	0	1	1
1	0	0	1	1	1	1
1	0	1	1	0	1	1
1	1	0	0	0	1	0
1	1	1	0	1	1	0

$T_1 = 1$
$T_0 = xq_1' + x'q_1 = x \oplus q_1$

T_1	q_1q_0 00	0	11	10
x 0	1	1	1	1
1	1	1	1	1

T_0	q_1q_0 00	01	11	10
x 0	0	0	1	1
1	1	1	0	0



(5p)

4. $9 \cdot A - (B + 1) = 8 \cdot A + (A - B - 1)$

CP	RTN	Styrsignaler (=1)
1	B → T	OE_B, LD_T
2	A – T – 1 → R	OE_A, f_3, f_2, LD_R
3	R → T	OE_R, LD_T
4	2A → R	OE_A, f_3, f_1, f_0, LD_R
5	2R → R	OE_R, f_3, f_1, f_0, LD_R
6	2R → R	OE_R, f_3, f_1, f_0, LD_R
7	R + T → R	OE_R, f_3, f_1, LD_R
8	R → A	OE_R, LD_A

(5p)

5. a)

State	S-term	RTN-beskrivning	Aktiva styrsignaler (=1)
Q_5	$Q_5 \cdot I_{xx}$	X→T, SP-1→SP	OE_X, LD_T, DecSP
Q_6	$Q_6 \cdot I_{xx}$	SP→MA	OE_{SP}, LD_{MA}
Q_7	$Q_7 \cdot I_{xx}$	PC→M	OE_{PC}, MW
Q_8	$Q_8 \cdot I_{xx}$	A+T→R	OE_A, f_3, f_1, LD_R
Q_9	$Q_9 \cdot I_{xx}$	R→PC, Next Fetch	OE_R, LD_{PC}, NF

Instruktionen består endast av OP-koden. Adressen efter OP-koden skrivs på stacken. Innehållen i A- och X-registret adderas och summan placeras i PC, dvs hopp görs till adressen X+A. Detta är JSR A,X.

(2p)

b)

State	S-term	RTN-beskrivning	Aktiva styrsignaler (=1)
Q_5	$Q_5 \cdot I_{F3}$	PC → MA, PC+1→PC	OE_{PC}, LD_{MA}, IncPC
Q_6	$Q_6 \cdot I_{F3}$	M → T	MR, LD_T
Q_7	$Q_7 \cdot I_{F3}$	PC→MA, PC+1→PC	OE_{PC}, LD_{MA}, IncPC
Q_8	$Q_8 \cdot I_{F3}$	M → MA	MR, LD_{MA}
Q_9	$Q_9 \cdot I_{F3}$	M *OR* T → R, Flags→CC	MR, f_2, f_0, LD_R, LD_{CC}
Q_{10}	$Q_{10} \cdot I_{F3}$	R → M, Next Fetch	OE_R, MW, NF

(5p)

6.

a) JMP gör ett hopp till en given (absolut) adress, där adressen finns som andra ord i instruktionen. BRA gör också ett hopp men i detta fall adderas instruktionens andra ord till det aktuella PC-värdet. Det innebär att det andra ordet innehåller avståndet till destinationen för hoppet. Om programmet flyttas i minnet kommer adresserna inom programmet att ändras, men eftersom avstånden inom programmet inte påverkas så kommer BRA att fungera, medan JMP hoppar till den gamla inaktuella adressen. JMP är lättare att hantera när man vet hoppadressen och den är snabbare än BRA. **(2p)**

b) Funktionen hos signalen NF är att ladda tillståndsräknaren med värdet 3 som motsvarar första tillståndet i FETCH. Om sista tillståndet i EXECUTE inte aktiverar NF så fortsätter tillståndsräknaren räkna tills den varvar och börjar om med RESET följt av FETCH, dvs processorn startar om på det aktuella programmets startadress. **(2p)**

c) Under RESET-fasen bildar ALU:n adressen FF_{16} som laddas i R-registret och sedan flyttas till MA-registret. Därefter läser processorn innehållet på adressen FF_{16} som skall vara startadressen till det program man vill starta, lägger den i PC och övergår till FETCH-fasen. **(2p)**

d)

Adr	Data	~	Läge			
20	11 32	4		LDX	#TAB	
22	10 0C	4		LDAB	#12	
24	81 0B	7		LDAA	11,X	
26	88	6	LOOP1	LDX	B,X	
27	E2	4	LOOP2	DEX		
28	00	3		NOP		
29	5C FC	5		BPL	LOOP2	27 – 2B = FC
2B	11 32	4		LDX	#TAB	
2D	41	4		INCA		
2E	5B F6	5		BMI	LOOP1	26 – 30 = F6
30	5A 0F	5		BRA	NEXT	41 – 32 = 0F
32	-- -- -- -- -- -- -- --	-	TAB	RMB	8	
3A	01 FD 05 FB 09 F6 14	-		FCB	1,-3,5,-5,9,-10,20	
41	00	3	NEXT	NOP		

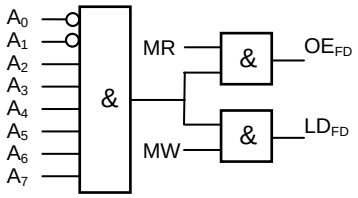
(3p)

e) $\underline{t}$ = [4+4+7+(6+(4+3+5)*10+4+4+5)*5+5+3] μs= [23+(19+12*10)*5] μs = $\underline{718\mu s}$ **(3p)**

7.

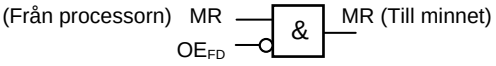
a) Inportens ”three-state driver” skall aktiveras (OE_{FC}) vid läsning på adress FC_{16} och utportens register skall laddas (LD_{FC}) med data från databussen vid skrivning på adress FC_{16}. = 11111100_2

(Ring på ingången till OCH-grinden nedan betyder att signalen inverteras innan den når grinden. Signalerna A_0-A_7 är adressbitarna från adressbussen.)



(2p)

b) Vid läsning från inporten kommer indata att kollidera med data från minnets adress FC_{16}. Man måste därför förhindra att minnet lägger ut sin data på databussen vid läsning på inportsadressen FC_{16}. Nedan visas hur detta kan fixas.



(2p)

c) Man använder en variabel i minnet och skriver alltid samma data i denna som på utporten. När man vill veta senaste utmatade värde kan man läsa det i minnesvariabeln. **(2p)**

8.

```
CRLFCNT PSHX              Spara register på stack

        CLRA              Nollställ CR-räknare
        CLR   LFCNT       Nollställ LF-räknare i minnet

CRLOOP  LDAB  1,X+        Hämta data från textsträng. Öka pekare
        TSTB              Strängslut?
        BEQ   CREX        Ja, avsluta

        ANDB  #$7F        Maska paritetsbit (bit 7)

        CMPB  #$0D        CR?
        BNE   LFTST       Nej, testa LF
        INCA              Öka CR-räknare
        BRA   CRLOOP

LFTST   CMPB  #$0A        LF?
        BNE   CRLOOP      Hämta nästa ASCII-tecken
        INC   LFCNT       Öka CR-räknare
        BRA   CRLOOP

CREX    LDAB  LFCNT       Hämta LF-räknare
        PULX              Återställ register
        RTS

LFCNT   RMB   1           LF-räknare
```

(6p)