CHALMERS

Institutionen för data- och informationsteknik
Avdelningen för datorteknik

TENTAMEN (med svar och vissa lösningar)

KURSNAMN	**Maskinorienterad programmering**
PROGRAM:	**Dataingenjör och elektroingenjör åk 1/ lp 3** **Mekatronikingenjör åk 2/ lp 3**
KURSBETECKNING	**LEU500**
EXAMINATOR	**Lars-Eric Arebrink**
TID FÖR TENTAMEN	**Måndag 2010-03-08 kl 14.00 – 18.00**
HJÄLPMEDEL	**Av institutionen utgiven** **”Instruktionslista för CPU12” (INS2)** **Tabellverk eller miniräknare får ej användas.**
ANSV LÄRARE: **besöker tentamen**	**Lars-Eric Arebrink** **772 5718** **Vid flera tillfällen**
DATUM FÖR ANSLAG av resultat samt av tid och plats för granskning	**Resultatlistor med de anonyma koderna anslås senast 2010-03-25 på kursens hemsida.** **Granskning på institutionen 2010-03-25 och 2010-03-26 kl 12.30-13.00.**
ÖVRIG INFORM. **BETYGSGRÄNSER.** **SLUTBETYG**	**Tentamen omfattar totalt 60 poäng.** **24p ≤ betyg 3 < 36 p ≤ betyg 4 < 48 p ≤ betyg 5** **För godkänt slutbetyg 3, 4 eller 5 på kursen fordras betyg 3, 4 eller 5 på tentamen samt godkända laborationer.**

1. Besvara kortfattat följande frågor, som alla utom h) - k) avser CPU12.

a) Översätt assemblerinstruktionen CLC till maskinspråk och visa hur maskinkoden placeras i minnet. **Svar: 10 FE (Hex) = ANDCC #$FE** **(1p)**

b) Vilken assemblerinstruktion har den hexadecimala maskinkoden 6B E2 16 24?
Svar: STAB $1624,X **(2p)**

c) Assemblerinstruktionen TFR B,X kan skrivas som en alternativ assemblerinstruktion. Vilken?
Svar: SEX B,X **(1p)**

d) Översätt assemblerinstruktionen CPY -$FF,SP till maskinspråk. Visa hur maskinkoden placeras i minnet. **Svar: AD F1 01** **(2p)**

e) Assemblerinstruktionen BRN $1000 har operationskoden på adressen 1057H. Visa hur maskinkoden placeras i minnet. På vilken adress utförs nästa instruktion?
Svar: 21 A7, Nästa instruktion finns på adressen 1059H. **(2p)**

För vilka värden W ($0 \le W \le 255$) på minnesordet på adressen Wadr utförs hoppet i f) och g)?

f)
```
LDAB  #$C0
CMPB  Wadr
BGE   Hopp
```
Svar: $128 \le W \le 192$ **(2p)**

g)
```
LDAB  #$65
CMPB  Wadr
BPL   Hopp
```
(Tänk på att overflow kan inträffa vid jämförelsen!)

Svar: $0 \le W \le 101$ eller $230 \le W \le 255$ **(3p)**

h) Det finns en principiell skillnad mellan asynkron och synkron seriekommunikation. Vilken är denna skillnad?
Svar: Vid asynkron seriekommunikation har sändare och mottagare olika klocksignal. Vid synkron seriekommunikation har de i princip samma klocka (vilket oftast åstadkoms genom att klocksignalen inkluderas i datasignalen). **(1p)**

i) Fem CAN-noder, som är kopplade till en bussledning så att nolla är dominant, börjar samtidigt sända var sitt meddelande. Identifierarna i de olika meddelandena är 539H, 0, 35H, 7FFH och 85H. Ange identifierarna för meddelandena i den tidsordning de sänds.
Svar: 0, 35H, 85H, 539H och 7FFH (Lägre identiferarvärde ger högre prioritet.) **(2p)**

j) Skriv det decimala talet 653,375 som ett 32-bitars flyttal enligt IEEE-standard 754-1985 (23 bitar av mantissan och 8 bitars karakteristika). Ge svaret på hexadecimal form.
Svar: $N = 653{,}375 = (1010001101.011)_2 = 1.010001101011\ 2^9$ **(2p)**
$s = 0$ (+), $c = 9 + 127 = 136 = 128 + 8 = (10001000)_2$
Nflyt = s/c/f = 0/10001000/01000110101100000000000 = 44235800H

k) Visa approximativt hur många bitar man skulle behöva i mantissan på ett flyttal för att man skall kunna ”lita på” 12 decimala siffror om talet översätts till decimal form.
Svar: 12 decimala siffror ger 10^{12} olika talkombinationer.
Eftersom $10^3 \approx 2^{10}$ är $10^{12} = (10^3)^4 \approx (2^{10})^4 = 2^{40}$ vilket motsvarar 40 bitar. **(2p)**

2.

a) Skriv en subrutin ADDNIB i assemblerspråk för CPU12, som adderar den vänstra halvan av 8-bitars dataord i en nollterminerad sträng i minnet med den högra halvan av samma dataord och placerar summan i en ny nollterminerad sträng. Se figuren nedan som beskriver additionen. Vid anrop av subrutinen skall adressen till indatasträngen finnas i X-registret och adressen till utdatasträngen i Y-registret. Endast flaggregistret får vara förändrat vid återhopp. För full poäng skall programmet vara ”korrekt” radkommenterat.

$$\begin{array}{r} b_7\ b_6\ b_5\ b_4 \\ +b_3\ b_2\ b_1\ b_0 \\ \hline s_4\ s_3\ s_2\ s_1\ s_0 \end{array}$$

```
*
*   Subrutin ADDNIB
*
ADDNIB  PSHD
        PSHX
        PSHY
*
ADLOOP  LDAA    1,X+      Hämta databyte från inträng
        TFR     A,B       Kopiera databyte
        ANDA    #$0F      Maska fram bit 3-0
        LSRB              Flytta bit 7-4 i kopian till bit 3-0
        LSRB
        LSRB
        LSRB
        ABA               Utför nibbleaddition
        STAA    1,Y+      Placera summan i utsträng
        BNE     ADLOOP    Nästa byte om summan ej 0.
*                         (Automatisk nollterminering)
ADDEX   PULY
        PULX
        PULD
        RTS
```

(5p)

b) Hur många ”E-klockperioder” använder CPU12 (HCS12) för att köra programsekvensen nedan?

```
          ORG     $1000
          LDAA    #$F6    1
ALOOP     LDX     #$FF9C  2       *10
XLOOP     IBNE    X,XLOOP 3*100   *10
          LBRN    ALOOP   3       *10
          IBNE    A,ALOOP 3       *10

N = 1+(2+3*100+3+3)*10 = 3081
```

F6H motsvarar -10_{10}
FF9C motsvarar -100_{10}

(3p)

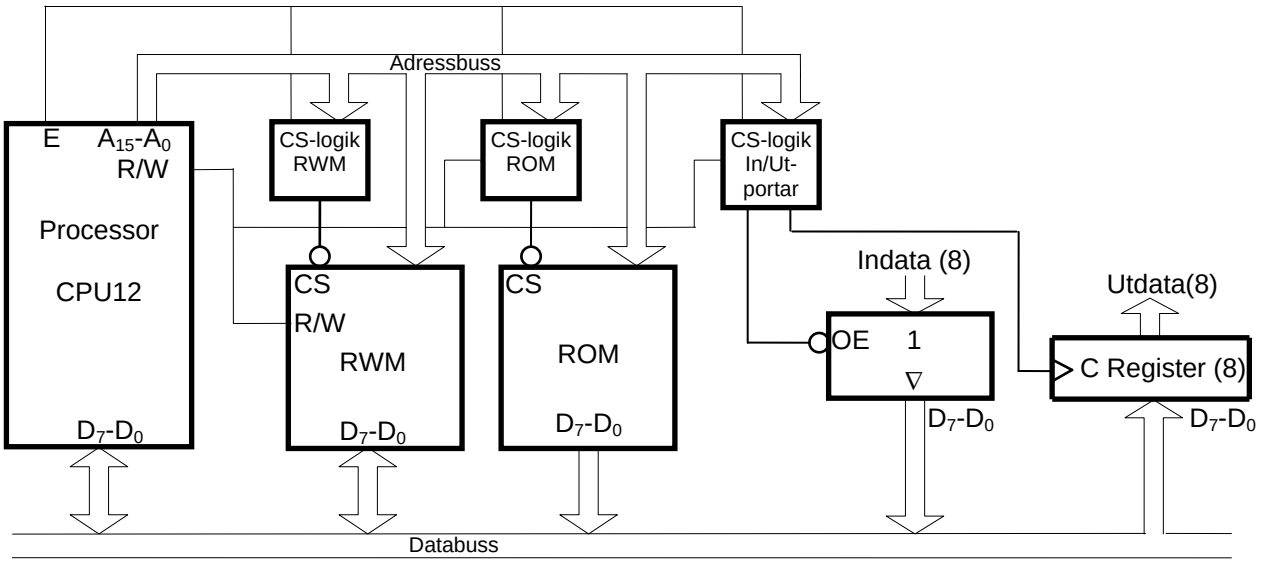
c) Programsekvensen nedan visar ett sätt att överföra två inparametrar 87H och 1234H till en subrutin. Visa med en programsekvens hur subrutinen kan hämta inparametrarna till register A och X och senare göra ett återhopp till adressen efter andra inparametern nedan. Inga registerinnehåll behöver sparas på stacken. Tänk på att subrutinen kan anropas från flera ställen i samma huvudprogram.

```
        .
        .
JSR     SUBRUT
FCB     $87         Inparameter 1
FDB     $1234       Inparameter 2
        .
        .
```

```
SUBRUT  LDY     ,SP     Stackinnehåll till Y-reg
        LDAA    ,Y      Inparameter 1till A-reg
        LDX     1,Y     Inparameter 2 till X-reg
        LEAY    3,Y     Justera Y till återhoppsadress
        STY     ,SP     Uppdatera SP med återhoppsadress
        .
        .
        RTS
```

(4p)

3. Ett datorsystem visas nedan:

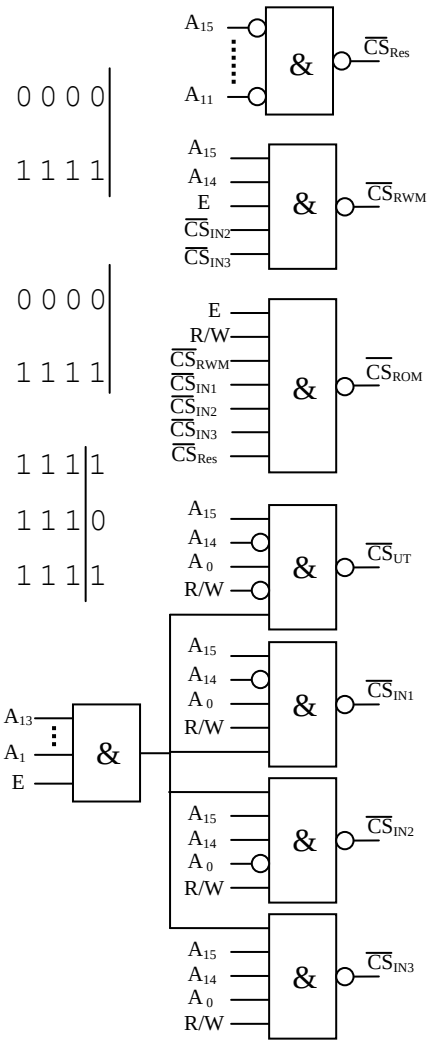


Figuren ovan visar principen för anslutning av externa minnesmoduler och externa in-/utportar till processorn CPU12. En 16 kbyte RWM-modul skall vara placerad med start på adress C000H. På adresserna för RESET-vektorn skall två inportar vara placerade och på adress BFFFH skall en inport och en utport placeras. Resten av adressrummet skall fyllas ut med en 64 kbyte ROM-modul, men 2048 adresser med start på adress 0 skall inte aktivera någon minnesmodul eller port vid läsning eller skrivning. Det innebär att inportarna skall prioriteras före minnesmodulerna och att RWM-modulen skall prioriteras före ROM-modulen.

Rita CS-logiken för minnesmodulerna och portarna. Ange adressintervallen för de två minnes-modulerna. Använd fullständig adressavkodning. Endast grundläggande logikgrindar med valfritt antal ingångar får användas. **(8p)**

Svar:

Reserverat:	Startadress	0000H	=	0000	0	000	0000	0000								
2048 = 2k = 2^{11}																
	Slutadress	07FFH	=	0000	0	111	1111	1111								
RWM 16k:	Startadress	C000H	=	11	00	0000	0000	0000								
16k = 2^{14}																
	Slutadress	FFFFH	=	11	11	1111	1111	1111								
IN1/UT:	Adress	BFFFH	=	10	11	1111	1111	111	1							
IN2:	Adress	FFFEH	=	11	11	1111	1111	111	0							
IN3:	Adress	FFFFH	=	11	11	1111	1111	111	1							



Resulterande adressintervall:
RWM: C000H – FFFDH
ROM: 800H – BFFEH

4. Avbrottssystemet i CPU12 skall användas till att köra två olika program, ProgA och ProgB, pseudoparallellt. Programmen som utgörs av evighetsslingor har startadresserna 1000H resp. 2000H. De använder var sin stack med BOS på adresserna 2000H resp. 3000H. En händelsedetektor som genererar binärpulser skall också ge avbrott. Vid varje sådant avbrott skall en 8-bitars variabel EVENT ökas med ett.

Man har tillgång till en digital binär signal med frekvensen 100 Hz. En oanvänd inport som får användas finns på adressen INPORT. CS-signalen för inporten är inte tillgänglig.

a) Rita ett kopplingsschema som visar hur man kan generera avbrott med frekvensen 100 Hz och då pulser kommer från händelsedetektorn. Det får förutsättas att adressområdet 8000H – 8FFFH är tomt. Det finns inga övriga avbrottskällor att ta hänsyn till i systemet! Processorns IRQ'-ingång skall användas. D-vippor, NAND- och NOT-grindar får användas. **(2p)**

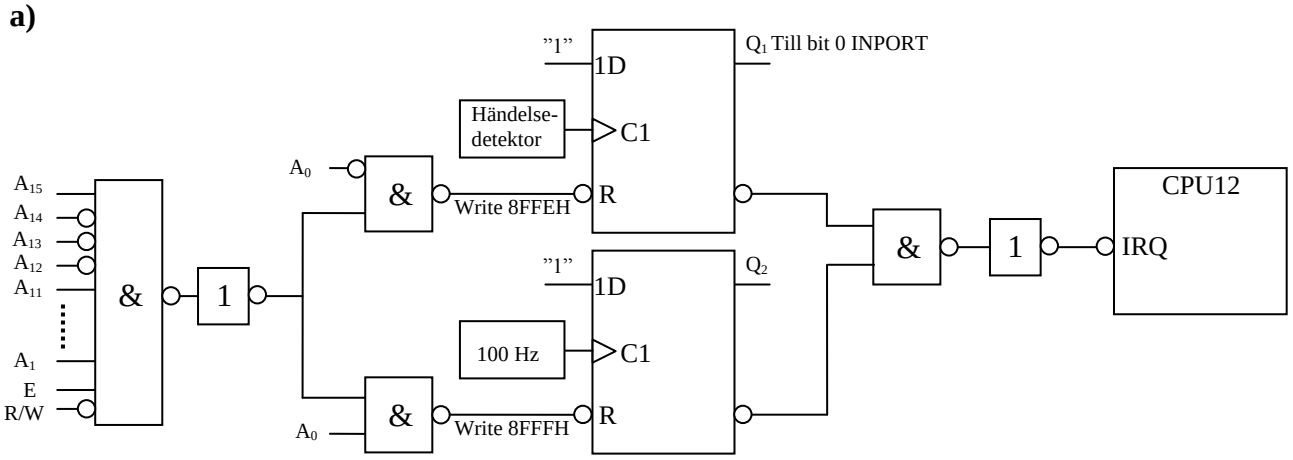
b) ProgA inleds med en del som bara körs vid starten och fortsätter sedan med en evighetsslinga. I den första delen av ProgA görs nödvändiga initieringar av avbrottssystemet för de två avbrottskällorna och för processbytet vid varje 100Hz-avbrott. Det förutsätts att systemet startar med reset av processorn och att ProgA då startas. Skriv programmet ProgA. Utrymme för globala variabler finns på adresserna 1E00H- 1E0FH. **(4p)**

c) Skriv avbrottsrutinen som uppdaterar variabeln EVENT och utför processbyte. **(3p)**

Alla odefinierade symboliska adresser ovan är definierade på annat ställe i programmet. Assemblerspråk för processorn CPU12 skall användas. Radkommentarer skall finnas!

Uppgift 4 svar:

a)




```
BOSA    EQU     $2000
BOSB    EQU     $3000
ProgA   EQU     $1000
ProgB   EQU     $2000
IRQVEC  EQU     $FFF2
IRQFF   EQU     $8FFE
*
        ORG     $1E00   Globala variabler
EVENT   RMB     1       Händelseräknare
SPSAVE  RMB     2       Passiv stackpekare
```

b)

```
        ORG     ProgA
        CLR     IRQFF           Nollställ avbrottsvippa Event
        CLR     IRQFF+1         Nollställ avbrottsvippa 100Hz
        CLR     EVENT           Nollställ händelseräknaren
        MOVW    #IRQR,IRQVEC
        LDS     #BOSB-9         Stackpekare B efter avbrott
        MOVB    #$C0,0,SP       Init CCR i B-stacken
        MOVW    #ProgB,7,SP     Init returadress till prog B
        STS     SPSAVE          Andra programmets stack
        LDS     #BOSA           Init A-stack
        CLI                     Aktivera avbrott
P_A     BRA     P_A             Evighetsslinga A
```

c)

```
IRQR    LDAA    INPORT          Vilken avbrottskälla?
        ANDA    #1              Bit 0?
        BEQ     HZ100           Nej, processbyte
        CLR     IRQFF           Ja, nollställ avbrottsvippa Event
        INC     EVENT           Öka händelseräknaren
        BRA     IRQEX           Hoppa ut från IRQR
HZ100   CLR     IRQFF+1         Nollställ avbrottsvippa 100Hz
        LDD     SPSAVE          Hämta andra stackpekaren
        STS     SPSAVE          Spara aktuell stackpekare
        TFR     D,SP            Byt stackpekaren till den andra
IRQEX   RTI
```

5.

a) Du använder en korskompilator för HCS12 med följande anropskonventioner för C-funktioner:

- Parameterlistan behandlas från höger till vänster, samtliga inparametrar överförs via processorns stack.
- Lokala variabler behandlas i den ordning de deklareras, dvs sist behandlad finns överst i stacken.
- Varje funktion som har lokala variabler inleds med prologen
  ```
  LEAS   -?,SP
  ```
  och avslutas med epilogen
  ```
  LEAS   ?,SP
  RTS
  ```

- Returparameter lämnas i D- eller B-registret beroende på storlek.

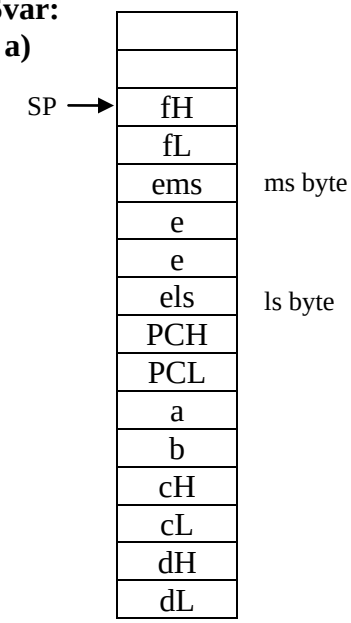
Antag att en funktion definieras på följande sätt:

```
int funca( char a, char b, short c, int d )
{
  long  e;
  short f;
    . . . .
 }
```

Visa stackens innehåll direkt efter det att funktionens prolog har körts.
Platsen för samtliga variabler skall visas. **(3p)**

Svar:

a)



b) Översätt C-funktionen nedan till assemblerspråk för CPU12:
(Antag att anropskonventionerna i a-uppgiften gäller)

```
int funcb(int n) {
  char i;
  int j;
  i = 1;
  j = n;
  while (i < 50 ){
    if (j > 5)
      i = i + 10;
    else
      j = i + j;
  }
return j;
}
```

Svar:

```
int funcb(int n){  _funcb:    LEAS   -3,SP
  char i;
  int j;
  i = 1;                      LDAB   #1
                              STAB   2,SP
  j = n;                      LDD    5,SP
                              STD    0,SP
  while (i < 50 ){    _5:     LDAB   2,SP
                              CMPB   #$32
                              BGE    _6
    if (j > 5)                LDD    0,SP
                              CPD    #5
                              BLE    _7
      i = i + 10;             LDAB   2,SP
                              SEX    B,D
                              ADDD   #$000A
                              STAB   2,SP
                              BRA    _8

    else              _7:
      j = i + j;              LDAB   2,SP
                              SEX    B,D
                              ADDD   0,SP
                              STD    0,SP

                      _8:     BRA    _5

  }
  return j;           _6:     LDD    0,SP
}                             LEAS   3,SP
                              RTS
```

(5p)

6. Metoderna som används för att förbättra minnesprestanda hos primärminnet i en dator bygger på egenskaper hos minnesmodulerna (halvledarkretsar) och på hur programmen använder minnet samt hur data lagras i minnet. Vilka är dessa egenskaper? Hur kan man utnyttja dem? **(3p)**

Svar: Minnesmodulerna har begränsad kapacitet och för lång accesstid. För program och data gäller ”locality of reference in time and space”, vilket innebär att bara en mycket liten del av adressrummet utnyttjas under ett godtyckligt valt kort tidsintervall och att sannolikheten är stor att processorn kommer att använda adresser i närheten av den adress den redan använder. Det innebär att man skulle kunna klara sig med ett betydligt mindre minne under förutsättning att det innehåller den data processorn behöver.

Man kan därför sänka accesstiden genom att sätta in ett litet men mycket snabbt minne (cache) mellan processorn och ”main memory”. Processorn läser och skriver bara i cacheminnet och när data saknas där hämtas saknade data och närliggande data till cache från primärminnet. Sannolikheten är sedan stor att de följande accesserna görs i cacheminnet.

Primärminnesstorleken kan man ”öka” (virtuellt minne) genom att låta hela adressrummet finnas på en hårddisk, medan bara det som används för tillfället finns i det verkliga mindre primärminnet (det fysiska minnet). Enligt samma resonemang som för cacheminnet ovan är sannolikheten stor att data som processorn behöver verkligen finns i det fysiska minnet.

Bilaga 1

Assemblerspråket för CPU12 .

Assemblerspråket använder sig av mnemoniska beteckningar som processorkonstruktören MOTOROLA specificerat för maskininstruktioner och instruktioner till assemblatorn, s k pseudoinstruktioner eller assemblatordirektiv. Pseudoinstruktionerna listas i tabell 1.

Tabell 1

Direktiv	Förklaring
ORG N	Placerar den efterföljande koden med början på address N. (ORG för ORiGin = ursprung)
L RMB N	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adressen L. (RMB för Reseve Memory Bytes)
L EQU N	Ger symbolen L konstantvärdet N. (EQU för EQUates = beräknas till)
L FCB N1, N2	Avsätter en byte för varje argument i följd i minnet. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adressen L. (FCB för Form Constant Byte)
L FDB N1, N2	Avsätter ett bytepar (två bytes) för varje argument i följd i minnet med mest signifikant byte på den lägsta adressen. Respektive bytepar ges konstantvärdet N1, N2 etc. Följden placeras med början på adressen L. (FDB för Form Double Byte)
L FCS ”ABC”	Avsätter en byte för varje tecken i teckensträngen ”ABC” i följd i minnet. Respektive byte ges ASCII-värdet för A B C, etc. Följden placeras med början på adressen L. (FCS för Form Character String)

ASCII-koden

Tabell 2 7-bitars ASCII

0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1	$b_6b_5b_4$ $b_3b_2b_1b_0$
NUL	DLE	SP	0	@	P	`	p	0 0 0 0
SOH	DC1	!	1	A	Q	a	q	0 0 0 1
STX	DC2	“	2	B	R	b	r	0 0 1 0
ETX	DC3	#	3	C	S	c	s	0 0 1 1
EOT	DC4	$	4	D	T	d	t	0 1 0 0
ENQ	NAK	%	5	E	U	e	u	0 1 0 1
ACK	SYN	&	6	F	V	f	v	0 1 1 0
BEL	ETB	‘	7	G	W	g	w	0 1 1 1
BS	CAN	(	8	H	X	h	x	1 0 0 0
HT	EM	)	9	I	Y	i	y	1 0 0 1
LF	SUB	*	:	J	Z	j	z	1 0 1 0
VT	ESC	+	;	K	[Ä	k	{ä	1 0 1 1
FF	FS	,	<	L	\Ö	l	|ö	1 1 0 0
CR	GS	-	=	M	]Å	m	}å	1 1 0 1
S0	RS	.	>	N	^	n	~	1 1 1 0
S1	US	/	?	O	_	o	RUBOUT (DEL)	1 1 1 1