



## TENTAMEN

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

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. **(1p)**
- b) Vilken assemblerinstruktion har den hexadecimala maskinkoden 6B E2 16 24? **(2p)**
- c) Assemblerinstruktionen TFR B,X kan skrivas en alternativ assemblerinstruktion. Vilken? **(1p)**
- d) Översätt assemblerinstruktionen CPY -\$FF,SP till maskinspråk. Visa hur maskinkoden placeras i minnet. **(2p)**
- e) Assemblerinstruktionen BRN \$1000 har operationskoden på adressen 1057H. Visa hur maskinkoden placeras i minnet. På vilken adress utförs nästa instruktion? **(2p)**

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

f) LDAB # $\$C0$   
 CMPB Wadr  
 BGE Hopp **(2p)**

g) LDAB # $\$65$   
 CMPB Wadr (Tänk på att overflow kan inträffa vid jämförelsen!)  
 BPL Hopp **(3p)**

h) Det finns en principiell skillnad mellan asynkron och synkron seriekommunikation. Vilken är denna skillnad? **(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. **(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. **(2p)**

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. **(2p)**

2. a) Skriv en subrutin ADDNIBB 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.

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

(5p)

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

```

LDAA    #$F6
ALOOP   LDX    #$FF9C
XLOOP   IBNE   X,XLOOP
        LBRN   ALOOP
        IBNE   A,ALOOP

```

(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.

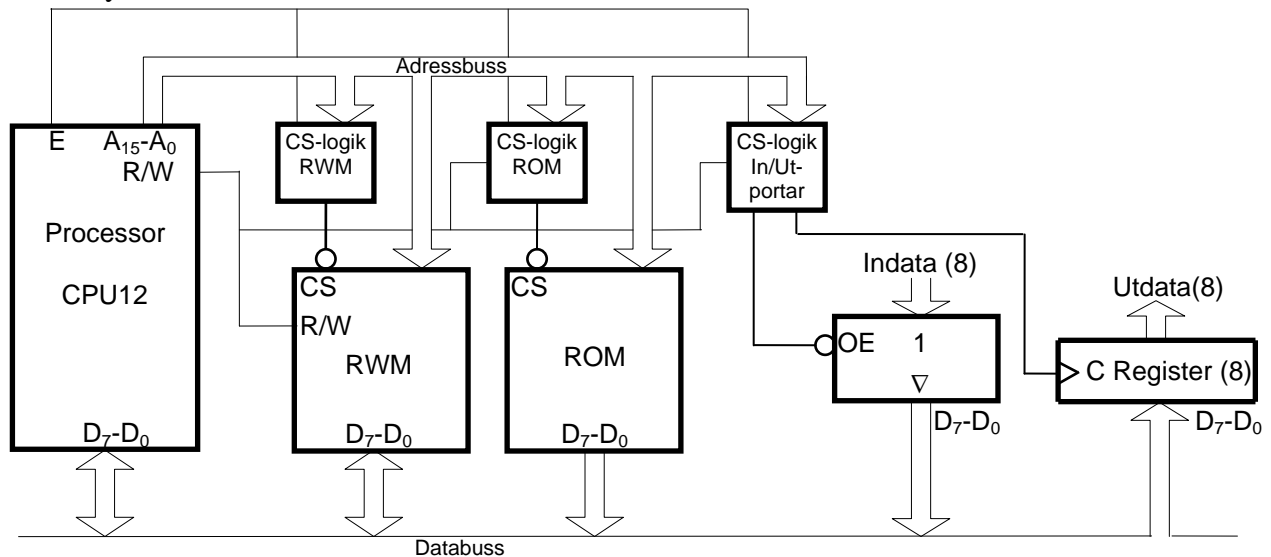
```

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

```

(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 minnesmodulerna. Använd fullständig adressavkodning. Endast grundläggande logikgrindar med valfritt antal ingångar får användas. **(8p)**

4. Avbrottsystemet 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.

- Rita ett kopplingschema 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)**
- 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 avbrottsystemet 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)**
- 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!

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 deklarerats, dvs sist behandlad finns överst i stacken.
- Varje funktion som har lokala variabler inleds med prologen `LEAS -?,SP` och avslutas med epilogen `LEAS ?,SP` följt av `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)**

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;
}
```

**(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)**

## Assemblerspråket för CPU12 .

Assemblerspråket använder sig av mnemoniska beteckningar som processorkonstruktören MOTOROLA specificerat för maskininstruktioner och instruktioner till assemblern, så som pseudoinstruktioner eller assemblerdirektiv. Pseudoinstruktionerna listas i tabell 1.

**Tabell 1**

Direktiv	Förklaring
ORG N	Placerar den efterföljande koden med början på adress 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**

000	001	010	011	100	101	110	111	$b_6b_5b_4$ $b_3b_2b_1b_0$
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