CHALMERS

Institutionen för data- och informationsteknik
Avdelningen för datorteknik

TENTAMEN(med svar och vissa lösningsförslag)

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 2011-03-14 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 2011-04-01 på kursens hemsida.** **Granskning på institutionen 2010-04-05 och 2010-04-06 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 i) -m) avser CPU12.

a) Översätt assemblerinstruktionen LDS #$3000 till maskinspråk och visa hur maskinkoden placeras i minnet. **Svar:** CF 30 00 **(1p)**

b) Vilken assemblerinstruktion har den hexadecimala maskinkoden 18 16? **Svar:** SBA **(1p)**

c) En instruktion med den hexadecimala maskinkoden 0E E9 00 0F 80 är placerad med operationskoden på adressen 1800_{16}. Skriv instruktionen med assemblerspråk
Svar: BRSET -$100,Y,#$0F,$1785. **(3p)**

d) Assemblerinstruktionen SEV kan skrivas som en alternativ assemblerinstruktion. Vilken?
Svar: ORCC #$02 **(1p)**

e) Översätt assemblerinstruktionen CPY $100,SP till maskinspråk. Visa hur maskinkoden placeras i minnet. **Svar:** AD F2 01 00 **(2p)**

f) Assemblerinstruktionen LBRA $8000 har operationskoden på adressen 1200_{16}. Visa hur maskinkoden placeras i minnet. På vilken adress utförs nästa instruktion?
Svar: 18 20 6D FC Nästa instruktion utförs på adressen 8000_{16} efter hoppet. **(2p)**

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

g)
```
LDAA  #$40
CMPA  Wadr
BHI   Hopp
```
Svar: $0 \leq W < 64$ **(2p)**

h)
```
LDAB  #$A5
ADDB  Wadr
BPL   Hopp
```
(Tänk på att overflow kan inträffa vid additionen!)
Svar: $91 \leq W \leq 218$ **(4p)**

i) Vid synkron seriell överföring av data kan data- och klocksignal överföras på separata ledningar, men detta gäller inte alltid. Förklara varför! **Svar:** Om ledningarna är för långa eller om datahastigheten är för hög så anländer data och klocka ej samtidigt till mottagaren. **(1p)**

j) Det finns ett samband mellan längden på en CAN-buss och bithastigheten som kan användas på bussen. Förklara kortfattat detta samband! **Svar:** Om avståndet är för stort mellan noderna kommer löptiden för signalen att bli så stor att noderna inte kan synkroniseras inom bitintervallen. **(2p)**

k) Packa upp flyttalet 43935000_{16}, som är packat enligt IEEE-standard 754-1985 (23 bitar av mantissan och 8 bitars karakteristika) till decimal form. **Svar:** 294,625 **(2p)**

l) Visa approximativt hur många bitar man skulle behöva i mantissan på ett flyttal för att man skall kunna ”lita på” 6 decimala siffror om talet översätts till decimal form.

Svar: $10^6 = (10^3)^2 \approx (2^{10})^2 = 2^{20}$, dvs 20 bitar **(2p)**

m) Vad menas med begreppet ”locality of reference”? **Svar:** 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. **(2p)**

2.

a) STRING1 är en sträng med ett jämnt antal 8-bitars dataord. STRING1 avslutas med två dataord med värdet 0 i konsekutiva (på varandra följande) adresser. Skriv en subrutin ADD2 i assemblerspråk för CPU12, som adderar alla dataorden från STRING1 parvis och placerar summorna som 8-bitars dataord i en ny nollterminerad (ett dataord med värdet 0) sträng STRING2 enligt principen:

$M(X) + M(X+1) \rightarrow M(Y)$
$M(X+2) + M(X+3) \rightarrow M(Y+1)$, osv.

Dataorden i STRING1 är tal utan tecken. Alla additioner som resulterar i overflow skall räknas och antalet sådana additioner skall returneras i D-registret vid återhoppet.

Vid anrop av subrutinen skall startadressen till STRING1 finnas i X-registret och startadressen till STRING2 i Y-registret. Endast D-registret och flaggregistret får vara förändrade vid återhopp. För full poäng skall programmet vara ”korrekt” radkommenterat.

```
*
*   Subrutin ADD2
*
ADD2        PSHX                Spara register
            PSHY
*
            CLRB                Nollställ C-räknare låg byte
            CLR     CCNTH       - ” -       hög byte

ADLOOP      LDAA    1,X+        Hämta databyte från STRING1
            ADDA      1,X+        Addera till nästa byte

            BCC     NOCY        Carry = 0? Ja, ej overflow
            INCB                Öka carryräknare (overflow)
            BNE     NOCY
            INC     CCNTH       Öka hög byte

NOCY        STAA    1,Y+        Placera summan i STRING2
            BNE     ADLOOP      Nästa varv
*                               (Automatisk nollterminering)

            LDAA    CCNTH       Hämta overflowräknare hög byte

            PULY                Återställ register
            PULX
            RTS                 Retur: Antal ”overflow” i D-reg

CCNTH       RMB     1           C-räknare hög byte
```

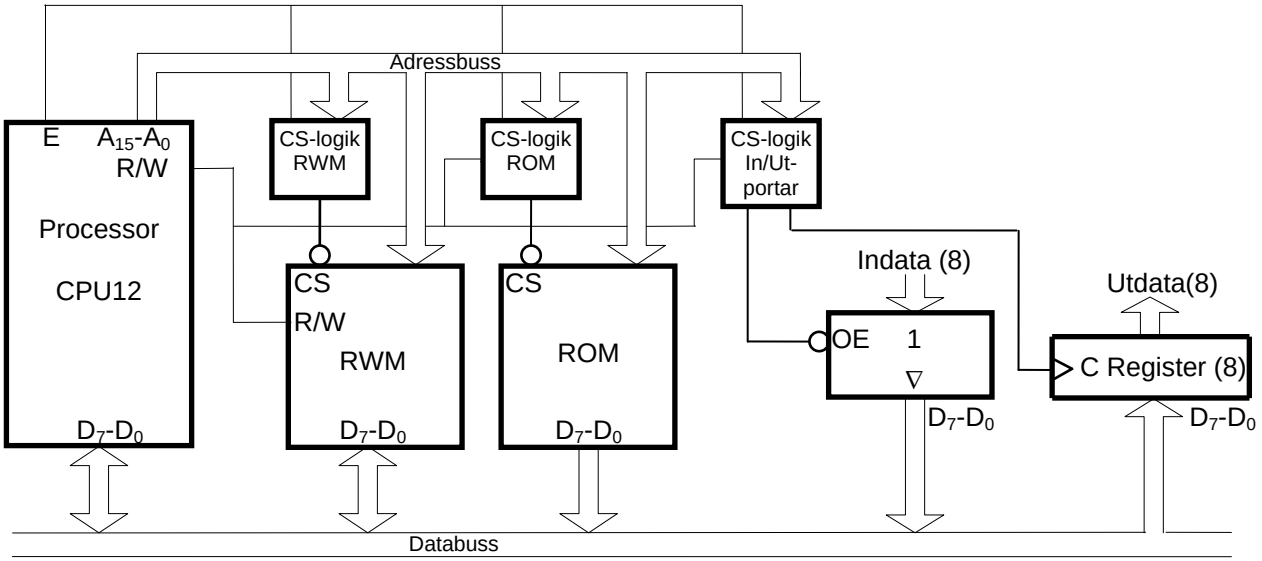
(7p)

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

```
                                  ~
            LDD     #$FFEC        2          (-20)
DLOOP         LDY     #$FFF6        2*20       (-10)
YLOOP         IBNE    Y,YLOOP       3*10*20
            LBRN    YLOOP         3*20
            IBNE    D,DLOOP       3*20
```

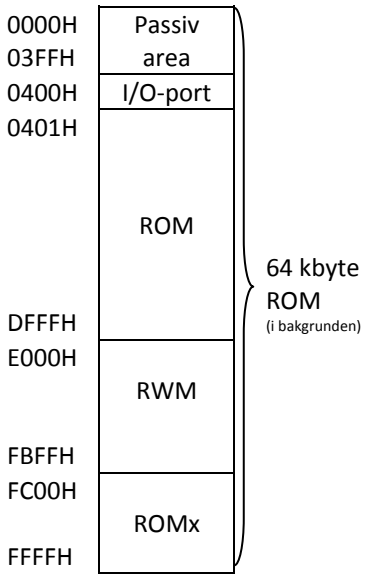
N = 2 + (2 + 3*10 + 3 +3)*20 = 762 st **(3p)**

3. Ett datorsystem visas nedan:



Figuren ovan visar principen för anslutning av externa minnesmoduler och externa in-/utportar till processorn CPU12. Hela adressrummet skall i princip fyllas ut med en 64 kbyte ROM-modul, men på vissa adresser skall in- och utportar och en 8 kbyte RWM-modul vara placerade. Dessutom skall de första 400_{16} adresserna inte aktivera någon port eller minnesmodul vid läsning eller skrivning. RWM-modulen skall i princip ha slutadressen $FFFF_{16}$, men de sista 1024_{10} adresserna skall reserveras för ROM-modulen, som alltså skall prioriteras där. En inport och en utport skall placeras på adressen 400_{16}.

Rita CS-logiken för minnesmodulerna och portarna. Ange adressintervallen för minnesmodulerna. Använd fullständig adressavkodning. Endast grundläggande logikgrindar med valfritt antal ingångar får användas.



Passiv area, 1k = 2^{10}

A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: 0000H =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: 03FFH =	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
			CS													

I/O-port:

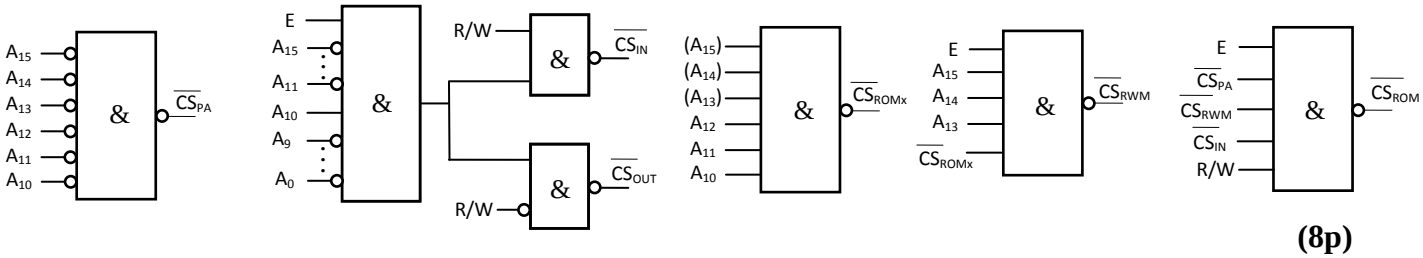
A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Adr: 0400H =	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
						CS										

RWM: 8k = 2^{13}

A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: E000H =	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: FFFFH =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	CS															

ROMx: 1k = 2^{10}

A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Start: FC00H =	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Slut: FFFFH =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
				CS												

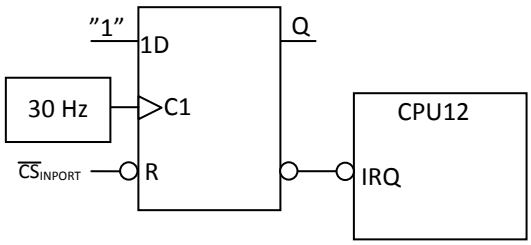


(8p)

4. En dator med processorn CPU12 skall styra en maskin via en utport. I samband med maskinstyrningen behöver man läsa av en parameter som finns tillgänglig som ett 8-bitars tal på inporten INPORT. CS-signalen för inporten (aktiv låg) och en digital signal med frekvensen 30 Hz finns tillgängliga i datorn.

Eftersom datorns huvudprogram är upptaget med beräkningar skall processorns avbrottssystem användas för att effektivisera maskinstyrningen. En färdig subrutin CONTROL sköter all utmatning till utporten.

a) Föreslå en koppling med vars hjälp man kan generera IRQ-avbrott 30 gånger per sekund. D-vippor och standardgrindar får användas. Avbrottssystemet används inte till något annat i datorn.



(2p)

b) Skriv en avbrottsrutin IRQR för IRQ-avbrott. Avbrottsrutinen skall läsa av inporten INPORT och placera det inlästa värdet i variabeln INVAR 30 gånger per sekund. Dessutom skall den utföra subrutinen CONTROL 10 ggr per sekund. En 8-bitars hjälpvariabel får användas. Den lagras i minnet på adressen COUNT. Man får förutsätta att inporten inte läses på något annat ställe än i avbrottsrutinen.

```
IRQRUT  MOVB    INPORT,INVAR    Läs inport och uppdatera INVAR. Nolla vippa.
        DEC     COUNT           10 Hz-räknaren COUNT = 0?
        BNE     IRQEX           Nej, tillbaka till huvudprogram

        MOVB    #3,COUNT        Ja, ominitiera 10 Hz-räknaren
        JSR     CONTROL         Sköt utmaning via CONTROL

IRQEX   RTI                     Tillbaka till huvudprogram
```

(4p)

c) Skriv det avsnitt av huvudprogrammet som initierar avbrottssystemet. IRQ-vektorn finns i ett flyktigt minne på adresserna FFF2H och FFF3H.

```
        (LDS    #BOS            Bottom of stack)
        .
        .
        .
        MOVW    #IRQRUT,$FFF2   Initiera avbrottsvektorn
        MOVB    #3,COUNT        Initiera räknare för 10 Hz
        TST     INPORT          Nollställ avbrottsvippan
        CLI                     Aktivera avbrottssystemet
```

(2p)

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 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` följt av `RTS`.

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

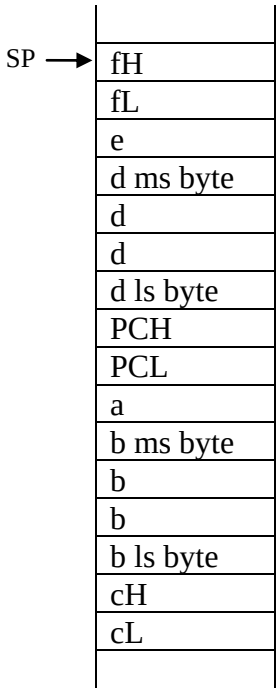
- För XCC gäller dessutom: char 8 bitar, short och int 16 bitar, long 32 bitar.

Antag att en funktion definieras på följande sätt:

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

Visa stackens innehåll direkt efter det att funktionens prolog har körts. Platsen för samtliga variabler skall visas.

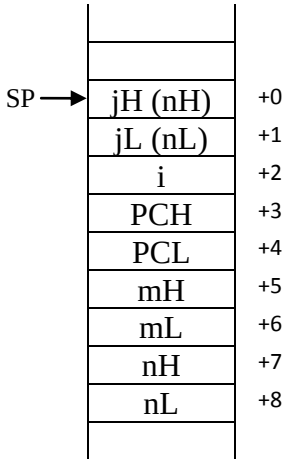
Svar: **(3p)**



5. (forts.)

b) Översätt C-funktionen nedan till assemblerspråk för CPU12. Visa även stackens innehåll innan do-while-satsen börjar utföras. (Antag att anropskonventionerna i a-uppgiften gäller.)

```
int funcb(int m, int n) {          funcb     LEAS   -3,SP
   unsigned char i;
   int j = n;                               LDD    7,SP
                                            STD    0,SP
   do{                             DOLOOP
    for ( i = 0;i < 200; i++ )              CLR    2,SP

                                   FORLOOP  LDAB   2,SP
                                            CMPB   #200
                                            BHS    WHILE

     j = j + m;                             LDD    0,SP
                                            ADDD   5,SP
                                            STD    0,SP
                                            INC2,SP
                                            BRA    FORLOOP

   } while (j <= 10000);           WHILE    LDD    0,SP
                                            CPD    #10000
                                            BLE    DOLOOP

  return j;                                 LDD    0,SP

  }                                         LEAS   3,SP
                                            RTS
```



(6p)

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