CHALMERS

Institutionen för data- och informationsteknik
Avdelningen för datorteknik

TENTAMEN

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	**Lördag 2012-01-14 kl 8.30 – 12.30**
HJÄLPMEDEL	**Av institutionen utgiven** **”Instruktionslista för CPU12” (INS2)** **Tabellverk eller miniräknare får ej användas.**
ANSV LÄRARE: **besöker tentamen**	**Lennart Hansson** **772 1681** **vid flera tillfällen**
DATUM FÖR ANSLAG av resultat samt av tid och plats för granskning	**När rättningen är färdig anslås resultatet med anonyma koder och tid för granskning på kursens hemsida.** **Lägg din anonyma kod på minnet! Den används när resultatet anslås.**
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 g) - i) avser CPU12.

a) Assemblerinstruktionen ORCC #1 kan skrivas på ett alternativt sätt. Visa detta sätt. **(2p)**

b) Vilken assemblerinstruktion har den hexadecimala maskinkoden 64 EB 24 38? **(2p)**

c) Översätt assemblerinstruktionen MOVB 2,X,4,Y till maskinspråk. Visa hur maskinkoden placeras i minnet. **(2p)**

d) Översätt assemblerinstruktionen LBGT $3518 till maskinspråk. Operationskoden finns på adressen 4876_{16}. Visa hur maskinkoden placeras i minnet. **(2p)**

För vilka värden W ($0 \leq W \leq 255$) utförs hoppet i programavsnitten e) och f)?

e)
```
LDAA  #W
NEGA          (Tänk på vad som händer om w = 0!)
CMPA  #$50
BHI   Hopp
```
(3p)

f)
```
LDAB  #$A3
CMPB  #W      (Tänk på att ”overflow” kan inträffa!)
BMI   Hopp
```
(4p)

g) I ett system där CAN-noder kommunierar via en buss finns det en princip för hur en enskild nod kan garanteras ”ensamrätt” när den sänder ett meddelande på bussen. Förklara kortfattat hur denna princip fungerar! **(2p)**

h) Skriv talet 120,875 som ett flyttal enligt IEEE-standard 754 (23 bitar av mantissan och 8 bitars karakteristika). **(2p)**

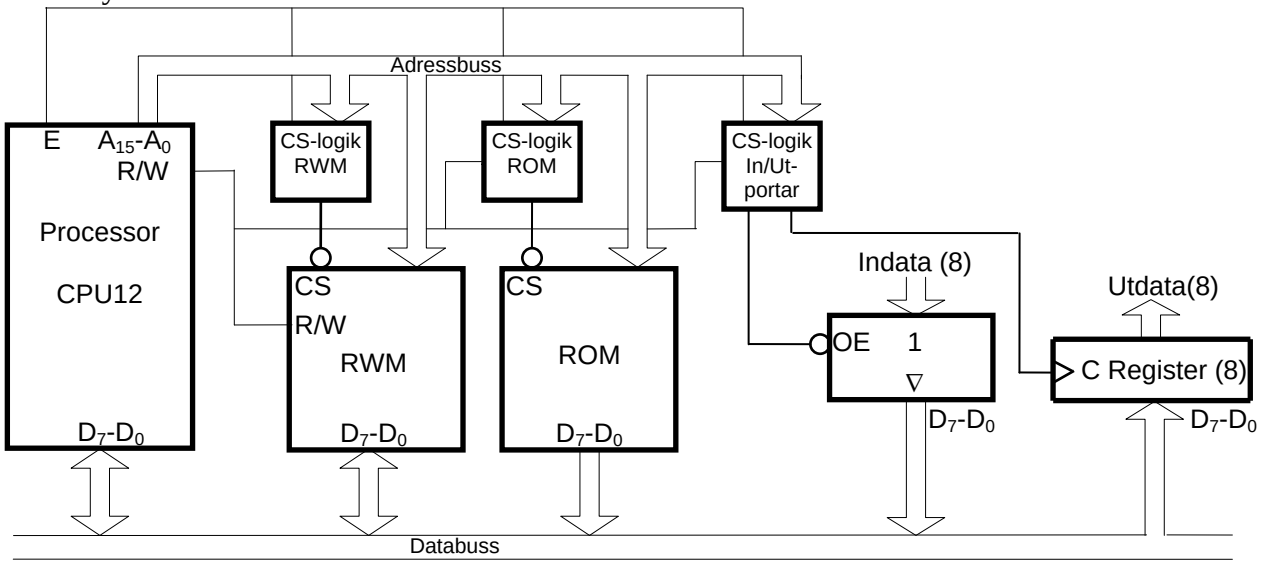
i) I en modern processor delas cacheminnet upp i flera nivåer. Förklara orsaken till detta. **(1p)**

2.

a) Skriv en subrutin (BINASC) i assemblerspråk för CPU12 som översätter det binära talet i bit 0-3 i register B till ASCII-tecknet (7-bitars) för motsvarande hexadecimala tal. Vid återhopp skall ASCII-tecknet finnas i register B med bit 7 nollställd. Endast register B och CC får vara förändrade.
För full poäng på uppgiften skall subrutinen vara korrekt radkommenterad. **(4p)**

b) Skriv en subrutin SWAP för processorn CPU12, som byter plats på databitarna i ett block i minnet så att $b_7b_6b_5b_4b_3b_2b_1b_0$ ersätts med $b_3b_2b_1b_0b_7b_6b_5b_4$ i hela blocket. Vid anrop av subrutinen finns antalet 8-bitars dataord i blocket i B-registret (högst 255 st) och begynnelseadressen i X-registret. Endast flaggregistret får vara förändrat vid återhopp från subrutinen. Skriv subrutinen i assemblerspråk för CPU12. Radkommentarer skall finnas. **(7p)**

3. Ett datorsystem visas nedan:



Figuren ovan visar principen för anslutning av externa minnesmoduler och externa in-/utportar till processorn CPU12. En 32 kbyte RWM-modul skall i princip vara placerad från adress 0, men de första 1024 adresserna i adressrummet skall inte aktivera någon minnesmodul eller port vid läsning eller skrivning. En inport och en utport skall också anslutas. De skall ha samma adress och placeras direkt efter de första 1024 adresserna i adressrummet. Det innebär att inporten skall prioriteras före RWM-modulen på denna adress. Andra halvan av adressrummet skall fyllas ut med en 16 kbyte RWM-modul direkt efter den första RWM-modulen och en 16 kbyte ROM-modul sist i adressrummet.

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

4. En dator med processorn CPU12 skall användas i styrenheten i en maskin. Styrprogrammet skall läsa av två inportar via IRQ-avbrott, PORTA 10 gånger per sekund och PORTB en gång per minut. CS-signalerna för portarna är inte tillgängliga.

Det finns en binär signal med den konstanta frekvensen 400 Hz tillgänglig för generering av avbrott. På adresserna $30F8_{16}$-$30FF_{16}$ aktiveras inga minnesmoduler eller portar. Endast avläsningen av portarna skall vara avbrottsstyrd i systemet.

a) Föreslå en koppling med vars hjälp man kan generera IRQ-avbrott 400 gånger per sekund. D-vippor, AND- och NOT-grindar får användas. Avbrottssystemet används inte till något annat i datorn. **(2p)**

b) Skriv en avbrottsrutin IRQR, som läser av portarna enligt beskrivningen ovan och placerar de avlästa värdena på de symboliska adresserna ADRA och ADRB i minnet. Ledigt utrymme för globala variabler finns på adresserna $1FF0_{16}$-$1FFF_{16}$. **(4p)**

c) Skriv ett avsnitt av huvudprogrammet där IRQ-avbrott initieras. IRQ-vektorn antas vara placerad i RWM på adresserna $FFF2_{16}$ och $FFF3_{16}$. **(3p)**

De symboliska adresserna 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 konventioner för C-funktioner:

- Inparameterlistan behandlas från höger till vänster, samtliga inparametrar överförs via processorns stack.
- Lokala variabler som deklareras placeras på stacken i den ordning de deklareras, dvs sist behandlad finns överst i stacken. Övriga lokala variabler placeras också på stacken i den ordning behovet av dem uppstår, dvs den sista finns överst på 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 och att inga övriga lokala variabler behövs:

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

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

5. (forts.)

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

```
char funcb(char m, char n) {
  char i;
  char j = 0;
  for ( i = 0; i < m; i++ )
   j = j + 2*n;
return j;
}
```

(6p)

c) Vilket värde returneras från funktionen funcb om den anropas med värdena m = 6 och n = 25? Motivera svaret!

(3p)

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

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