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	**Måndag 2013-03-11 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 tel. 772 5718** **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 samt plats 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.** **Onödigt komplicerade lösningar kan ge poängavdrag. Svar på uppgifter skall motiveras.** **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) - j) avser CPU12.

a) Översätt assemblerinstruktionen JMP $2345 till maskinspråk och visa hur maskinkoden placeras i minnet. **(1p)**

b) Assemblerinstruktionen ORCC #1 kan skrivas som en alternativ assemblerinstruktion. Vilken? **(1p)**

c) Översätt assemblerinstruktionen LDAA -25,X till maskinspråk. Visa hur maskinkoden placeras i minnet. **(2p)**

d) En instruktion som har OP-koden på adress 3947_{16} har den hexadecimala maskinkoden 04 51 00. Skriv instruktionen med assemblerspråk. **(3p)**

e) Assemblerinstruktionen BNE $1270 har operationskoden på adressen 1300_{16}. Vad händer när denna instruktion assembleras? **(2p)**

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

f)
```
LDAA  #$70
CMPA  Wadr
BLT   Hopp
```
(2p)

g)
```
LDAA  #$70
CMPA  Wadr
BLO   Hopp
```
(2p)

h) I CAN-system används något som kallas ”stuffbitar”. Vad är detta? Vilken funktion fyller stuffbitarna? **(2p)**

i) Enligt standarden IEEE 754 har flyttalstypen double ordlängden 64 bitar (1 teckenbit, 11 bitars karakteriska och 52 bitars fraction) och är uppbyggd på samma sätt som typen float (32 bitar) med formatet s/c/f. Visa approximativt hur många decimala värdesiffror (giltiga siffror) man får vid översättningen av ett flyttal med full upplösning av typen double till decimal form. **(2p)**

j) Vad är problemet med att använda principen ”fully associative” för cacheminnen? **(2p)**

2.

a) I minnet i ett datorsystem med processorn CPU12 finns ett antal 8-bitars dataord lagrade i en tabell på den kända adressen DTAB och framåt (ökande adress). Tabellen avslutas med talet FF_{16}.

Skriv en subrutin MODIFY i assemblerpråk för CPU12-processorn som inverterar bit 7 och 6, ettställer bit 5 och 4, nollställer bit 3 och 2 samt lämnar bit 1 och 0 oförändrade för alla dataord i tabellen. Slutmarkeringen FF_{16} skall inte påverkas. Endast register CC får vara förändrat vid återhopp från subrutinen. För full poäng skall programmet vara ”korrekt” radkommenterat. **(4p)**

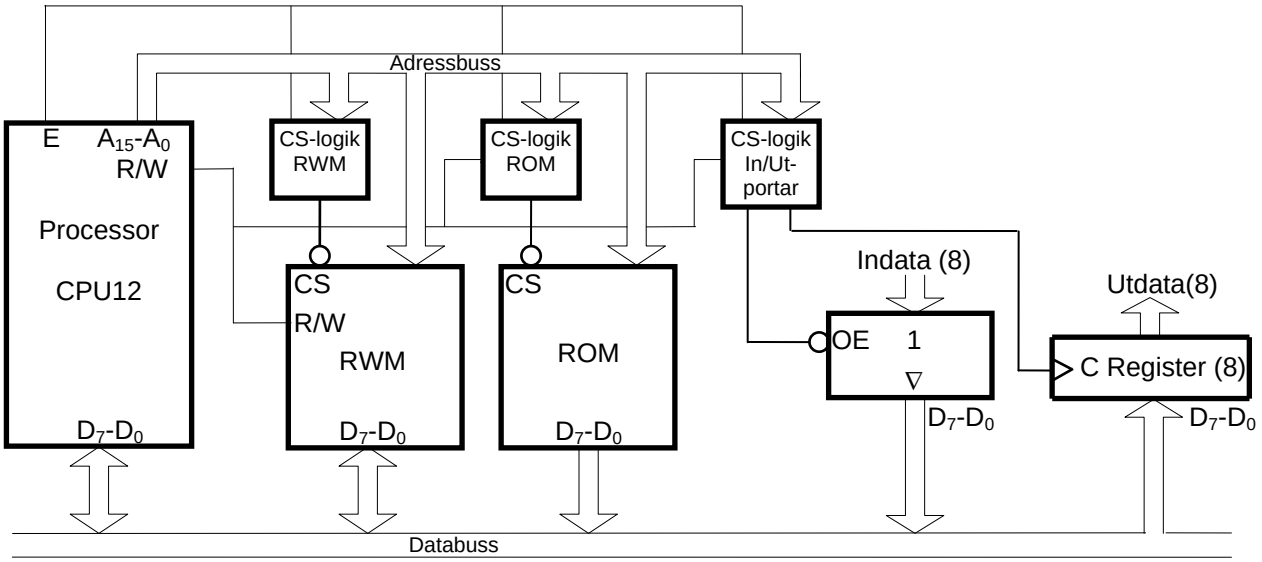
b) Skriv ett avsnitt av ett huvudprogram som anropar subrutinen MIX i c) nedan. Blocket med data i minnet som skall bearbetas börjar på den symboliska adressen BLOCK, som antas vara definierad på annat ställe i programmet. Antalet dataord i BLOCK är 120 st. Stackpekaren antas vara initierad i början av programmet. För full poäng skall programavsnittet vara ”korrekt” radkommenterat.

(2p)

c) Skriv en subrutin MIX i assemblerspråk 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 $b_7b_3b_6b_2b_5b_1b_4b_0$ i hela blocket. Vid anrop av subrutinen finns antalet 8-bitars dataord i blocket i B-registret (högst 255 st) och begynnelse-adressen i X-registret. Endast flaggregistret får vara förändrat vid återhopp från subrutinen. För full poäng skall programmet vara ”korrekt” radkommenterat.

(7p)

3. Ett datorsystem visas nedan:



Figuren ovan visar principen för anslutning av externa minnesmoduler och externa in-/utportar till processorn CPU12.

En 64 kbyte ROM-modul skall i princip vara placerad från adress 0, men de första 256 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 256 adresserna i adressrummet. Det innebär att inporten skall prioriteras före ROM-modulen på denna adress.

En 4 kbyte RWM-modul skall placeras med start på adressen $F000_{16}$, men de sista 256 adresserna skall användas av ROM-modulen.

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

4. En maskin skall styras med hjälp av en dator med processorn CPU12. I samband med detta skall ett 16-bitars värde läsas av var femte sekund på två inportar med adresserna 600_{16} och 601_{16}. Dessutom skall en givare (8 bitar) som är ansluten till en av datorns inportar på adressen 602_{16} läsas av var tionde minut.

Eftersom datorn normalt är upptagen med beräkningsarbete för maskinstyrningen skall avbrott användas för avläsningarna av inportarna. CS-signalerna till inportarna är ej tillgängliga. Adressintervallet 800_{16}-$8FF_{16}$ är helt tomt, dvs. inga moduler är aktiva i detta intervall. Adressintervallet $3CF0_{16}$-$3CFF_{16}$ i RWM är ledigt och kan användas för globala variabler.

Det finns en binär signal med den konstanta frekvensen 100 Hz tillgänglig för lämplig användning.

a) Föreslå en koppling för avbrottsgenerering på processorns IRQ'-ingång. D-vippor med positiv flanktriggning och asynkron resetingång samt standardgrindar med valfritt antal ingångar får användas. Det finns inga andra avbrottskällor i systemet. **(2p)**

b) Skriv avbrottsrutinen IRQR som läser av inportarna enligt beskrivningen ovan och placerar 16-bitarsvärdet på adresserna $3CD0_{16}$ och $3CD1_{16}$ och 8-bitarsvärdet värdet på adressen $3CD2_{16}$ i minnet. **(4p)**

c) Skriv ett avsnitt av huvudprogrammet där IRQ-avbrott initieras. IRQ-vektorn antas vara placerad i RWM på adresserna $FFF2_{16}$ och $FFF3_{16}$. **(2p)**

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 funktionen funca anropas från main och definieras på följande sätt samt att inga övriga lokala variabler behövs:

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

Visa stackens innehåll direkt efter det att funktionens prolog har körts. Platsen för samtliga variabler skall visas. **(2p)**

5. (forts.)

b) Översätt C-programmet nedan (main och funcb) till assemblerspråk för CPU12. Visa även stackens innehåll innan ”do-while”-satsen börjar utföras. (Antag att konventionerna i a-uppgiften gäller.)

```
char funcb(char m, char n);

char k = 10;

void main(){
  funcb(3,30);
}

  char funcb(char m, char n) {
    char j = k;
    unsigned char i;
    do
      for ( i = 0; i < 20; i++ )
      j = j + m;
    while (j <= 100);
    return j+n;
  }
```

(7p)

c) Vilket talvärde returneras från funktionen funcb? Motivera svaret! **(3p)**

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