EDA485 Maskinorienterad programmering Z

Tentamen

Måndag 14 mars 2005, kl. 08.30 - 12.30 i M-salar

Examinatorer
Jan Skansholm, tel. 772 1012
Rolf Snedsböl, tel 772 1665

Kontaktpersoner under tentamen
Som ovan

Tillåtna hjälpmedel

Häftet

Instruktionslista för CPU12

I den får rättelser och understrykningar vara införda, inget annat.

Du får också använda boken

Vägen till C, Bilting, Skansholm, Studentlitteratur

I denna får finnas understrykningar och mindre kommentarer.

Tabellverk och miniräknare får ej användas!

Allmänt
Siffror inom parentes anger full poäng på uppgiften. **Full poäng kan fås om**:

- redovisningen av svar och lösningar är läslig och tydlig. **OBS!** Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift.
- din lösning ej är onödigt komplicerad.
- du motiverat dina val och ställningstaganden
- redovisningen av en hårdvarukonstruktion innehåller funktionsbeskrivning, lösning och realisering.
- redovisningen av en mjukvarukonstruktion i assembler är fullständigt dokumenterad, d v s är redovisad både i strukturform (flödesplan eller pseudospråk) och med kommenterat program i assemblerspråk, om inget annat anges i uppgiften.
- C-program är utformade enligt de råd och anvisningar som givits under kursen. I programtexterna skall raderna dras in så att man tydligt ser programmens struktur. När så anges skall programtexten också vara indelad i moduler med användning av include-filer.

Betygsättning
För godkänt slutbetyg på kursen fordras att både tentamen och laborationer är godkända. Poäng på tentamen bestämmer slutbetyget enligt skalan

20p ≤ **betyg 3** < 30p ≤ **betyg 4** < 40p ≤ **betyg 5**

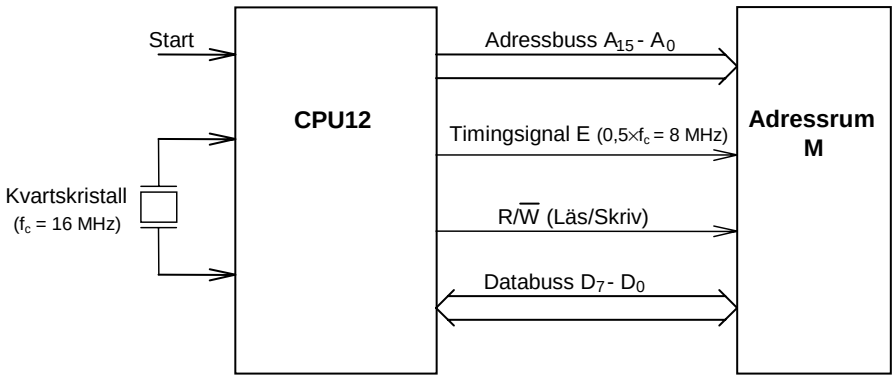
Lösningar
anslås tisdag 15 mars, kl 09.00 på kursens www hemsida.

Betygslistan
anslås såsom anges på kursens hemsida.

Granskning
Tid och plats anges på kursens hemsida.

Institutionen för datorteknik
CHALMERS TEKNISKA HÖGSKOLA

1. I figur 1 visas hur CPU12 kommunicerar med adressrum M.



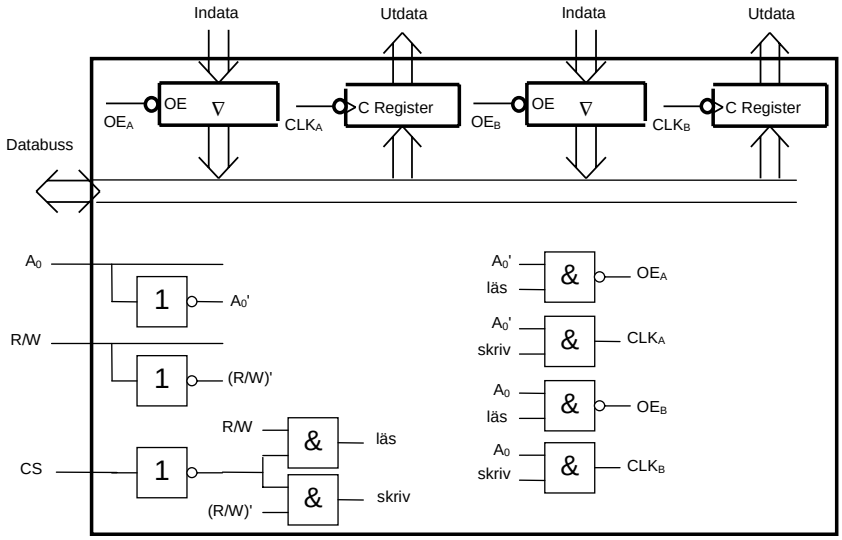
Figur 1

CPU12 har minnesadresserad I/O och skall ha en modul för in- och utmatning, enligt figur 2, som skall ta upp adresserna 0FF0h - 0FF1h.

a. Konstruera CS-logiken för denna in- och utmatningsmodul. Använd fullständig adressavkodning . **(2p)**

b. Varför skapas CLK_A resp CLK_B med en OCH-grind och ej med en NAND-grind? **(1p)**

c. Vilka instruktioner används vanligen för inmatning respektive utmatning av data vid minnesadresserad I/O? **(2p)**



Figur 2

2

I labbet skrev du rutinen ”STOP” och ”OUTZERO”. Skriv dessa rutiner enligt nedanstående specifikation.

Dokumentation: Här räcker det med flödesdiagram och radkommentarer.

```
*Subrutin STOP. Subrutinen stoppar borrmotorn. Borrmotorn stoppas genom att
*subrutinen OUTZERO först matar ut värdet "0" på bit 3 av utporten DCTRL
*och sedan uppdaterar kopian av styrordet, DCCOPY. Endast bit 3 på utporten
*och DCCOPY påverkas.
*Anrop:   JSR  STOP
*Indata:    Inga
*Utdata:    Inga
*Registerpåverkan:Ingen
*Anropade subrut:  OUTZERO

*Subrutin OUTZERO. Läser kopian av borrmaskinens styrord på adress DCCopy.
*Nollställer en av bitarna och skriver det nya styrordet till utporten
*DCTRL och tillbaka till kopian DCCopy. Vilken bit som nollställs bestäms
*av innehållet i B-registret (0-7) vid anrop av subrutinen. Ifall
*innehållet > 7 utförs ingenting.
*
*Anrop:            LDB  #5
*                  JSR  OUTZERO
*Här skall bit nummer 5 i styrordet nollställas
*
*Indata:           B-registret skall innehålla ett tal 0-7, som är numret
                   på den bit i styrsignalordet som skall nollställas.
                   Bitnumrering framgår av följande figur.
```

7	6	5	4	3	2	1	0

```
*Utdata:           Inga
*Registerpåverkan: Ingen
*Anropade subrut:  Inga
```

(6p)

---

3. Avbrottssystemet för CPU12:

a. vilka avbrottsnivåer (typer av avbrott) finns? **(1p)**

b. vilken avbrottsingång har högst prioritet? **(1p)**

c. ge den instruktion som maskerar den maskerbara avbrottsingången! **(1p)**

d. Förutsätt att det finns två yttre enheter (interface) som kan generera avbrott på IRQ-ingången på processorn. Vad krävs för att processorn skall kunna bestämma vilken av dessa två yttre enheter som begärde avbrott. **(3p)**

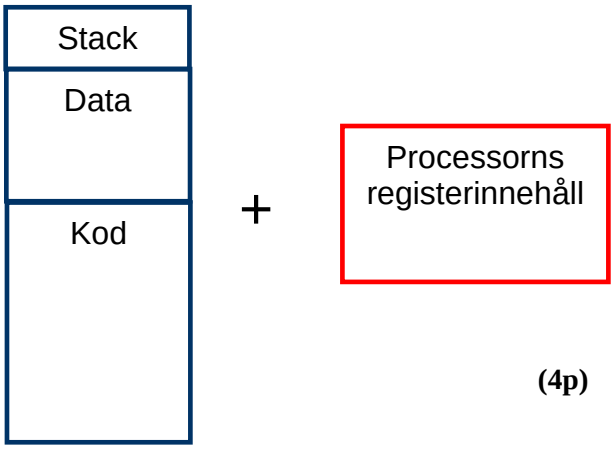
---

4. I ett realtidssystem kännetecknas en exekverande process av det som visas i figur 3.

Förutsätt att två processer exekveras pseudo-parallellt i ett CPU12-system.

Ge exempel på hur processbytet kan vara arrangerat!

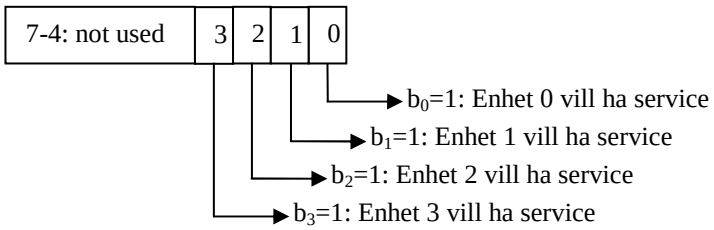
Du skall ej skriva program utan förklara kortfattat med ord hur processbytet går till väga **(4p)**



Figur 3

5. Polling i ett icke avbrottsdrivet system.
Ett CPU12-system skall användas för att betjäna fyra yttre enheter numrerade 0 t o m 3.

Oberoende av varandra kan enheterna begära att bli servade. Begäran om service görs genom att en till enheten hörande statusflagga ettställs. Enheternas statusflaggor, som också numreras 0 t o m 3, har i ordningsföljd samlats i bitarna 0 - 3 av ett statusregister på adress 0C00H. Se figur



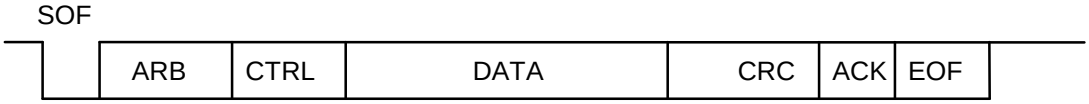
Enheternas servicerutiner finns tillgängliga och har lagrats som subrutiner med namnen DSR0 – DSR3. En enhets statusflagga nollställs i slutet av servicerutinen.

Skriv ett huvudprogram i CPU12 assemblerspråk som oavbrutet övervakar enheterna. De skall "pollas" i "round robin"-ordning. Programmet skall dokumenteras i enlighet med texten på försättsbladet. **(7p)**

6. Frågor om seriell överföring:

a. Hur uppstår ett paritetsfel? **(1p)**

b. Nedan visas ett typiskt CAN-meddelande. Kan du kort ange syfte med de olika fälten i meddelandet? **(2p)**



7. På kortet ML4, som ingår i laborationsutrustningen, finns bl.a. en stegmotor. Från ett program kan man styra denna motor via porten ML4OUT på adressen `0x400`. Från denna port kopplas fyra enpoliga kablar till stegmotorn, på vilken de ansluts till fyra faser som kallas RÖD, BLÅ, GUL och VIT. Kablarna är anslutna till de höga bitarna, b_7-b_4, på utporten ML4OUT enligt följande schema:

Anslutning av faser:	
RÖD	bit 7
BLÅ	bit 6
GUL	bit 5
VIT	bit 4

Stegmotorns axel fås att rotera genom att man styr ut höga och låga signaler till de olika faserna. För att få stegmotorn att rotera ett steg skall 0V kopplas till två av faserna medan de två andra faserna skall kopplas till +5V. Här motsvarar +5V hög signal, dvs. en etta på ML4OUT och 0V låg signal, en nolla på ML4OUT. Riktningen som stegmotorn roterar framgår av följande tabell.

Stegmotorns rotationsriktning				
Fas	MEDURS → ← MOTURS			
BLÅ	GND	GND	+5V	+5V
GUL	+5V	+5V	GND	GND
RÖD	GND	+5V	+5V	GND
VIT	+5V	GND	GND	+5V

Som tabellen visar finns det fyra tillstånd. Man ansluter först låg nivå (0V) till blå och röd fas samtidigt som faserna gul och vit får en hög nivå (+5V). I nästa tillstånd ändras endast två av faserna, nämligen röd och vit som inverteras. Varje tillståndsövergång innebär att stegmotorn roterar ett steg.
Eftersom stegmotorn behöver en viss tid för att rotera ett steg kan man inte byta tillstånd hur fort som helst. Du kan här anta att 10 ms är en tillräckligt lång tid mellan tillståndsbytena.

Uppgiften är att i C skriva en programmodul bestående av filerna `stegmotor.h` och `stegmotor.c`. I modulen skall det finnas två funktioner: `motor_init`, som initierar stegmotorn så att den sätts i starttillståndet, och `motor_vrid` som roterar stegmotorn ett visst antal steg. Funktionen `motor_vrid` har två parametrar, en som anger antalet steg motorn skall rotera och en som anger om rotationen skall ske medurs eller moturs. Om rotationen skall ske medurs ges denna parameter värdet ’true’ annars ges den värdet ’false’. Varje gång funktionen `motor_vrid` anropas skall motorn börja vrida sig utgående från den position den befinner sig i som resultat av tidigare anrop av funktionen. Du skall också skriva de definitioner som är nödvändiga för att programmet skall kunna komma åt utporten på kortet ML4. I denna uppgift får du förutsätta att funktionen `hold` som du konstruerade på laborationerna är färdigskriven och kan användas. Den ger som du minns en fördröjning med så många millisekunder som dess parameter anger.

(9p)

8. I denna uppgift skall man skriva ett program i C. Det antas att man utvecklar programmet på en vanlig C-kompilator och har tillgång till alla standardfunktioner.

a. Skriv en funktion `konsonant` som undersöker om ett tecken innehåller en konsonant eller inte. Funktionen skall som enda parameter ha en **`char`** som innehåller det tecken som skall undersökas. Som resultat skall funktionen ge ett sanningsvärde. Tänk på att ett tecken kan innehålla både stora och små bokstäver. För att få full poäng på denna uppgift får du inte använda långa if- eller switch-satser där du jämför med en konsonant i taget.
Tips: Använd dig gärna av en standardfunktion.

(5p)

b. I det s.k. rövarspråket dubbleras alla konsonanter och ett ’o’ placeras mellan de dubblerade konsonanterna. Vokaler och övriga tecken är oförändrade. Om man t.ex. översätter texten

```
"hej alla rövare"
```

till rövarspråket så får man

```
"hohejoj alollola rorövovarore"
```

Uppgiften är att skriva ett program som läser in ett meddelande i klartext från tangentbordet och som översätter meddelandet till rövarspråk och lagrar det i en textfil med namnet `hemlig.txt`. För att undersöka om ett tecken är en konsonant skall du använda dig av funktionen `konsonant` från deluppgift a. (Det får du göra även om du inte löst den deluppgiften.) **(5p)**

Bilaga 1 - Assemblerspråket för mikroprocessorn CPU12.

Assemblerspråket använder sig av de mnemoniska beteckningar som processorkonstruktören MOTOROLA specificerat för maskininstruktioner och instruktioner till assemblatorn, s k pseudoinstruktioner eller assemblatordirektiv. Pseudoinstruktionerna framgår av följande tabell:

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 adress L. (RMB för Reserve Memory Bytes)
L EQU N	Ger label L konstantvärdet N (EQU för EQUates = beräknas till)
L FCB N1, N2	Avsätter i följd i minnet en byte för varje argument. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FCB för Form Constant Byte)
L FDB N1, N2	Avsätter i följd i minnet ett bytepar (två bytes) för varje argument. Respektive bytepar ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FDB för Form Double Byte)
L FCS "ABC"	Avsätter en följd av bytes i minnet, en för varje tecken i teckensträngen "ABC". Respektive byte ges ASCII-värdet för A, B, C etc. Följden placeras med början på adress L. (FCS för Form Caracter String)

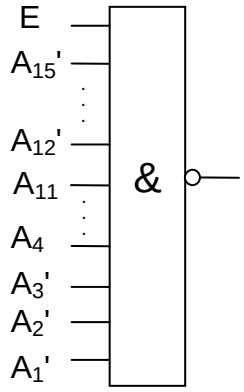
Bilaga 2 - ASCII-koden.

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

Lösningar och svar

1.

a) CS' = (A_{15}'A_{14}'A_{13}'A_{12}'$A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3$'$A_2$'$A_1$'·E)'



b) För att utporten skall laddas med negativ flank och i slutet av pulsen.

c) För inmatning används vanligen LDAA inport och för utmatning STAA utport.

---

2. **a)** Se dina lösningar du gjorde i labbet..

---

3. Avbrottssystemet för CPU12:

a) en icke-maskerbar XIRQ och en maskerbar IRQ

b) XIRQ

c) SEI = ORCC %00010000

d) Jämför med IO-modulen för avbrott i simulatorn och rita en liknande skiss som innehåller:
en avbrottsvippa för varje yttre enhet
och möjlighet att nollställa enhetens vippan
en OR-grind för att bilda en gemensam IRQ-signal till processorn
ett gemensamt (eller två separata) statusregister för att detektera vilken enhet som begärt avbrott

---

4. exempel på gång vid processbytet

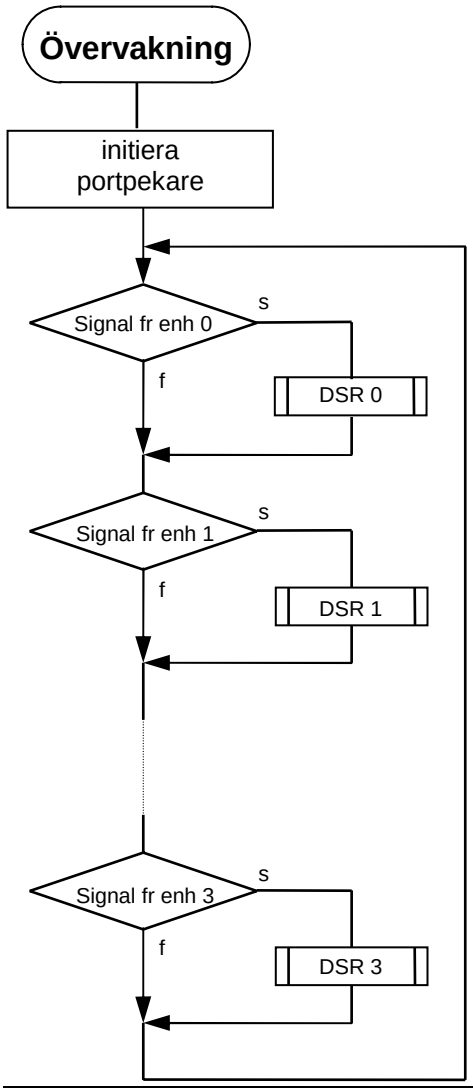
1) klockavbrott

2) registerinnehåll lagras på stack

3) stackpekarens innehåll byts

4) RTI-instruktionen exekveras

5. Övervakningen och betjäningen beskrivs med flödesplan och kommenterat assemblerprogram.



	Assemblerprogram		**Kommentarer**
	LDX	#$0C00	* portadress
L0	LDAA	,X	* läs port och
	BITA	#%00000001	* undersök enhet 0
	BEQ	L1	* om nej, fortsätt
	BSR	DSR0	* om ja, serva enh 0
L1	LDAA	,X	* läs port och
	BITA	#%00000010	* undersök enhet 1
	BEQ	L2	* om nej, fortsätt
	BSR	DSR1	* om ja, serva enh 1
L2	.		
	.		
	.		
L3	LDAA	,X	* läs port och
	BITA	#%00001000	* undersök enhet 3
	BEQ	L4	* om nej, fortsätt
	BSR	DSR3	* om ja, serva enh 3
L4	BRA	L0	* börja om på enh 0

6.

a) ett paritetsfel uppstår när värde på en av tecknets bitar ändrats

b) SOF Start Of Frame (Hård synkronisering)
ARB Arbitrering (tävlan om bussen) Identifierare anger objekttyp; även prioritet
CTRL Control, anger bland ammat antal byte i datafältet etc.
Data Data Max 8 bytes
CRC Kontrollsumma (Tillsammans med felhantering och ACK ⇒ Samtidig exekvering)
ACK Kvitto, Inbyggd handskakning
EOF Slutmarkering

```
// Uppgift 7

// Filen ports.h
#define ML4OUT_ADR 0x400
#define ML4OUT     *((portptr) ML4OUT_ADR)

// Filen stegmotor.h
void motor_init();
void motor_vrid(int antal_steg, int medurs);

// Filen stegmotor.c
#include "stegmotor.h"
#include "ports.h"
#include "clock.h"     // innehåller deklaration av funktionen hold

static port tillstand[] = {0x30, 0xA0, 0xC0, 0x50};
static int aktuellt_tillstand= 0;

void motor_init() {
  aktuellt_tillstand = 0;
}

void motor_vrid(int antal_steg, int medurs) {
  int steg = (medurs) ? 1 : -1;
  int i;
  for (i=0; i<=antal_steg; i++) {
    aktuellt_tillstand = (aktuellt_tillstand + 4 + steg) % 4;
    ML4OUT = tillstand[aktuellt_tillstand];
    hold(10);
  }
}


////////////////////////////////////////////////////////////
// Uppgift 8

#include <stdio.h>
#include <string.h>

int konsonant(char c) {
  return (int)strchr("bcdfghjklmnpqrstvwxzBCDFGHJKLMNPQRSTVWXZ", c);
}

int main() {
  FILE *f = fopen("hemlig.txt", "w");
  int c;
  while ((c=getchar()) != EOF) {
    fputc(c, f);
    if (konsonant(c)) {
      fputc('o',f);
      fputc(c, f);
    }
  }
  fclose(f);
}
```