EDA485 Maskinorienterad programmering Z
DIT150 Maskinorienterad programmering GU

Tentamen

Måndag 12 mars 2007, kl. 14.00 - 18.00 i V-salar

Examinatorer
Stig-Göran Larsson, tel. 772 1693
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 bladet

C Reference Card

samt boken

Vägen till C, Bilting, Skansholm, Studentlitteratur

Även i denna får rättelser och understrykningar vara införda, inget annat.

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. På tentamen fordras 20p, varav minst 10p på datorteknikdelen (uppg 1-4) och 7p på C-delen (uppg 5-6). Tentamen ger slutbetyget:

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

Lösningar
anslås tisdag 7 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 data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

1. ***Adressavkodning***. Konstruera adressavkodningslogiken för ett MC12-system där vi önskar:

- en RWM-area med startadress $4000 och slutadress $4FFF.
- en 128-bytes sammanhängande I/O-area med startadressen $0000.
- en ROM-area med slutadress $FFFF.

Alla chip select signaler är aktiva låga.
Du har tillgång till en 8-kbytes RWM-modul och en 4-kbytes ROM-modul.

Använd fullständig adressavkodning för RWM-modulen och för I/O-arean.

Använd ofullständig adressavkodning för ROM-arean.

Visa hur du resonerat och rita upp adressavkodningslogiken! **(7p)**

---

2. ***Avbrott och assemblerprogrammering med MC12***.
En pulsgenerator är ansluten via en avbrottsvippa till IRQ-ingången på en MC12-system.

Pulsgeneratorn har en frekvens på 100 Hz. För att nollställa avbrottsvippan krävs en läsning på den symboliska adressen IRQCLR. Pulsgeneratorn är den enda avbrottskällan anslutet till IRQ-ingången på processorn.

a. Du skall skriva en avbrottsrutin (IRQCNT) som läser en 8-bitars inport (IRQIN) och adderar det inlästa värdet till en 32-bitars variabel (IRQVAR). Både IRQIN och IRQVAR är variabler på tvåkomplementsform. **(6p)**

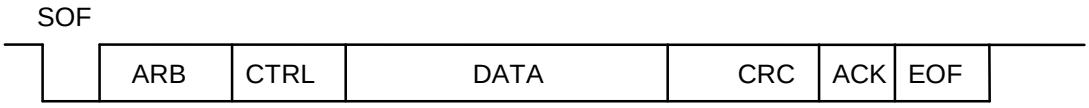
b. Skriv en initieringsrutin IRQINIT som initierar avbrottssystemet och som gör att IRQCNT anropas vid avbrott och att IRQVAR nollställs från början. **(4p)**

---

3. ***Småfrågor***.

a. *Seriekommunikation*. Hur uppstår ett paritetsfel och hur upptäcks det? Motivera! **(2p)**

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



c. *Realtidssystem*. Är CAN-protokollet avsett för tidsstyrda eller händelsestyrda realtidssystem. Motivera ditt svar!

(3p)

---

4. ***Assemberprogrammering***. I en laboration skrev du ett antal assemblerrutiner för borrmaskinutrustningen.

Nedan visas en specifikation av en av rutinerna.

```
* Subrutin OUTONE.
* - läser kopian av borrmaskinens styrord på adress DCCopy.
* - ettställer en av bitarna och skriver det nya styrordet
*   till utporten DCTRL samt tillbaka till kopian DCCopy.
* - biten som ska ettställas ges av innehållet i B-registret (0-7)
*   vid anrop.
* - om (B)> 7 utförs ingenting.
*
* Anrop:  LDB    #bitnummer
*         JSR    OUTONE
*
* Bitnumrering framgår av följande figur.
```

7	6	5	4	3	2	1	0

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

Skriv subrutinen OUTONE i assemblerkod för CPU12! **(6p)**

---

5. Inom matematiken säger man att två vektorer $(u_1, u_2, \ldots u_n)$ och $(v_1, v_2, \ldots v_n)$ är *ortogonala* om summan

$$\sum_{k=1}^{n} u_k v_k$$

är lika med noll. Skriv en C-funktion ortogonala som avgör om två heltalsvektorer är ortogonala. Funktionen skall ge ett sanningsvärde som resultat. Heltalsvektorerna skall representeras med hjälp av två heltalsfält. Utforma funktionen så att den kan hantera vektorer med godtyckliga längder. (De båda vektorerna får dock antas vara lika långa.) I funktionen får du *inte* använda dig av indexering, utan du *måste* använda pekare för att stega igenom fälten.

Skriv också en main-funktion som deklarerar och initierar två heltalsvektorer och som därefter anropar funktionen ortogonala för att se om vektorerna är ortogonala. I så fall skall funktionen main ge utskriften ”ortogonala”.

(8p)

---

6. Deluppgift a.

På den sista laboration konstruerade du en funktion hold som användes för att fördröja exekveringen så många millisekunder som parametern angav. Funktionen använde sig av en global variabel tick som räknades upp av realtidsklockan vid varje klockavbrott. Tiden mellan varje klockavbrott var 10 ms.

Din första uppgift är nu att konstruera en ny funktion wait_for som är snarlik funktionen hold. Den nya funktionen skall ha två parametrar. Den första skall vara en pekare till en **int**-variabel som innehåller ett sanningsvärde som anger om ett viss händelse inträffat eller inte. Funktionen wait_for skall normalt vänta tills denna variabel blir sann. Den andra parametern till wait_for skall vara ett time-out intervall. Funktionen skall vänta högst det antal millisekunder som denna parameter anger. Den andra parametern får vara negativ. I så fall skall det tolkas som om det inte finns någon time-out. Då kan funktionen wait_for vänta hur länge som helst.

Funktionen skall som resultat ge värdet true om den återvände för att den händelse den väntat på inträffat och värdet false om den återvände för att time-out intervallet löpt ut.

(5p)

Deluppgift b.

I ett hus skall ett inbrottsalarm installeras. En infravärmesensor som ger ett avbrott till datorn då en värmekälla detekteras skall användas. Sensorn har ett kombinerat status- och kontrollregister på adressen 1234 (hex). Sensorn startas och stängs av genom att bit nr 0 i registret sätts till 1 resp. 0. När en värmekälla upptäcks sätter sensorn bit nr 1 i registret till 1. Om man har satt bit nr 6 i registret till 1 genererar sensorn även en avbrottssignal och sätter bit nr 7 till 1. Sensorn kan återställas efter ett alarm genom att man sätter bit 7 till 0. (Detta nollställer även bit nr 1.) Avbrottsvektorn för sensorn ligger på adressen FF80.

Den andra uppgiften är att i C skriva ett program som hanterar sensorn och som ger ett larm om sensorn indikerar en värmekälla. För att undvika falskt alarm skall programmet utformas så att det krävs två separata indikationer inom 10 sekunder från sensorn innan larm ges. Du får anta att det finns en färdigskriven funktion raise_alarm som man kan anropa för att ge larm.

Du får använda dig av alla de funktioner och makrodefinitioner du skrev i laboration nr 5, t.ex. funktionerna för att initiera realtidsklockan och för att hantera klockavbrott. Din uppgift är alltså att skriva main-funktionen samt de funktioner som initierar sensorn och hanterar avbrott från den. Använd dig av funktionen wait_for från deluppgift a. (Det får du göra även om du inte löst den deluppgiften.) För enkelhets skull får du anta att det finns en färdig avbrottsrutin med namnet alarmtrap skriven i assembler. Det enda denna funktion gör är att anropa en funktion med namnet alarm_inter. Du måste alltså själv skriva funktionen alarm_inter. Du kan däremot inte förutsätta att adressen till avbrottsrutinen är inlagd i avbrottsvektorn. Det måste du göra själv.

(7p)

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

Preliminära, kortfattade lösningar och svar

1. Minnesmodulerna och I/O-arean i adressrum M tar upp följande adressområden:
ROM-modul. 4kbyte ⇒ $2^2 \bullet 2^{10}$ byte ⇒ 12 Adressbitar ⇒ [A11,A0] direkt till ROM-kapsel.
RWM-modul. 8kbyte ⇒ $2^3 \bullet 2^{10}$ byte ⇒ 13 Adressbitar ⇒ [A12,A0] direkt till RWM-kapsel
I/O-modul. 128 byte ⇒ 7 Adressbitar ⇒ [A06,A0] direkt till I/O-modul

Vi utnyttjar här hälften av den 8kbyte RWM-modulen.
Observera att RWM-modulens insignal A_{12} här kan anslutas direkt till 0 eller 5V.
Studera tabell och CS-logiken nedan

Minnesmodulerna och I/O-portarna tar upp följande adressområden:

			A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
RWM	Start: 4000		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: 4FFF	≠	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
I/O	Start: 0000 -		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: 007F		0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
ROM	Start: F000 -		1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: FFFF		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

CS-RWM: Rita en NAND-grind enligt {A_{15}'· A_{14}· A_{13}'· A_{12}'· E}' (med 0V in på RWM-modulens A_{12})
CS-I/O: Rita en NAND-grind enligt {A_{15}'· A_{14}'·· A_{13}'·· A_{12}'·· A_{11}'·· A_{10}'·· A_9'·· A_8'·· A_7'· E}'

CS-ROM: Rita en NAND-grind enligt {A_{15}· R/W· E}'

2.

```
* Avbrottsrutin IRQCNT läser IRQIN (8 bit) som teckenutvidgas. Teckenbyten
* sparas på stacken för att senare adderas till byte 0 och 1 av IRQVQR
IRQCNT    ldab    IRQIN           läs 8 bitar
          sex     b,d             .. och ändra till 16 bit
          psha                    Spara tecken-byte

          addd    IRQVAR+2        addera bit0-15
          std     IRQVAR+2

          lddd    IRQVAR          höga delen
          adcb    1,s             addera bit 16-23
          stab    IRQVAR+1

          adca    1,s             addera bit 24-31
          staa    IRQVQR

          ldaa    IRQCLR          nollställ avbrottsvippan

          leas    1,s             Städa stacken
          rti



IRQINIT   psha
          pshx

          movw    #0,IRQVAR       Init Var
          movw    #0,IRQVAR+2

          ldaa    IRQCLR          nollställ avbrottsvippan

          ldx     #IRQCNT         avbrottsvektor
          stx     $fff2           (alt 3ff2)
          cli

          pulx
          pula
          rts
```

3.

a. En (eller ett udda antal) bit (/bitar) har ändrats på grund av en störning vid överföring mellan sändare och mottagare.
Paritetsfel upptäcks genom att räkna antal ettställda bitar i mottaget data. Ett seriegränssnitt har oftast en statusbit PE (Parity Error) som indikerar fel.

b. SOF: Indikerar Start; Hård synkronisering för mottagarna ;
ARB: Anger prioritet på meddelandet; Tävlan om bussen
CTRL:StyrInfo; Anger meddelandets längd (antal databyte)
DATA: Överföra databytes (max 8) mellan sändare och mottagare.
CRC: Kontrollsumma- Felupptäckt; Mottagaren vet att inläst data är korrekt
ACK:Kvitto; Sändaren kan se att mottagaren har läst meddelandet
EOF: Slutmarkering

c. CAN är avsett för händelsestyrda realtidssystem.

I ett tidsstyrd realtidssystem är det tiden (klockan) som anger vilken aktivitet som skall startas då dessa (processer) är schemalagda innan ”run-time”. Här passar passar därför ett tidsstyrd protokoll bäst (Token, TDMA etc).
I ett händelsestyrd realtidssystem startas aktiviteter (processer) sporadiskt beroende på yttre händelser. Här passar CAN bra.

4.

```
Max       equ    7

OUTONE    pshd
          pshx
          pshc

          cmpa      #Max        B>7?
          bhi       OExit       .. om JA

          ldx       #OneTab     Pekare
          ldaa      b,x         .. hämta mönster

          ora       DCCopy      Uppdatera styrord
          staa      DCCopy      .. och kopia
          staa      DCtrl

OExit     pulc
          pulx
          puld
          rts

OneTab    fcb       1,2,4,8,$10,$20,$40,$80
```

5.

```
#include <stdio.h>

int ortogonala(const int *v1, const int *v2, int n) {
  int sum=0;
  const int *p1, *p2;
  for (p1=v1, p2=v2; p1<v1+n; p1++, p2++)
    sum += *p1 * *p2;
  return sum==0;
}

int main() {
 int a[] = {1, 0, -2};
 int b[] = {4, 9, 2};
 if (ortogonala(a, b, 3))
   printf("ortogonala");
}
```

6.

```
// deluppgift a

int wait_for(int *condition, time_type max_time) {
// vänta tills *condition är sann eller högst max_time ms
// om max_time < 0 finns ingen timeout
 time_type stop;
 stop = tick + max_time / 10;
 if (max_time > 0)
   while(!*condition && tick < stop)
     ;
 else
   while(!*condition)
     ;
 return *condition;
}
```

```
// deluppgift b

// I filen alarm_ports.h
#define SENSORREG_ADR     0x1234
#define SENSORREG         *((portptr) SENSORREG_ADR)

#define SENSOR_VEC_ADR      0xFF80   // Adress till avbrottsvektor
#define SENSOR_VEC          *((vecptr) SENSOR_VEC_ADR)

#define sensor_on_bit       0x01
#define sensor_intr_bit     0x40
#define sensor_reset_bit    0x80

// I filen alarm.c
#include "ports.h"
#include "clock.h"
#include "alarm_ports.h"
#include "alarminter.h"   // innehåller deklaration av alarmtrap

void raise_alarm();

static int alarm_signalled;

void init_alarm() {
  port shadow = 0;
  alarm_signalled = 0;
  SENSOR_VEC = alarmtrap;
  set(shadow, sensor_on_bit);
  set(shadow, sensor_intr_bit);
  SENSORREG = shadow;
}

void alarm_inter(void) {     // Anropas vid avbrott
  clear(SENSORREG, sensor_reset_bit);
  alarm_signalled = 1;
}


int main() {
  init_clock();
  init_alarm();
  while (1) {
    if (alarm_signalled) {
      alarm_signalled = 0;
      if (wait_for(&alarm_signalled, 10000)) {
        alarm_signalled = 0;
        raise_alarm();
      }
    }
  }
}
```