EDA485 Maskinorienterad programmering Z

Tentamen

Torsdag 31 augusti 2006, kl. 08.30 - 12.30 i V-salar

Examinatorer
Rolf Snedsböl, tel. 772 1665

Jan Skansholm, tel. 772 1012

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" av Bilting och 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. Tentamen omfattar 50p. För godkänt 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 fredag 1 september 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. I figur 1 visas modellen för en portenhet med såväl dataregister som statusregister.

a. Är det en inport eller en utport? Hur avgör du det? **(1p)**

b. Visa med Booleska uttryck hur de interna signalerna $\overline{CS}_D$ och $\overline{OE}_S$ bildas ur insignalerna CS (chip select), RS (register select) och R/$\overline{W}$ (read/write) **(2p)**

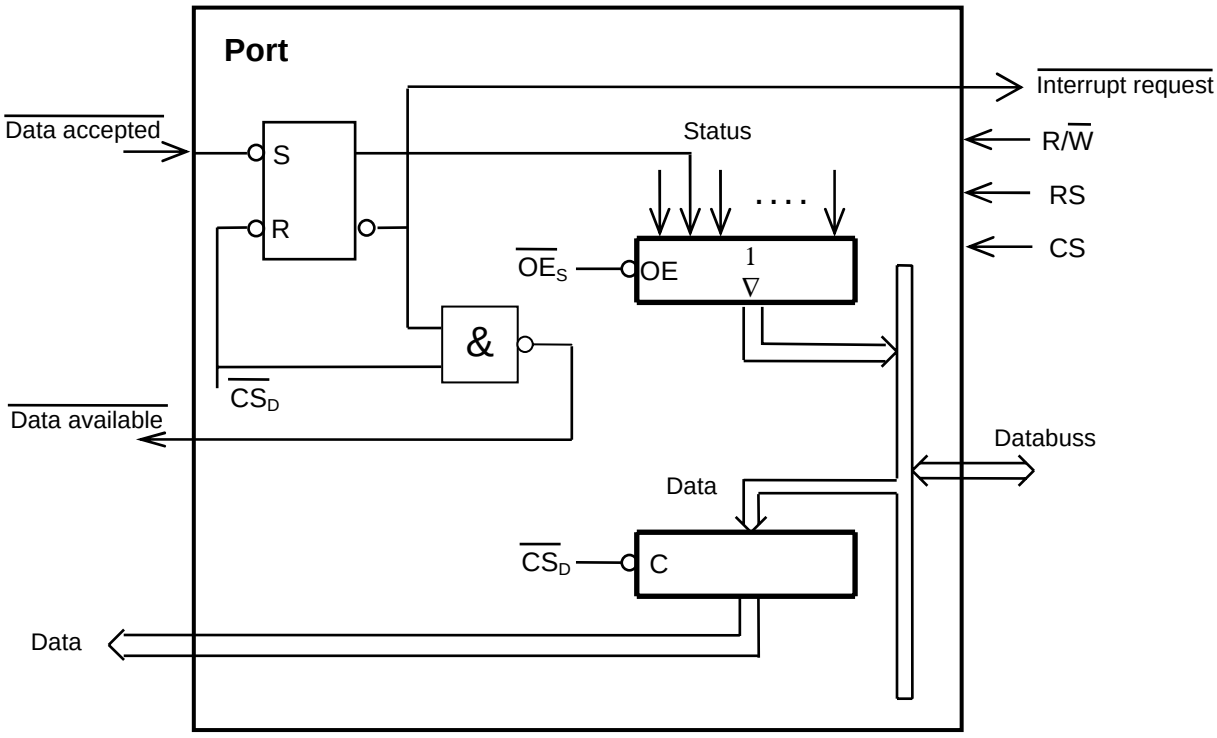
c. Förklara kortfattat med ord vad SR-latchen har för roll. **(2p)**

d. Förutom att gå till dataregistret går $\overline{CS}_D$ även till SR-latchen. Varför? **(2p)**

En dator har minnesadresserad I/O och skall ha en modul, enligt figur 1, som **enbart** skall ta upp (använda) adresserna 7FF0h - 7FF1h.

e. Konstruera CS-logiken för denna modul. **(2p)**

f. Vilka CPU12-instruktioner används vanligen för inmatning respektive utmatning av data vid minnesadresserad I/O? **(1p)**



Figur 1

2. Ett CPU12-system skall användas för att betjäna fem yttre enheter numrerade 0 t o m 4.

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 4, har i ordningsföljd samlats i bitarna 0 - 4 av ett statusregister på adress C000H.

Enheternas servicerutiner finns tillgängliga och har lagrats som subrutiner med namnen DSR0 - DSR4. 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. **(8p)**

3. Småfrågor.

a. Vad menas med att ett avbrott är maskerbart? **(1p)**

b. Redogör för hur avbrott går till i ett HC12-system där du har en (endast en) yttre enhet ansluten till IRQ-ingången på HC12:an!

Du skall ge kodexempel på en rutin som initierar systemet för avbrott och kodexempel på en typisk avbrottsrutin. **(5p)**

c. Redogör för begreppen "*Hard*", "*Soft*" och "*Hard Critical Deadline*". **(3p)**

d. Redogör för begreppen CSMA/CD och CSMA/CR som beskriver hur bussåtkomst sker i ett Ethernet respektive ett CAN-nät. **(3p)**

---

4. I C-standarden beskrivs standardfunktionen strncat på följande sätt:

***char** *strncat(**char** *s1, **const char** * s2, size_t n);*

Description

The strncat function appends not more than n characters from the array pointed to by s2 to the end of the string pointed to by s1. The initial character of s2 overwrites the null character at the end of s1. A terminating null character is always appended to the result. (Thus, the maximum number of characters that can end up in the array pointed to by s1 is

strlen(s1)+n+1.) If copying takes place between objects that overlap, the behavior is undefined.

Returns

The strncat function returns the value of s1.

Skriv en egen version av funktionen strncat. Du får *inte* använda indexering, utan måste använda dig av pekare. Du får *inte* heller använda dig av någon annan standardfunktion, utan måste skriva *allt* själv.

(8p)

---

5. Antag att en dator används för enkel tidtagning vid en idrottstävling. Till datorn finns kopplat två sensorer samt en klockkrets. (Dessutom finns en display, men den behöver inte programmeras i denna uppgift.) Den första sensorn känner när startskottet går och den andra när den tävlande passerar mållinjen. De två sensorerna är kopplade till *samma* 16-bitars styrregister. Detta ligger på adressen 1234 (hex) och adressen till dess avbrottsvektor är FF80. Styrregistret aktiveras och inaktiveras genom att bit nr 0 i det sätts till 1 resp. 0. Om registret är inaktiverat påverkas det inte av inkommande signaler, men om det är aktiverat gäller följande: När en signal kommer från någon av de två sensorerna sätts bit nr 7 i registret till 1. Om man har satt bit nr 6 i registret till 1 genereras då även en avbrottssignal till processorn. Styrregistret skall återställas efter ett avbrott genom att man sätter bit 7 till 0.

Klockkretsen är kopplad till ett annat 16-bitars styrregister, vilket ligger på adressen 1230 (hex). Adressen till dess avbrottsvektor är FF70. Detta styrregister har samma konfiguration och fungerar på samma sätt som styrregistret för sensorerna. Den enda skillnaden är de inkommande signalerna kommer från klockkretsen istället för sensorerna. Klockkretsen genererar 500 signaler per sekund.

Uppgiften är att skriva ett C-program som gör en tidsmätning. När programmet startar skall det visa tiden 0 på en display och vänta tills startskottet går. När detta sker skall klockan aktiveras och tiden skall visas fortlöpande på displayen. Displayen skall visa tiden uttryckt i hundradels sekunder och den visade tiden skall uppdateras hundra gånger per sekund. När den tävlande passerar mållinjen skall klockan stoppas och sluttiden visas konstant på displayen. Programmet behöver bara klara en tidsmätning. (Vill man göra en ny får man starta om programmet genom att trycka på reset-knappen.)

Du får förutsätta att det finns en färdigskriven C-funktion med namnet display. Denna har en parameter av typen **long int** och när den anropas visar den parameterns värde på en display.

Du får också förutsätta att följande två färdigskrivna assemblerrutiner finns:

```
                segment  text
                define   _clocktrap
                define   _sensortrap

_clocktrap:     JSR      _clockinter
                RTI

_ sensortrap:   JSR      _sensorinter
                RTI
```

Det finns också en färdigskriven assemblerrutin som anropar funktionen man när processorn startar.

Skriv resten av programmet (i C). Den kompilator du använder utnyttjar 16 bitar för typen **int** och 32 bitar för typen **long int**. Du måste skriva *allt* själv. I denna uppgift får du inte använda dig av några av de deklarationer eller hjälpfunktioner som användes under laborationerna.

(12p)

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. a) Man ser att det är en **utport** för att dataregistret

- kan laddas från en för data och status gemensam buss, datorns buss
- alltid har data ut mot omvärlden
- tar emot en signal ”data accepted” från omvärlden och ger ”data available”-signal

b) CS_D- och OE_S-signalerna kan ha följande uttryck

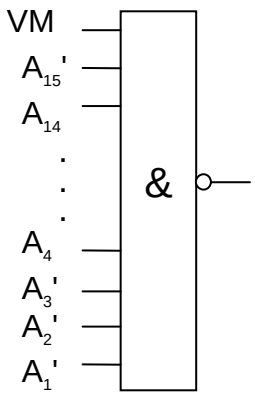
$\overline{CS_D} = \{CS \cdot \overline{RS} \cdot (R/W)'\}'$

$\overline{OE_S} = \{CS \cdot RS \cdot R/W\}'$

c) SR-latchens roll är att vara en länk mellan omvärld och processor. Den skall hålla status från omvärlden, så att en etta i registret indikerar att omvärlden är beredd att ta emot ett ord ifrån porten. Den skall hålla status ifrån processorn så att en nolla indikerar att processorn lagt ut ett nytt ord i portens dataregister.

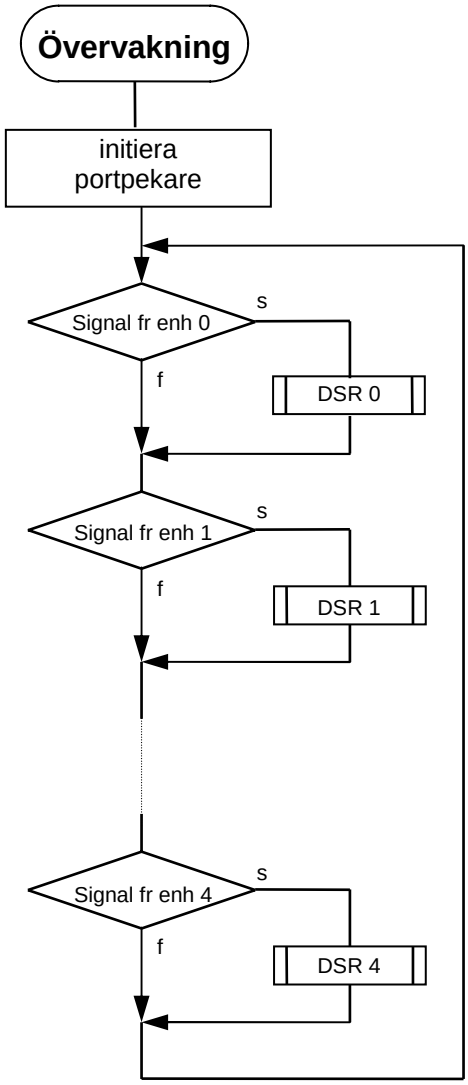
d) CS_D-signalen nollställer SR-latchen och det görs när processorn adresserar dataregistret för att skriva ett nytt ord i detta.

e) $CS' = (A_{15}'A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3'A_2'A_1' \cdot VMA)'$



f) För inmatning används vanligen LDA inport och för utmatning STA utport.

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



	Assemblerprogram		**Kommentarer**
	LDX	#$C000	* 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	.		
	.		
	.		
L4	LDAA	,X	* läs port och
	BITA	#%00010000	* undersök enhet 4
	BEQ	L5	* om nej, fortsätt
	BSR	DSR4	* om ja, serva enh 4
L5	BRA	L0	* börja om på enh 0

3.

a. Med ett maskerbart avbrott menas att det finns en mekanism att förhindra en avbrottsbegäran att upptäckas av processorn. Avbrottbegäran skall maskeras i situationer då det är olämpligt att den stör pågående programexekvering.

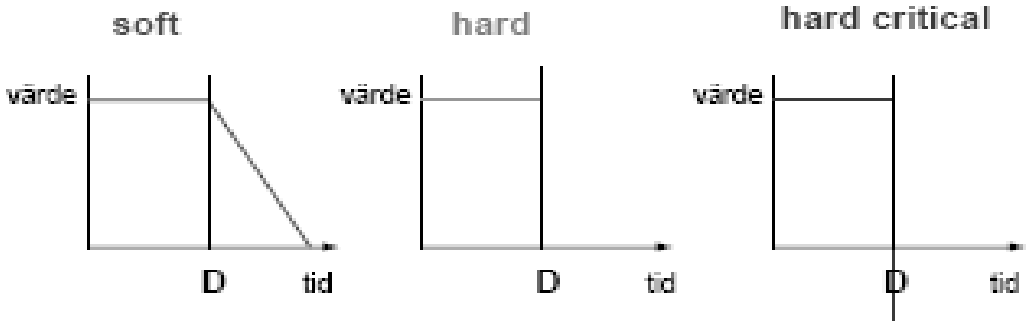
b. -

1	startbit
2	bit 0 i datavärdet
3 – 9	bitarna 1-7 av datavärdet
10	paritetsbit
11	stoppbit
12	stoppbit

- datavärdet är 10101011B

- "framing error" uppstår om sändaren och mottagaren har sådan frekvensskillnad att den sista biten kommer att tolkas fel. Ett exempel på detta är att en förvänta stoppbit får värdet 0.

c. En deadline's värdefunktion beskriver resultatet av det som händer om systemet ej levererar sitt svar i tid: Man skiljer på



Kvalitén hos vissa tjänster kan komma att sänkas.

Resultatet är värdelöst" för systemet.

Resultatet är nödvändigt för undvikande av katastrof.

d. CSMA/CD: Carrier Sense Multiple Access Collision Detect

Lyssna först – Vem som helst kan börja skicka ett meddelande när bussen är ledig – Busskrock upptäcks. Algoritm: Börja om vid fel

CSMA/CR: Carrier Sense Multiple Access Collision Resolution

Lyssna först – Vem som helst kan börja skicka ett meddelande när bussen är ledig – Busskrocksupplösning. Algoritm: Starkast vinner

---

UPG 4

```
char *strstr(const char *s1, const char *s2) {
   const char *p1, *p2;
   for (; *s1; s1++) {
     for (p1 = s1, p2 = s2; *p2 && *p1==*p2; p1++, p2++)
       ;
     if (!*p2)
       return s1;
   }
   return 0;
}
```

UPG5

```
// Deluppgift a
```

```
// Filen monitor.h
#include "clock.h"
void monitor(time_type interval, int (*test)(void), void (*alarm)(void));

// Filen monitor.cpp
#include "monitor.h"
#include "clock.h"
```

```
void monitor(time_type interval, int (*test)(void), void (*alarm)(void)) {
  while (1) {
    hold(interval);
    if (!test())
      alarm();
  }
}


// Deluppgift b

// Filen heart_ports.h
#ifndef HEART_PORTS_H
#define HEART_PORTS_H

// Färdiga Typer
typedef unsigned char port;        // port
typedef unsigned char *portptr;    // pekare till en port
typedef void (**vecptr) (void);    // pekare till avbrottsvektor

// Färdiga Användbara makron
#define set(r, mask)    (r) = (r) | mask
#define clear(r, mask)  (r) = (r) & ~mask

// Egna nya definitioner
#define HEARTREG_ADR     0xD00
#define HEARTREG         *((portptr) HEARTREG_ADR)

#define HEART_VEC_ADR      0xFF90   // Adress till avbrottsvektor
#define HEART_VEC          *((vecptr) HEART_VEC_ADR)

#define POWERREG_ADR     0xD02
#define POWERREG         *((portptr) POWERREG_ADR)

#define on_bit        0x01
#define intr_bit      0x40
#define reset_bit     0x80
#endif

// Filen heart.c
#include "heart_ports.h"
#include "monitor.h"

void call_nurse(void);         // Färdigskriven funktion
void heart_trap(void);         // Avbrottsrutin i assembler
static int beat = 0;

void heart_inter(void) {       // Anropas av rutinen heart_trap
  clear(HEARTREG, reset_bit);
  beat = 1;
}

void init_heart(void) {
  HEART_VEC = heart_trap;
  set(HEARTREG, on_bit);
  set(HEARTREG, intr_bit);
}

int heart_test(void) {
  int b = beat;
  beat = 0;
  return b;
}

void heart_alarm(void) {
  POWERREG = 6 << 4;
  set(POWERREG, on_bit);
  call_nurse();
}

int main() {
  init_clock();
  init_heart();
  monitor(5000, heart_test, heart_alarm);
}
```