DAT015 Maskinorienterad programmering IT
DIT151 Maskinorienterad programmering GU

Tentamen

Måndag 17 december 2007, kl. 14.00 - 18.00 i H-salar

Examinatorer
Jan Skansholm, tel. 772 10 12
Rolf Snedsböl, tel. 772 16 65

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 1-3) och 7p på C-delen (uppg 4-5). Tentamen ger slutbetyget:

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

Lösningar
anslås på kursens www hemsida.

Betygslistan
anslås såsom anges på kursens hemsida.

Granskning
Tid och plats anges på kursens hemsida.

© Rolf Snedsböl och Jan Skansholm, 2007

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

1. Konstruera adressavkodningen för ett MC12-system där vi önskar en yttre ROM-modul och två yttre RWM-moduler (RWM1 och RWM2). Dessutom skall en 256-bytes stor IO-modul finnas. Alla chip select signaler är aktiva låga.

RWM-modulerna (4kByte vardera) skall bilda ett sammanhängande (sekventiellt) minne med start på adress $0000.

ROM-modulen som är 32kByte skall användas för adressområdet [$E000,$FF00].

IO-modulen skall placeras sist i minnet ([$FF00,$FFFF]) och då överlappa ROM-modulen.

a. Rita adressavkodningslogiken för CSROM, CSRWM1, CSRWM2 och CSIO. Använd fullständig adressavkodning. **(5p)**

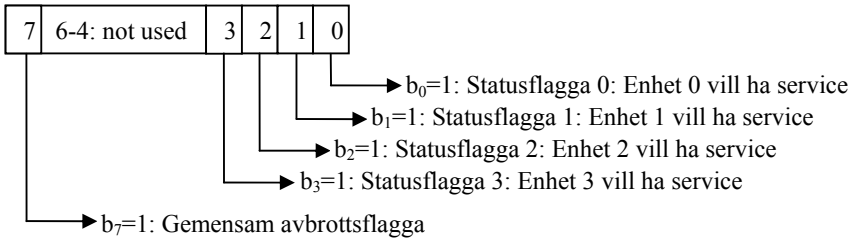
b. Rita ett blockdiagram bestående av processor adressavkodningslogik, minnesmoduler och IO-modul där det framgår hur adressbussen är ansluten mellan enheterna. **(2p)**

c. Rita en bild av processorns adressrum där det framgår hur de olika IO- och minnesmodulerna är placerade där du använder ofullständig adressavkodning för RWM-modulerna. **(2p)**

d. Vad har signalen VMA för syfte i dessa sammanhang. Förklara! **(1p)**

2. *Avbrott och assemblerprogrammering av MC12*

Ett CPU12-system skall bland annat användas för att betjäna fyra yttre enheter numrerade 0 t o m 3. Oberoende av varandra kan enheterna begära att bli servade (begära avbrott). Begäran om service görs genom att en till enheten hörande statusflagga ettställs och att IRQ skickas till processorn. Enheternas statusflaggor, som också numreras 0 t o m 3, har i ordningsföljd samlats i bitarna 0 - 3 av ett statusregister på adress $FF64. Se figur nedan. Bit 7 i statusregistret är en gemensam avbrottsflagga för enheterna 0 t o m 3.



För att kvittera avbrottet krävs en skrivning på adress $FF60 för enhet 0, $FF61 för enhet 1, $FF62 för enhet 2 och slutligen adress $FF63 för enhet 3. Du behöver inte ta hänsyn till den gemensamma avbrottsflaggan när du kvitterar avbrottet.

Enheternas servicerutiner finns tillgängliga och har lagrats som subrutiner med namnen SUB0 – SUB3.

I systemet finns även en realtidsklocka RTC som också är avbrottsdriven (servicerutin RtcINC).

Realtidsklockan har ett statusregister på adressen RtcStat. När bit 7 i statusregistret är ettställd innebär detta att klockan begär avbrott (IRQ skickas till processorn). Bit7 nollställs genom en skrivning till registret (RtcStat) och därmed kvitteras avbrottet.

a. Skriv en subrutin RtcINC som ökar den 32-bitars realtidsklockan med ett. Variabeln hittas på adressen RtcVar. **(2p)**

b. Skriv en avbrottshanterare som undersöker vilken enhet som begärt avbrott och hoppar till respektive subrutin. Enhet 3 har högst prioritet,därefter realtidsklockan och slutligen de övriga enheterna i fallande nummerordning. **(5p)**

c. Skriv en kodsekvens som initierar systemet för avbrott. **(3p)**

© Rolf Snedsböl och Jan Skansholm, 2007

3. Småfrågor.

a. Vad menas med en ”symboltabell” i samband med assemblering. Vad innehåller den? Ge exempel. **(1p)**

b. *Grafik***.** Det fokuserades på ”trianglar” under grafikföreläsningen. Vad innehåller en triangel för information. **(1p)**

c. "Deadline" är ett vanligt begrepp för ett realtidssystem och det beskrivs med en värdefunktion. Vad menar man med en ”deadline’s värdefunktion”? **(2p)**

d. Kommunikation mellan processor och minne sker via adress- och databussar. Vad menas i dessa sammanhang med ett synkront respektive asynkront bussprotokoll? Ange fördelar och nackdelar. **(2p)**

e. *CAN protokollet*. Nedan visas ett typiskt CAN-meddelande. Vad är syftet med ARB-fältet. Beskriv och ge exempel. **(2p)**

f. Det finns hård- (IRQ) och mjukvaruavbrott. Vad skiljer dessa åt och vad har de gemensamt. Ge exempel på användning. **(2p)**

4. Din uppgift är att skriva en C-funktion med följande deklaration:

```
char *trim(const char *s1, char *s2);
```

Funktionen skall kopiera den text `s1` pekar på till det utrymme som pekas ut av `s2`, men vid kopieringen skall alla inledande och avslutande s.k. vita tecken *inte* tas med. Med vita tecken menas mellanslag, tabulator och nyradstecken. Tänk på att det kan finnas vita tecken *inne* i den text som skall kopieras. Dessa tecken skall förstås tas med. Funktionen `trim` skall som resultat ge en pekare till den trimmade kopian av texten. Du får inte använda dig av några av C:s standardfunktioner, utan du måste skriva allt själv. Du får inte heller använda indexering, utan måste utnyttja pekare.

Tips. Leta först framifrån efter första icke vita tecken. Sök sedan upp slutet på texten och leta därifrån bakåt efter sista icke vita tecken. Kopiera sedan den mellanliggande texten till det utrymme som pekas ut av `s2`.

(8p)

5.

En AD-omvandlare är ansluten till en dator. Omvandlaren har 64 analoga ingångar (kanaler), numrerade från 0 till 63. AD-omvandlaren kan bara avläsa en ingång i taget och omvandla den analoga signalen till digital form. Det digitala värdet ges som ett positivt 16-bitars tal. (Enheten är millivolt.) Varje omvandling tar en viss tid och förutsätter att den analoga signalen är stabil medan omvandlingen pågår. Om den analoga signalen är instabil kan en avläsning misslyckas.

AD-omvandlaren kopplas till datorn via ett resultatregister och ett styrregister. Dessa register består båda av 16-bitar och de har de *oktala* adresserna 150000 resp. 150002. Resultatregistret innehåller det avlästa värdet efter en omvandling. Styrregistret används för att initiera omvandlingar och kontrollera AD-omvandlarens tillstånd. Styrregistret innehåller följande bitar: (Se nästa sida)

© Rolf Snedsböl och Jan Skansholm, 2007

Bit	Namn	Användning
0	AD start	När en etta skrivs i detta startas en omvandling
6	Interrupt enable	Om en etta skrivs i detta ges ett avbrott när omvandlingen är klar
7	Done	Skall sättas till 0 före omvandlingen. Sätts automatiskt till 1 när omvandlingen är klar.
8-13	Channel number	I detta anger man vilken av de 64 analoga ingångarna som skall avläsas.
15	Error	Sätts till 1 av AD-omvandlaren om omvandlingen misslyckades.

När man skriver i styrregistret måste man skriva *alla* 16-bitarna på en gång.

Deluppgift a:

Skriv en modul (med en .h file och en .c fil) som innehåller två C-funktioner, `adc_read`, som initierar avläsning av en analog ingång och `adc_get_value` som ger det avlästa värdet som resultat. Du måste också skriva de definitioner av typer och portar som behövs. (Dessa definitioner kan med fördel läggas i en separat .h fil.)

Funktionen `adc_read` skall som parameter få numret på den ingång som skall avläsas. Detta värde skall ligga i intervallet 0 till 63. Om ett felaktigt värde ges skall ingen avläsning initieras. Resultattypen skall vara **`void`**.

Funktionen `adc_get_value` skall ge värdet -1 om avläsningen ännu inte är klar och värdet -9 om AD-omvandlingen misslyckades. Dessa två värden skall definieras som makron med namnen `BUSY` resp. `ERROR` i .h filen så att det anropande programmet kan använda dessa.

(6 p)

Deluppgift b.

Skriv ett program som innehåller två separata processer vilka med jämna tidsintervall avläser var sin analog ingång. Programmet skall vara utformat så att avläsningarna pågår i en timme innan programmet avslutas. Den ena processen skall avläsa ingång nummer 0 varannan sekund och den andra processen ingång nummer 1 var tredje sekund. När en process har initierat en inläsning skall den vänta 50 ms. Om avläsningen då inte är klar skall den vänta ytterligare 50 ms osv. tills avläsningen är klar eller tills ett avläsningsfel uppstått. De avlästa värdena skall normalt ligga i intervallet 20 till 40. Om ett avläst värde ligger utanför dessa värden skall detta rapporteras till en operatör. Du kan anta att det finns en färdigskriven funktion med namnet `warning` som kan användas. Denna har en parameter av typen **`const char *.`** Om avläsningsfel uppstod skall ett speciellt meddelande om detta ges.

Du får anta att den realtidskärna som presenterades på en av föreläsningarna finns tillgänglig. Filen `process.h` visas i bilagan och de funktioner som deklareras i denna kan anropas.

Eftersom AD-omvandlaren bara kan avläsa en ingång i taget måste den skyddas från påverkan medan avläsning pågår. Detta kan man åstadkomma genom att använda sig av en semafor (med ett enda”passertillstånd”) om dess funktioner `signal` och `wait` anropas på rätt sätt i de två processerna.

(6 p)

Bilaga

© Rolf Snedsböl och Jan Skansholm, 2007

```
// Filen process.h
#ifndef PROCESS_H
#define PROCESS_H

#define DEFAULT_STACK_SIZE 128
#define MINIMUM_PRIORITY 1
#define DEFAULT_PRIORITY MINIMUM_PRIORITY+9

typedef struct process_struct process;      // definieras i filen process.c
typedef struct semaphore_struct semaphore;  // definieras i filen process.c
typedef void (*function)(void);

extern void init_processes();
extern process *create_process(function f, int prio, int stack_size);
extern void start_process(process *);
extern process *running_process();
extern int get_process_id(process *);
extern void delay_process(process *p, long int t);  // t ges i ms
extern int  get_process_priority(process *);
extern void set_process_priority(process *p, int prio);
extern void terminate_process(process *p);
extern int terminated(process *p);
extern semaphore *create_semaphore(int init_value);
extern void signal(semaphore *);
extern void wait(semaphore *);

// macron för förenklade funktionsanrop (i brist på överlagrade funktioner)
#define new_process(f)       create_process((f), DEFAULT_PRIORITY,
DEFAULT_STACK_SIZE)
#define get_id()             get_process_id(running_process())
#define delay(t)             delay_process(running_process(), (t))
#define get_priority()       get_process_priority(running_process())
#define set_priority(i)      set_process_priority(running_process(), (i));
#define terminate()          terminate_process(running_process())
#endif
```

© Rolf Snedsböl och Jan Skansholm, 2007

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

© Rolf Snedsböl och Jan Skansholm, 2007

Preliminära, kortfattade lösningar och svar

1. RWM. 4kbyte ⇒ $2^2 \bullet 2^{10}$ byte ⇒ 12 Adressbitar ⇒ [A11,A0] direkt till RWM-kapsel.
ROM. 32kbyte ⇒ $2^5 \bullet 2^{10}$ byte ⇒ 15 Adressbitar ⇒ [A14,A0] direkt till ROM-kapsel. (8k används)

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
RWM1	Start: 0000 -	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: 0FFF	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RWM2	Start: 1000 -	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: 1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM	Start: E000	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: FF00	1	1	1	1	1	1	1	1	x	x	x	x	x	x	x	x
IO	Start: FF00 -	1	1	1	1	1	1	1	1	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

CSRWM1': Rita en NAND-grind enligt {A_{15}'A_{14}'A_{13}'A_{12}'E}'
CSRWM2': Rita en NAND-grind enligt {A_{15}'A_{14}'A_{13}'A_{12} E}'

Låt CSIO göra "deselect" på ROM-modulen.
CSROM': Rita en NAND-grind enligt {A_{15}·A_{14} A_{13}CSIO'·E R/W}'
CSIO': Rita en NAND-grind enligt {$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}$ A_9A_8·E}'

Anslutet till RWM1: [A_{11},A_0], R/W och CSRWM1'

Anslutet till RWM2: [A_{11},A_0], R/W och CSRWM2'

Anslutet till ROM: [A_{12},A_0], R/W och CSRWM1'. Väljer att ansluta [A_{14},A_{13}] till 0V och därmed använda endast 8k av den 32-ks ROM-modulen.

Ofullständig adressavkodning för RWM (exempelvis):
CSRWM1': Rita en NAND-grind enligt {A_{15}'A_{12}'E}'
(Signalen är aktiv för: [$0000,$0FFF], [$2000,$2FFF], [$4000,$4FFF], [$6000,$6FFF])
CSRWM2': Rita en NAND-grind enligt {A_{15}'A_{12} E}'
(Signalen är aktiv för: [$1000,$1FFF], [$3000,$3FFF], [$5000,$5FFF], [$7000,$7FFF])

VMA (Valid Memory Adress). Syfte: att INTE aktivera några chip select signaler under den tid som adressbussen är instabil.

2.

```
* Subrutin RtcINC ökar klockan med ett
* Indata: Global variabel RtcVar
* Utdata: Global variabel RtcVar
* Påverkade register: CC
* Anrop: jsr   RtcINC
RtcINC   pshd
         ldd    RtcVar+2   Inc clock Low part
         addd   #1
         std    RtcVar
         bcc    RIExit
         inc    RtcVar+1   mid part
         bcc    RIExit
         inc    RtcVar     high part
RIExit   puld
         rts
```

© Rolf Snedsböl och Jan Skansholm, 2007

```
* Avbrottshanterare IRQ anropar avbrottssubrutiner i följande
* prioritetsordning: SUB3, RtcINC, SUB2, SUB1 och SUB0.
*
IRQ      ldaa   $FF64      Irq från Enhet3 ?
         bita   #%1000
         beq    Ej3
         jsr    SUB3       Betjäna enhet3
         clr    $FF63      Kvittera avbrott
         bra    IrqExit

Ej3      ldab   RtcStat    Irq från Klocka?
         bpl    EjRtc
         jsr    RtcINC     Betjäna klockan
         clr    RtcStat    Kvittera avbrott
         bra    IrqExit

EjRtc    bita   #%100      Irq från Enhet2?
         beq    Ej2
         jsr    SUB2       Betjäna enhet 2
         clr    $FF62      Kvittera avbrott
         bra    IrqExit

Ej2      bita   #%10       Irq från Enhet1?
         beq    Ej1
         jsr    SUB1       Betjäna enhet 1
         clr    $FF61      Kvittera avbrott
         bra    IrqExit

Ej1      jsr    SUB0       Betjäna enhet 0
         clr    $FF60      Kvittera avbrott

IrqExit  rti


* Kodsekvens som initierar systemet för avbrott
         ldd    #IRQ       Avbrottsvektor
         std    $FFF2
         ldd    #0         Nolla klockan
         std    RtcVar
         std    RtcVar+2
         clr    RtcStat    Nolla avbrottsvippa
         clr    $FF60      Nolla avbrottsvippa
         clr    $FF61      Nolla avbrottsvippa
         clr    $FF62      Nolla avbrottsvippa
         clr    $FF63      Nolla avbrottsvippa
         cli
```

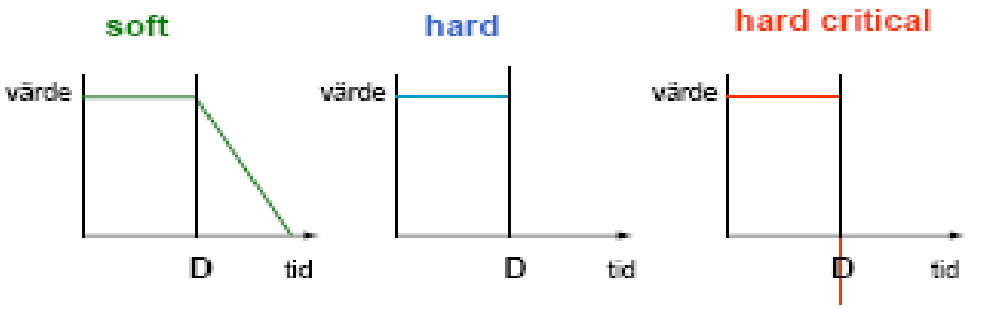
3. Småfrågor

a. Symboltabellen fylls i under pass 1 av assembleringen. Tabellen innehåller Symboler (lägesnamn) och dess motsvarande värden (adresser). Exempel: ”loop” finns på adress $1234.

b. 3 hörn (3*xyz); 3 textur koordinater

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

© Rolf Snedsböl och Jan Skansholm, 2007



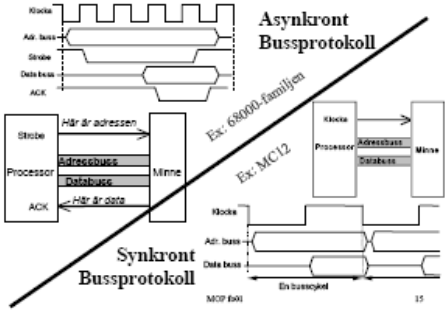
Kvalitén hos vissa tjänster kan komma att sänkas.

Resultatet är värdelöst" för systemet.

Resultatet är nödvändigt för undvikande av katastrof.

c) Asynkront: Handskakning. Olika snabba minnen
Synkront: Enbart klocka. Minnet: snabbare än klockan

d) Syftet med ARB-fältet är att säkerställa att en och endast en nod får ensamrätt om att skicka sitt meddelande i sin helhet. Om två noder samtidigt börjar att skicka var sitt meddelande tävlar de om bussen under sändningen av ARB-fältet. Den som har högsta prioritet får skicka sitt meddelande i sin helhet.



Upg4

```
#include <stdio.h>

char *trim(const char *s1, char *s2) {
  const char *p1;
  char *p2;

  /* Hoppa över inledande vita tecken */
  while (*s1 && *s1 == ' ' || *s1 == '\t' || *s1 == '\n')
    s1++;

  /* Leta reda slutet av texten */
  for (p1=s1; *p1; p1++)
    ;

  /* Sök sista icke-vita tecken */
  for (p1--; p1>s1 && *p1 == ' ' || *p1 == '\t' || *p1 == '\n'; p1--)
    ;

  /* Kopiera till s2 */
  for(p2=s2; s1<=p1;)
    *p2++ = *s1++;
  *p2 = '\0';
  return s2;
}
```

© Rolf Snedsböl och Jan Skansholm, 2007

// Uppgift 5 a

```
// Filen adc.h
typedef unsigned short int port;
typedef unsigned short int *portptr;

#define ADCDATA_ADR 0150000
#define ADCCTRL_ADR 0150002
#define ADCDATA *((portptr) ADCDATA_ADR)
#define ADCCTRL *((portptr) ADCCTRL_ADR)

#define start      0x0001
#define int_enable 0x0040
#define done       0x0080
#define channel    0x3f00
#define error      0x8000

// Filen adc_reader.h
#define BUSY  -1
#define ERROR -9
extern void adc_read(int channel);
extern int  adc_get_value();

// Filen adc_reader.c
#include "adc_reader.h"
#include "adc.h"

void adc_read(int chan) {
  port shadow = 0;
  if (chan >= 0 && chan <= 63) {
    shadow |= start;
    chan <<= 8;
    shadow |= chan;
    ADCCTRL = shadow;
  }
}

int  adc_get_value() {
  if (ADCCTRL & error)
    return ERROR;
  else if (ADCCTRL & done)
    return ADCDATA;
  else
    return BUSY;
}
```

© Rolf Snedsböl och Jan Skansholm, 2007

// Uppgift 5 b

```
    #include "process.h"
    #include "adc_reader.h"

    static semaphore *s;
    static const char *err_msg[] = {"Read error channel 0",    "Read
error channel 1"};
    static const char *ill_msg[] = {"Illegal value channel 0", "Illegal
value channel 1"};

    void watch(int channel_no, int interval) {
      while (1) {
        int i;
        wait(s);
        adc_read(channel_no);
        do {
          delay(50);
          i = adc_get_value();
        } while (i == BUSY);
        signal(s);
        if (i == ERROR)
          warning(err_msg[channel_no]);
        else if ( i <= 20 || i <= 40)
          warning(ill_msg[channel_no]);
        delay(interval);
      }
    }

    void f1(void) {
      watch(0, 2000);
    }

    void f2(void) {
      watch(1, 3000);
    }

    main() {
      process *p1, *p2;
      init_processes();
      s = create_semaphore(1);
      p1 = new_process(f1),
      p2 = new_process(f2);
      start_process(p1);
      start_process(p2);
      delay(3600000L);
      return 0;
    }
```

© Rolf Snedsböl och Jan Skansholm, 2007