EDA485 Maskinorienterad programmering Z

Tentamen

Måndag 10 mars 2008, 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

Kontaktperson under tentamen
Martin Thuresson, tel 772 1678

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 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***. I ett befintligt MC12-system finns en yttre ROM-modul (ROM1), en RWM-modul (RWM1) och en IO-modul (IO). Dessa är placerade på följande adresser.

ROM1	[$E000,$FFFF]
RWM1	[$0000,$3FFF]
IO	[$8000,$80FF]

Fullständig adressavkodningslogik används och dess chip select signaler är tillgängliga (CS_ROM1', CS_RWM1' och CS_IO'). CS-signalerna är aktivta låga.

a) Hur stora (hur många kByte) är de olika modulerna? **(1p)**

b) Rita en bild av processorns adressrum där det framgår hur de olika modulerna är placerade. **(1p)**

c) Utöka systemet med en 2kByte ROM-modul (ROM2) och en 4kByte RWM-modul (RWM2). Konstruera adressavkodningslogiken för dessa nya moduler. ROM1 och ROM2 skall bilda en sammanhängande minnesarea. Även RWM1 och RWM2 skall bilda en sammanhängande minnesarea. Använd ofullständig adressavkodning för att använda så få grindar som möjligt. **(4p)**

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

2. ***Avbrott för HC12.***

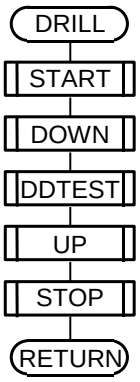
a) En yttre enhet (YE) är ansluten via en avbrottsvippa till IRQ-ingången på ett MC12-system. En skrivning på den symboliska adressen `IrqClr` nollställer avbrottsvippan. En befintlig avbrottsrutin `IrqRut` startas vid IRQ-avbrott.
Skriv en subrutin `IrqInit` som initierar systemet för avbrott **(4p)**

b) Vad menas med "software interrupt"? I vilka sammanhang används det?
Förklara och ge exempel. **(2p)**

c) En avbrottsvektor hittas på adressen $FFF8/$FFF9. Vad används denna till? **(1p)**

d) Rita en skiss över ordningen de olika registren placeras på stacken vid avbrott. **(1p)**

3. ***Assemblerprogrammering för HC12***. Under laboration 2 (borrmaskinslabbet) skrev du rutinen `DRILL` som borrar *ett* hål (se marginalen). `DRILL` anropar i sin tur rutinen `DDTEST` som i sin tur anropar bland annat `ALARM`. *Du skall nu skriva* ***`DDTEST`*** *och* ***`ALARM`*** *enligt specifikationen nedan.* (Övriga rutiner som `DELAY`, `OUTONE` och `OUTZERO` är givna så dessa behöver du inte skriva.)




```
************************************
* Subrutin DDTEST
* Väntar på att borret nått sitt bottenläge.
* Om borret ej har nått bottenläget inom 3 sekunder
* ges två larmsignaler och återhopp sker.
* För att inte borrningen skall fördröjas
* onödigt länge bör man läsa av sensorns
* värde två gånger per sekund.
*
* Anrop:               JSR DDTEST
* Indata:              Inga
* Utdata:              Inga
* Registerpåverkan:    Ingen
* Anropade subrutiner: ALARM
*                      DELAY
************************************
```

Forts nästa sida.

```
************************************
* Subrutin ALARM
* ger NTIMES larmsignaler med längden 1 s och med 0,5 s mellanrum.
*
* Anrop:               LDAB   #NTIMES
*                      JSR    ALARM
* Indata:              Antal larmsignaler, NTIMES, i B-registret.
* Utdata:              Inga
* Registerpåverkan:    Ingen
* Anropade subrutiner: DELAY, OUTZERO, OUTONE
************************************

************************************
* SUBRUTIN – DELAY
* Beskrivning:Skapar en fördröjning om ANTAL x 500 ms.
* Anrop:               LDAA   #6  Fördröj 6*500ms= 3s
*                      JSR    DELAY
* Indata:              Antal intervall,om 500 ms i A
*
* Utdata:              Inga
* Register-påverkan:   Ingen
* Anropad subrutin:    Ingen.
************************************

************************************
* Subrutin OUTZERO. Läser kopian av borrmaskinens styrord på
* adress DCCopy. Nollställer en av bitarna och skriver det nya
* styrordet till utporten DCTRL samt tillbaka till kopian DCCopy.
* Biten som nollställs ges av innehållet i B-registret (0-7) vid anrop.
* Om (B) > 7 utförs ingenting.
* Anrop:               LDAB   #bitnummer
*                      JSR    OUTZERO
*
* Bitnumrering framgår av följande figur.
```

7	6	5	4	3	2	1	0

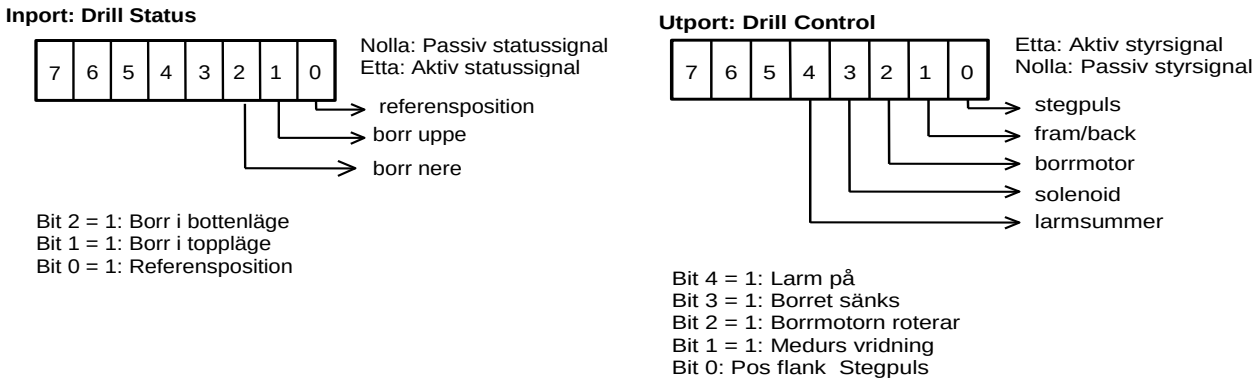
```
* Utdata:              Inga
* Registerpåverkan:    Ingen
* Anropade subrutiner: Inga
************************************

************************************
* 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 ettställs ges av innehållet i B-registret (0-7) vid anrop.
* Om (B)> 7 utförs ingenting.
*
* Anrop:               LDAB   #bitnummer
*                      JSR    OUTONE
*
* Bitnumrering framgår av följande figur.
```

7	6	5	4	3	2	1	0

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

Forts nästa sida.

En sammanställning av borrmaskinens styrregister (DCTRL) och statusregister (DSTATUS) visas i följande figurer:



Din uppgift är alltså att skriva rutinerna **`DDTEST`** och **`ALARM`** enligt specifikationerna ovan. (Övriga rutiner som `DELAY, OUTONE` och `OUTZERO` är givna så dessa behöver du inte skriva.)

(10p)

---

4. ***Småfrågor***

a) *CAN protokollet*. En av ramarna, som skickas i ett CAN-nät, betecknas ERROR-FRAME.

Vilket syfte har en sådan ram?
Vilka noder är det som skickar ramen och vad blir resultatet?
Vad är den stora vinsten med att använda denna ram. **(3p)**

b) *Realtidssystem*. Redogör för hur exekveringstiden hos ett program kan uppskattas med användning av ”programflödesgrafer”. **(2p)**

---

5. ***C-programmering***

Det finns ingen omvandlingsspecifikation för standardfunktionen `printf` som skriver ut ett värde på binär form. Du skall därför konstruera en funktion `tobin` med följande deklaration:

```
void tobin(unsigned int n, char *s);
```

Funktionen skall omvandla värdet i parametern `n` till ett bitmönster (en textsträng bestående av tecknen ’0’ och ’1’) och lägga det i det fält som pekas ut av parametern `s`. Om parametern `n` t.ex. innehåller värdet 5 och en **`unsigned int`** har längden 16 bitar så skall det fält `s` pekar på efter anropet innehålla textsträngen ”0000000000000101”. Funktionen `tobin` skall förutsätta att parametern `s`, när anropet sker, pekar på ett fält som är tillräckligt långt för att rymma alla bitarna i mönstret plus ett avslutande nolltecken. Funktionen `tobin` skall utformas så att den själv räknar ut hur många bitar som krävs för att presentera en **`unsigned int`** i den aktuella datorn. (Den skall alltså fungera både för 16- och 32-bitars **`int`**.)

Skriv också ett litet testprogram som (med hjälp av `scanf`) läser in ett värde till en variabel av typen **`unsigned int`**. Programmet skall sedan anropa funktionen `tobin` för att göra om det inlästa värdet till ett bitmönster. Slutligen skall bitmönstret skrivas ut med hjälp av `printf`.

(8p)

---

6. ***C-programmering***

En signalenhet som läser externa signaler är ansluten till en dator. Signalenheten har 64 ingångar (kanaler), numrerade från 0 till 63. Signalenheten kan bara avläsa en ingång i taget. Det avlästa värdet ges som ett positivt 16-bitars tal. Varje läsning tar normalt högst 2 ms.

Signalenheten kopplas till datorn via ett dataregister och ett styrregister. Dessa register består båda av 16-bitar och de har de *oktala* adresserna 270000 resp. 270002. Dataregistret innehåller det avlästa värdet. Styrregistret används för att initiera avläsningar och kontrollera signalenhetens tillstånd. Styrregistret innehåller följande bitar:

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

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

Deluppgift a: **(6p)**

Skriv en modul (med en .h file och en .c fil) som innehåller två C-funktioner, `sig_read`, som initierar avläsning och `sig_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 `sig_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 `sig_get_value` skall ge värdet -1 om avläsningen ännu inte är klar och värdet -9 om avläsningen misslyckades. Dessa två värden skall definieras som makron med namnen `BUSY` resp. `ERROR` i .h filen så att det anropande programmet kan använda dessa.

Deluppgift b. **(6p)**

Skriv ett program som innehåller två separata processer: en datainsamlare och en användare. Du kan t.ex. tänka dig att programmet ingår som en del i ett större program som kontrollerar ett flygplan. Datainsamlaren samlar kontinuerligt in uppgifter från olika mätpunkter på flygplanet och användarprocessen använder dessa uppgifter för att styra olika reglage.

Datainsamlaren skall med jämna tidsintervall utföra datainsamlingar. Det skall vara 50 ms fördröjning mellan varje datainsamling. Vid varje datainsamling skall processen läsa in data från 16 mätpunkter vilka är anslutna till kanalerna 0 till 15 på den signalenhet som beskrevs i deluppgift a. Medan en datainsamling pågår måste signalenheten skyddas så att ingen annan process kan använda den samtidigt. För detta ändamål skall en semafor utnyttjas. De 16 inlästa värdena skall sparas lokalt i processen medan en datainsamling pågår. Glöm inte att det fordras 2 ms fördröjning vid avläsningen av varje kanal. (Se deluppgift a.) Om inläsningen för en viss kanal inte skulle vara klar eller om värdet skulle vara felaktigt så skall man inte försöka göra om inläsningen för denna kanal, utan istället spara värdet 0.

När en datainsamling är avslutad skall datainsamlingsprocessen kopiera de inlästa värdena till en global variabel `input_data` som skall vara ett fält bestående av 16 komponenter av typen **`unsigned int`**. Medan denna kopiering pågår måste fältet `input_data` skyddas så att ingen annan process kan avläsa eller ändra fältet samtidigt. Använd ytterligare en semafor för att åstadkomma detta skydd.

Den andra processen, användarprocessen, skall ungefär 10 gånger per sekund kopiera fältet `input_data` till en lokal variabel och sedan anropa en extern funktion med namnet `control` som får den lokala kopian som parameter. Under kopieringen skall förstås fältet `input_data` skyddas så att ingen annan process kommer åt det samtidigt. (Du får anta att funktionen `control` är färdigskriven. Du skall alltså inte själv skriva denna.)

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.

Du får förstås anropa funktionerna `sig_read` och `sig_get_value` från deluppgift a även om du inte löst den deluppgiften.

Bilaga till deluppgift b. se nästa sida!

Bilaga till deluppgift b:

```
// 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 unsigned long int get_time();  // resultat ges i ms
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
```

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

Uppgift 1 Adressavkodning.

a) Ritar tabell, utökat med RWM2 och ROM2

		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: 3FFF	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RWM2	Start: 4000 -	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: 4FFF	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
I/O	Start 8000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop 80FF	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
ROM2	Start: D800	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
	Stop: DFFF	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM1	Start: E000	1	1	1	0	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

ROM1: [A12,A0] direkt till ROM-modul ⇒ 13 Adressbitar ⇒ 2^{13} byte ⇒ 8kbyte
RWM1: [A13,A0] direkt till RWM-modul ⇒ 14 Adressbitar ⇒ 2^{14} byte ⇒ 16kbyte
IO: [A7,A0] direkt till IO-modul ⇒ 8 Adressbitar ⇒ 256byte

b) Rita figur

c) ROM2: 2kbyte ⇒ $2^1 \bullet 2^{10}$ byte ⇒ 11 Adressbitar ⇒ [A10,A0] direkt till ROM-kapsel.
RWM2: 4kbyte ⇒ $2^2 \bullet 2^{10}$ byte ⇒ 12 Adressbitar ⇒ [A11,A0] direkt till RWM-kapsel.

CS_ROM2: Rita en NAND-grind enligt {$A_{15} \cdot A_{14} \cdot A_{13}$' ·E R/W}'
CS_RWM2: Rita en NAND-grind enligt {A_{15}'·A_{14}··E}'

d) VMA: förhindrar felaktig aktivering av minnesmoduler under tiden processorn ändrar sin adressbuss.

Uppgift 2 Avbrott.

a)

```
* Subrutin IrqInit initierar avbrottssystemet.
* Indata: -
* Utdata: -
* Påverkade register: -
* Anropade rutiner: -
*
IrqInit   pshc
          movw   #IrqRut,$FFF2  Avbrottsvektor
          clr    IrqClr         Nolla avbrottsvippa
          cli                   Tillåt avbrott
          pulc
          rts
```

b) Mjukvaruavbrott. Programmeraren bestämmer (med instruktionen SWI) när avbrott skall inträffa. Används exempelvis för att generera ett processbyte.

c) Processorn utnyttjar denna avbrottsvektor när den försöker utföra en icke giltig operationskod.

d) Se databladen s 2.

Uppgift 3 Assemblerprogrammering

Rita en Flödesplan för DDTEST! Subrutinhuvud: Se uppgift!

```
Down    equ   %00000100      Borr nere
DDTEST  pshc
        pshd
        ldab  #6             Varvväknare


Wait    ldaa  DSTATUS        Borr nere?
        bita  #Down
        bne   IsDown

        ldaa  #1             Vänta 0.5 s
        jsr   DELAY

        decb                 Vänta mer
        bne   Wait

        ldab  #2             Starta Larm
        jsr   ALARM

IsDown  puld
        pulc
        rts
```

Rita en Flödesplan för ALARM! Subrutinhuvud: Se uppgift!

```
AlarmOn equ   4              Aktiveringsbit för styrregister
NTIMES  equ   0              Offset stack
ALARM   pshc
        pshd
        pshb                 NTIMES till stack

Merlarm ldab  #AlarmOn       Aktivera larm
        jsr   OUTONE
        ldaa  #2             .. i 1s
        jsr   DELAY

        ldab  #AlarmOn       Stanna larm
        jsr   OUTZERO
        ldaa  #1             .. i 0,5s
        jsr   DELAY

        dec   NTIMES,S       Larmat klart?
        bne   Merlarm

        leas  1,s
        puld
        pulc
        rts
```

Uppgift 4

a) Syftet är att skriva sönder/skriva över pågående meddelande och indikera "jag har upptäckt ett fel".

Alla som upptäcker ett fel skickar en ERROR-FRAME.

Resultatet är att när en nod skickar en ERROR-FRAME kommer alla andra noder att detektera någon form för fel i det pågående utskicket och skickar var och en sin egen ERROR-FRAME.

Vinsten är att kan vi få synkroniserad exekvering i de olika noderna

b) En programflödesgraf (eng: Control Flow Graph) kan användas för att representera exekveringsvägarna i ett program. Metoden förutsätter att programmet som ska analyseras är representerade i basala block. Bidragen från de olika blocken summeras då programflödesgrafen reduceras. Minimala såväl som maximala exekveringstider bestäms.

5.

```
#include <stdio.h>
#define N (8 * sizeof(unsigned int))

void tobin(unsigned int n, char *s) {
  int i;
  s[N] = '\0';
  for (i=N-1; i >= 0; i--) {
    s[i] = n % 2 + '0';
    n = n / 2;
  }
}

main() {
  char a[N + 1];
  unsigned int k;
  printf("Skriv ett heltal: ");
  scanf("%u", &k);
  tobin(k,a);
  printf("Talet är: %s", a);
}
```

6. Deluppgift a

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

#define SIGDATA_ADR 0270000
#define SIGCTRL_ADR 0270002
#define SIGDATA *((portptr) SIGDATA_ADR)
#define SIGCTRL *((portptr) SIGCTRL_ADR)

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

// Filen sig_reader.h
#define BUSY  -1
#define ERROR -9
extern void sig_read(int channel);
extern int  sig_get_value();

// Filen sig_reader.c
#include "sig_reader.h"
#include "sig.h"

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

int  sig_get_value() {
  if (SIGCTRL & error)
    return ERROR;
  else if (SIGCTRL & done)
    return SIGDATA;
  else
    return BUSY;
}
```

Deluppgift b

```
#include "process.h"
#include "sig_reader.h"
#define N 16
static semaphore *sig_sem;
static semaphore *data_sem;
static unsigned int input_data[N];

void collector(void) {
  unsigned int a[N];
  while(1) {
    int value, i;
    wait(sig_sem);
    for (i=0; i<N; i++) {
      sig_read(i);
      delay(2);
      value = sig_get_value();
      if (value == ERROR || value == BUSY)
        value = 0;
      a[i] = value;
    }
    signal(sig_sem);
    wait(data_sem);
    for (i=0; i<N; i++)
      input_data[i] = a[i];
    signal(data_sem);
    delay(50);
  }
}

extern void control(int a[]);

void user(void) {
   unsigned int input_copy[N];
   int i;
   while(1) {
     wait(data_sem);
     for (i=0; i<N; i++)
       input_copy[i] = input_data[i];
     signal(data_sem);
     control(input_copy);
     delay(100);
   }
 }
```

```
main() {
  process *p1, *p2;
  init_processes();
  sig_sem = create_semaphore(1);
  data_sem = create_semaphore(1);
  p1 = new_process(collector),
  p2 = new_process(user);
  start_process(p1);
  start_process(p2);
  set_priority(DEFAULT_PRIORITY-1);
  return 0;
}
```