

DAT 015 – Maskinorienterad Programmering 2011/2012

Sammanfattning

“Syftet med kursen är att vara en introduktion till konstruktion och programmering av små inbyggda system.”

Ur innehållet:

- Vi repeterar kursens “lärandemål”
- Diskussion kring “övningstentor”
- Övriga frågor...

1. Programutveckling i C och assemblerspråk

Kunna utföra programmering i C och assemblerspråk samt kunna:

- beskriva och tillämpa modularisering med hjälp av funktioner och subrutiner.
- beskriva och tillämpa parameteröverföring till och från funktioner.
- beskriva och tillämpa olika metoder för parameteröverföring till och från subrutiner.
- beskriva och använda olika kontrollstrukturer.
- beskriva och använda sammansatta datatyper (fält och poster) och enkla datatyper (naturliga tal, heltal och flyttal).

- beskriva och tillämpa modularisering med hjälp av funktioner och subrutiner.

```

EXEMPEL
callfunc( int aa , int ab )
{
  aa = 1;
  ab = 2;
}
XCC12 genererar följande kod:
SEGMENT text
EXPORT _callfunc [r,2]
_callfunc:
; 2 | {
; 3 | aa = 1;
LDD #1
STD 2,SP
; 4 | ab = 2;
LDD #2
STD 4,SP
; 5 | }
RTS
    
```

Funktioners parametrar och returvärdet.

Kompilera följande deklarerationer till assembler och studera assemblerfilen. Vilken skillnad upptäcker du?

```

int a;
static int b;
    
```

Lagringsklass och synlighet.

```

; 1 | int a;
SEGMENT bss
a: RMB $2
EXPORT _a [r,2]
; 2 | static int b;
1: RMB $2
(symbolen _1 existerar endast under
assemblering och motsvarar då
symbolen 'b' i programmet. Symbolen
'b' exporteras inte.
    
```

Subrutiner för att manipulera styrregistret OUTONE och OUTZERO

- * Subrutin OUTONE. Läser kopian av bormaskinens styrdord på adress
- * DDCOPY. Ettställer en av bitarna och skriver det nya styrdordet till utporten DCTRL samt tillbaka till kopian DDCOPY.
- * Biten som nollställs ges av innehållet i B-registret (0-7) vid anrop.
- * Om (B) > 7 utförs ingenting.
- * Anrop: LDAB #bitnummer
- * Utdata: JSR OUTONE
- * Registerpåverkan: Inga
- * Anropade subrutiner: Inga

"bitnummer" = 0..7



```

*****
* SUBROUTIN - 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.
*****
    
```

- beskriva och tillämpa olika metoder för parameteröverföring till och från subrutiner.

Parameteröverföring via register

Antag att vi alltid använder register D, X, Y (i denna ordning) för parametrar som skickas till en subrutin. Då kan funktionsanropet (subrutinanropet)

```
dummyfunc(1a, 1b, 1c);
```

översätts till:

```

LDD 1a
LDX 1b
LDY 1c
BSR dummyfunc
    
```

Då vi kodar subrutinen dummyfunc vet vi (på grund av våra regler) att den första parametern skickas i D, den andra i X och den tredje i Y (osv).

Metoden är enkel och ger bra prestanda. Begränsat antal parametrar kan överföras.

Parameteröverföring via stacken

Antag att listan av parametrar som skickas till en subrutin behandlas från höger till vänster. Då kan

```
dummyfunc(1a, 1b, 1c);
```

Översätts till:

```

LDD 1c
PSHD
; (alternativt STD 2,-SP)
LDD 1b
PSHD
LDD 1a
PSHD
BSR dummyfunc
LEAS 6,SP
    
```

Innehåll	Kommentar	Adressering via SP i subrutinen
1c.lsb	Parameter 1c	6, SP
1c.msb		
1b.lsb	Parameter 1b	4, SP
1b.msb		
1a.lsb	Parameter 1a	2, SP
1a.msb		
PC.lsb	Återhopsadress, placeras här vid BSR	0, SP
PC.msb		

```

dummyfunc:
..
LDD 2,SP
; parameter 1a till register D
..
LDD 4,SP
; parameter 1b till register D
..
LDD 6,SP
; parameter 1c till register D
    
```

Parameteröverföring "In Line"

"In line" parameteröverföring, värdet 10 ska överföras till en subrutin:

```

BSR dummyfunc
FCB 10
...
    
```

```

dummyfunc:
LDAB [0,SP] ; parameter->B
LDX 0,SP ; återhopsadress->X
INX ; modifiera ..
STX 0,SP ; .. tillbaks till stack
..
..
..
RTS
    
```

Returvärdet via register

Register väljs, beroende på returvärdets typ (storlek), HCS12-exempel

Storlek	Benämning	C-typ	Register
8 bitar	byte	char	B
16 bitar	word	short int	D
32 bitar	long	long int	Y/D

En regel (konvention) bestäms och följs därefter vid kodning av samtliga subrutiner

- beskriva och använda olika kontrollstrukturer.

Kontrollstrukturer

If (Villkor) then ...

```
if (Villkor)
{
  Satser;
}
```

if (Villkor) then ... else ... end

```
if (Villkor)
{
  Satser1;
} else {
  Satser2;
}
```

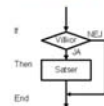
while (Villkor) loop

```
while (Villkor)
{
  Satser;
}
```

loop ... until (Villkor)

```
loop
until (Villkor)
do {
  Satser;
} while (Villkor);
```

If (...) {...}



```
if (DipSwitch != 0)
HexDisp = DipSwitch;
```

Bättre kodning...

```
DipSwitch EQU $600
HexDisp EQU $400
...
TST DipSwitch
BEQ end
LDAB DipSwitch
STAB HexDisp
end;
```

BEQ	"Hopp om iCKE zero	Z=0
BEQ	"Hopp om zero	Z=1

If (...) {...} else { ...}



```
if (DipSwitch == 0)
HexDisp = 1;
else
HexDisp = 0;
```

```
DipSwitch EQU $600
HexDisp EQU $400
...
LDAB DipSwitch
...
TSTB
BEQ not_else
LDAB #0
STAB HexDisp
BRA end
not_else: LDAB #1
STAB HexDisp
end;
```

BEQ	"Hopp om zero	Z=1
-----	---------------	-----

while (...) {...}



```
Delay(unsigned int count)
{
  while (count > 0)
  count = count - 1;
}
```

```
Delay: LDD "count"
Delay_loop:
NOP
...
NOP
SUBD #1
BHI Delay_loop
Delay_end: RTS
```

Testa: väl utan tecken		
BHI	Villkor R=M	C·Z=0
Testa: väl med tecken		
not	Villkor R=M	Z·(N&V)×0

- beskriva och använda sammansatta datatyper (fält och poster) och enkla datatyper (naturliga tal, heltal och flyttal).

```
/*
globals.c
Deklaration av globala variabler
*/
short shortint;
long longint;
int justint;
int intvec[10];

struct {
  int s1;
  char s2;
  char* s3;
} komplex;
```



```
; 1 | short shortint;
; SEGMENT bss
; _shortint: RMB $2
; EXPORT _shortint [r,2]
; 2 | long longint;
; longint: RMB $4
; EXPORT _longint [r,4]
; 3 | int justint;
; _justint: RMB $2
; EXPORT _justint [r,2]
; 4 | int intvec[10];
; _intvec: RMB $14
; EXPORT _intvec [r,20]
; 5 |
; 6 | struct {
; 7 | int s1;
; 8 | char s2;
; 9 | char* s3;
; 10 | } komplex;
; _komplex: RMB $5
; EXPORT _komplex [r,5]
```



```
main() {
  short shortint;
  long longint;
  int justint;

  struct {
    int s1;
    char s2;
    char *s3;
  } typen;
  justint = 0;
}
```

```
SEGMENT text
EXPORT _main [r,2]
_main:
LEAS -13,SP
; 2 | short shortint;
; 3 | long longint;
; 4 | int justint;
; 5 |
; 6 | struct {
; 7 | int s1;
; 8 | char s2;
; 9 | char *s3;
; 10 | } typen;
; 11 | justint = 0;
CLRA
CLRB
STD 5,SP
; 12 |
LEAS 13,SP
RTS
```

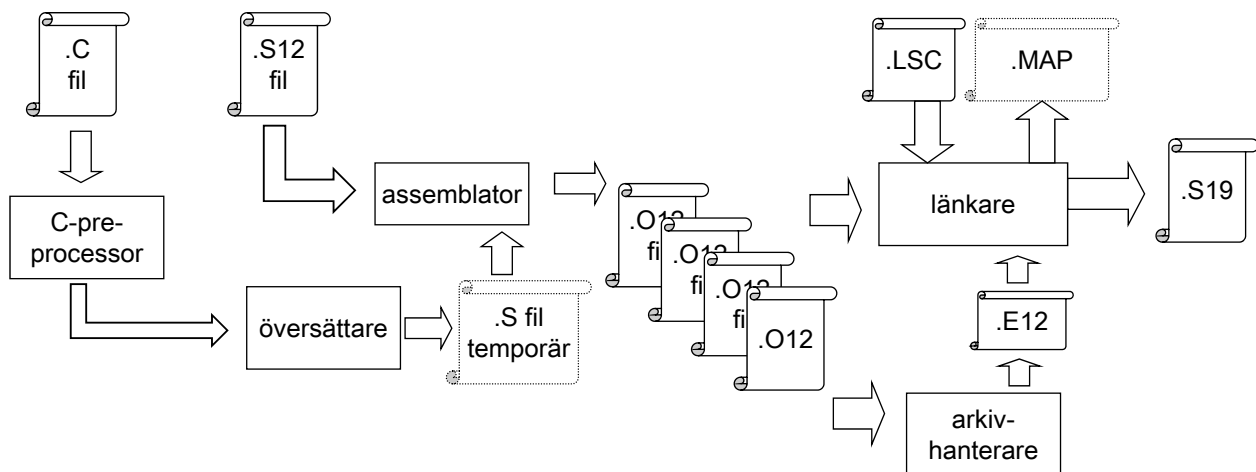
Kunna redogöra för olika lagringsklasser (GLOBAL, STATIC, LOCAL) och "synlighet".

2. Programutvecklingsteknik

Att självständigt kunna:

- beskriva översättningsprocessen, dvs. assemblatorns arbetssätt, preprocessorns användning, separatkompilering och länkning.
- konstruera, redigera och översätta (kompilera och assemblera) program
- testa, felsöka och rätta programkod med hjälp av avsedda verktyg.

- *beskriva översättningsprocessen, dvs. assemblatorns arbetssätt, preprocessorns användning, separatkompilering och länkning.*



- *konstruera, redigera och översätta (kompilera och assemblera) program*
- *testa, felsöka och rätta programkod med hjälp av avsedda verktyg.*

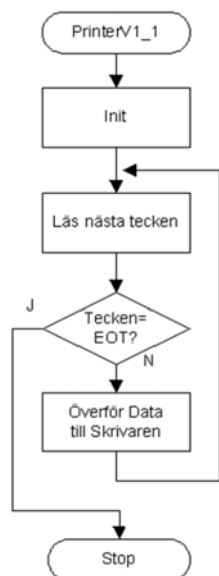
Dessa lärandemål har vi kontrollerat under laborationer.

3. Systemprogrammerarens bild av inbäddade system

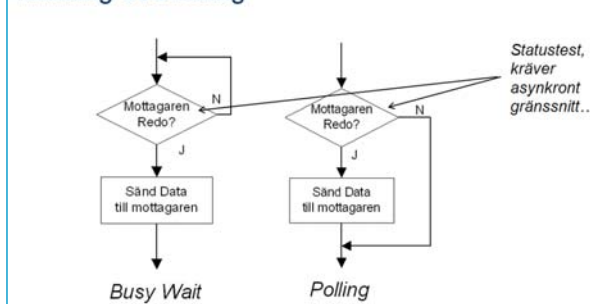
Att självständigt kunna:

- beskriva och tillämpa olika principer för överföring mellan centralenhet och kringenheter så som: ovillkorlig eller villkorlig överföring, statustest och rundfrågning.
- konstruera program för systemstart och med stöd för avbrottshantering från olika typer av kringenheter.
- kunna beskriva metoder och mekanismer som är centrala i systemprogramvara så som pseudoparallell exekvering och hantering av processer.
- beskriva och använda kretsar för tidmätning.
- beskriva och använda kretsar för parallell respektive seriell överföring.

- beskriva och tillämpa olika principer för överföring mellan centralenhet och kringenheter så som: ovillkorlig eller villkorlig överföring, statusstest och rundfrågning.



Villkorlig överföring



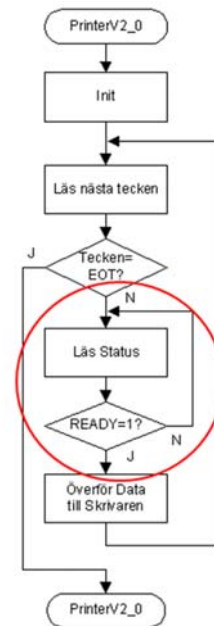
Statusstest, kräver asynkront gränssnitt...

Händelser i Datortsystemet	Händelser i skrivaren
Inväntar READY=1	Skrivaren är upptagen med att skriva ut ett tecken. READY=0
När READY=1 skrivs nästa tecken till skrivarens datargräns. Sätter DAV=1	Skrivaren är redo för nästa tecken och sätter READY=1
Inväntar READY=0	Ser att DAV=1. Läser nytt tecken från skrivarens datargräns. Sätter READY=0.
När READY=0 notifieras DAV som indikation på att det inte finns något tecken på skrivarens datargräns.	Skrivaren är upptagen med att skriva ut ett tecken. READY=0.

```

* Printer V3
PRINTER EQU 00800
PSTATUS EQU 00801
PCONTROL EQU 00802
EOT EQU 4

ORG $1000
LDX #TEXT
LOOP: LDA 1,X+
      CMPA #EOT
      BEQ STOP
      BRCLR PSTATUS, #4, Ready
      STAA PRINTER
      BSET PCONTROL, #2
INCRREADY: BRSET PSTATUS, #4, NotReady
           BRCLR PCONTROL, #2
      BRA LOOP
      BRA STOP
      NOP
      BRA STOP
ORG $2000
      FCS "Hej Du Kalle!"
      FCB EOT
    
```



Sammanfattning

- konstruera program för systemstart och med stöd för avbrottshantering från olika typer av kringenheter.

Exempel 4.43 Placering av Exceptionvektorer, assemblerkod

Följande programskelett illustrerar hur några avbrottsrutiner respektive avbrottsvektorer kan definieras i en fristående HCS12-applikation.

```

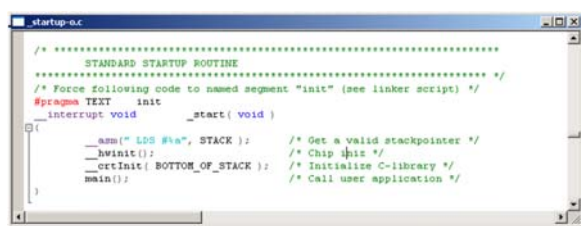
ORG $FFF2
FDB irq_service_routine
FDB xirq_service_routine
FDB software_interrupt_service_routine
FDB illegal_opcode_service_routine
FDB cop_service_routine
FDB clock_monitor_fail_service_routine
FDB Application_Start

; Symbolen "Application_Start_Address" kan vara godtycklig.
ORG Application_Start_Address
Application_Start:
LDS #TopOfStack
...
...
ANDCC #$FE ; nollställ I-flagga
JSR _main
    
```

Vår slutliga "appstart" blir nu:

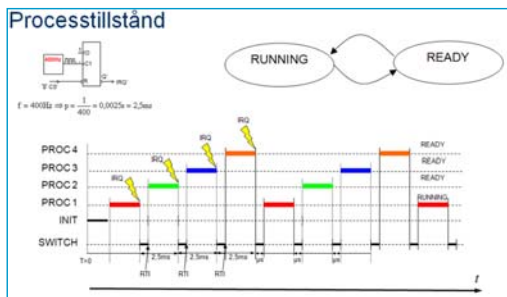
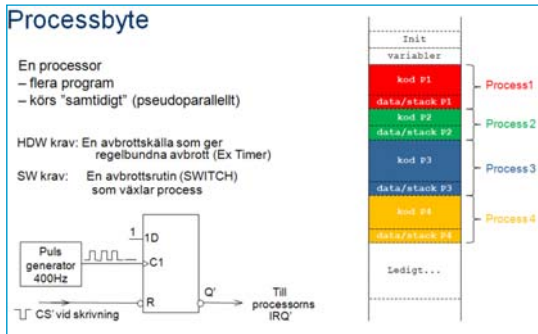
```

segment init
export _exit
import _main
function _start, __start_end
* Här börjar exekveringen...
_start
LDS #2FFF
JSR _main
_exit: NOP
BRA _exit
_start end
    
```



Sammanfattning

- kunna beskriva metoder och mekanismer som är centrala i systemprogramvara så som pseudoparallell exekvering och hantering av processer.



En realtidskärna

Jan Skansholm

```

In funktioner som ingår i realtidkärnan och som ett användningsprogram kan använda sig av deklarerats i filen process.h som måste inkluderas. Den ser ut på följande sätt:

#include PROCESS_H
#define PROCESS_H
#define DEFAULT_STACK_SIZE 128
#define MINIMUM_PRIORITY 1
#define DEFAULT_PRIORITY MINIMUM_PRIORITY
typedef struct process_attr_struct process; // doid definition i filen process.c
typedef struct semphive_struct semphive; // doid definition 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 *p);
extern process *running_process();
extern int get_process_id(process *p);
extern unsigned long int get_time(); // resultat ges i ms
extern void delay_process(process *p, unsigned long int t); // t ges i ms
extern int get_process_priority(process *p);
extern void set_process_priority(process *p, int prio);

void watch(int channel_no, unsigned long int interval) {
    while (1) {
        int i;
        wait(s); // begär exklusiv tillgång till AD-omvandlaren
        adc_read(channel_no);
        do {
            delay(50);
            i = adc_get_value();
        } while (i == BUSY);
        signal(s); // frisläpper AD-omvandlaren
        if (i == ERROR)
            warning(err_msg[channel_no]);
        else if (i <= 20 || i >= 40)
            warning(ill_msg[channel_no]);
        delay(interval);
    }
}

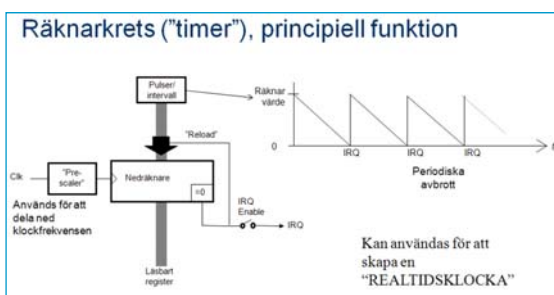
void f1(void) {
    watch(0, 2000);
}

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

Sammanfattning

13

- beskriva och använda kretsar för tidmätning.



.. Program för initiering..

```

; Adressdefinitioner
CRGINT EQU $38
RTICTL EQU $3B

timer_init:
; Initiera RTC avbrottsfrekvens
; Skriv tidbas för avbrottsintervall till RTICTL
MOVB #49, RTICTL
; Aktivera avbrott från CRG-modul
MOVB #80, CRGINT
RTS

Anmärkning: Det är olämpligt att använda detta värde då programmet testas i
simulator, använd då i stället det kortast tänkbara avbrottsintervallet enligt;

; Skriv tidbas för avbrottsintervall till RTICTL
MOVB #510, RTICTL ; För simulator
    
```

Realtidsklocka i HCS12

Address Offset	Use	Access
\$_00	CRG Synthesizer Register (SYNR)	RW
\$_01	CRG Reference Divider Register (REFDV)	RW
\$_02	CRG Test Flags Register (CTFLG) ¹	RW
\$_03	CRG Flags Register (CRGFLG)	RW
\$_04	CRG Interrupt Enable Register (CRGINT)	RW
\$_05	CRG Clock Select Register (CLKSEL)	RW
\$_06	CRG PLL Control Register (PLLCTL)	RW
\$_07	CRG RTI Control Register (RTICTL)	RW
\$_08	CRG COP Control Register (COPCTL)	RW
\$_09	CRG Force and Bypass Test Register (FORBYFP) ²	RW
\$_0A	CRG Test Control Register (CTCTL) ³	RW
\$_0B	CRG COP Arm/Timer Reset (ARMCOP)	RW

NOTES:
1. CTFLG is intended for factory test purposes only.
2. FORBYFP is intended for factory test purposes only.
3. CTCTL is intended for factory test purposes only.

Tre olika register används för realtidsklockan

Realtidsklocka i HCS12, avbrotts hantering

```

; Adressdefinition
CRGFLG EQU $37

timer_interrupt:
; Kvittera avbrott från RTC
BSET CRGFLG, #80
RII

; Avbrottsvektor på plats..
ORG $FFF0
FDB timer_interrupt
    
```

Adress (hex)	Funktion
FFFF	Real Time Interrupt
FFFE	Enhanced Capture Timer channel 1
FFFC	Enhanced Capture Timer channel 1
FFEA	Enhanced Capture Timer channel 2
FFFE	ForP Interrupt
FFFC	PWM Emergency Shutdown
FFFA	Reserverade
FFF0	

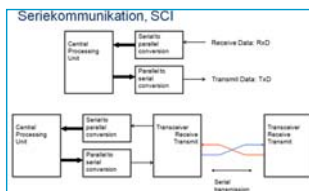
Sammanfattning

14

- beskriva och använda kretsar för parallell respektive seriell överföring.

Multiplexed External Bus Interface (MEBI)										
Offset	7	6	5	4	3	2	1	0	Mnemonic	
\$00	R W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	PORTA
\$01	R W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	PORTB
\$02	R W	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	DDRA
\$03	R W	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	1=OUT 0=IN	DDRB
\$04	R W								

Figur 4.5: Register för Port A/B som generell IO



Bestämna Baudrate-värde

$$BR = \frac{PLLCLK}{16 \times \text{baudrate}}$$

$$\text{baudrate} = \frac{PLLCLK}{16 \times BR}$$

9 600	48 · 10 ⁶ / 16 · 9600 = 312,5	48 · 10 ⁶ / 16 · 312 = 9615	48 · 10 ⁶ / 16 · 312 = 9585
57 600	48 · 10 ⁶ / 16 · 57600 = 52,08333	48 · 10 ⁶ / 16 · 52 = 57002	16 · 52 = 832
256 000	48 · 10 ⁶ / 16 · 256000 = 11,71875	48 · 10 ⁶ / 16 · 112 = 250000	16 · 112 = 1792

Baslock: EQ0 #000000 / 8 Mhz
 ; Baudrate register värden, baserat på PLL-klocka
 Baud9600: EQ0 #000000 (EnLock/(16*9600))

Exempel 4.50

Angi i såväl assemblerpråk som C, programkonstruktioner som initierar port A för användning som inport samt port B för användning som utport.

Lösning:

```

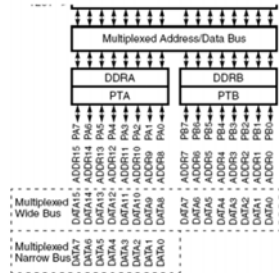
PORTA EQU 0
PORTB EQU 1
DDRA EQU 2
DDRB EQU 3
...
CLR DDRA
MOVB #$FF, DDRB
...
    
```

```

typedef struct sMEBI{
volatile unsigned char porta;
volatile unsigned char portb;
volatile unsigned char ddra;
volatile unsigned char ddrb;
}MEBI, *PMEBI;
    
```

```

#define MEBI_BASE 0
(( ( PMEBI ) ( MEBI_BASE ))-> ddra) = 0;
(( ( PMEBI ) ( MEBI_BASE ))-> ddrb) = 0xFF;
    
```



Initiering, "busy-wait"

Algorithm:

1. Initiera BAUDRATE
2. Aktivera Transmitter Receiver

Basadress = 5C8

Offset	7	6	5	4	3	2	1	0	Mnemonic	
\$00	R	BAUDR2	BAUDR1	BAUDR0	BAUDR7	BAUDR6	BAUDR5	BAUDR4	BAUDR3	Baud Rate Register 2
\$01	R	BAUDR2	BAUDR1	BAUDR0	BAUDR7	BAUDR6	BAUDR5	BAUDR4	BAUDR3	Baud Rate Register 1
\$02	R	SCIOCR2	SCIOCR1	SCIOCR0	SCIOCR7	SCIOCR6	SCIOCR5	SCIOCR4	SCIOCR3	Serial Control Register 2
\$03	R	SCIOCR2	SCIOCR1	SCIOCR0	SCIOCR7	SCIOCR6	SCIOCR5	SCIOCR4	SCIOCR3	Serial Control Register 1
\$04	R	SCIOCR2	SCIOCR1	SCIOCR0	SCIOCR7	SCIOCR6	SCIOCR5	SCIOCR4	SCIOCR3	Serial Control Register 2
\$05	R	SCIOCR2	SCIOCR1	SCIOCR0	SCIOCR7	SCIOCR6	SCIOCR5	SCIOCR4	SCIOCR3	Serial Control Register 1
\$06	R	SCIOCR2	SCIOCR1	SCIOCR0	SCIOCR7	SCIOCR6	SCIOCR5	SCIOCR4	SCIOCR3	Serial Control Register 2
\$07	R	SCIOCR2	SCIOCR1	SCIOCR0	SCIOCR7	SCIOCR6	SCIOCR5	SCIOCR4	SCIOCR3	Serial Control Register 1

```

SCIOCR0: EQ0 5C8 ; SCI 0 baudrate-register (16 bit).
SCIOCR2: EQ0 5CB ; SCI 0 styr-register 2.
; bitdefinitioner, styrregister
TE: EQ0 508 ; Transmitter enable.
RE: EQ0 504 ; Receiver enable.
    
```

Programmet...

```

; enkelt testprogram
ORG $1000
JSR serial_init
Loop: JSR in ; "eka" tecken
      JSR out
      BRA loop

; OUT tecken rutin
; Skriv tecken till SCIO
; Inparameter, register B: tecken.
out: BRCLR SCIOCR1, #TDRE, out ; vänta till TDRE=1
     STAB SCIODRL ; skicka tecken ...
     RTS

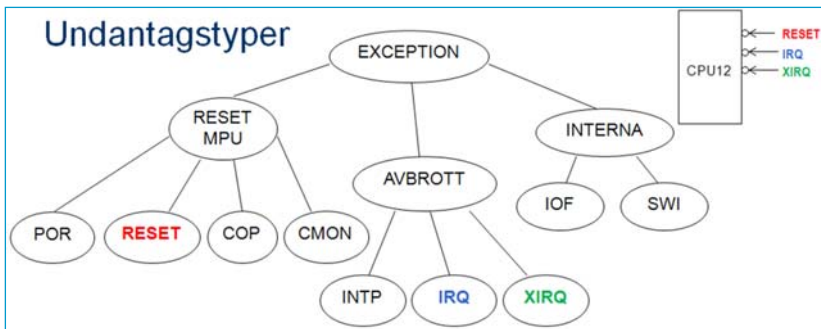
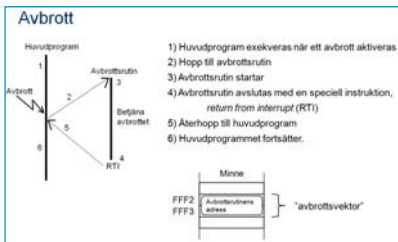
; IN tecken rutin
; Läs tecken från SCIO
; Returnera i register B
in: BRCLR SCIOCR1, #RDRF, in ; vänta till RDRF=1
    LDAB SCIODRL ; läs tecken
    RTS
    
```

4. Undantagshantering i datorsystem

Att självständigt kunna:

- beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart.
- konstruera enklare avbrottsystem med användning av digitala komponenter.
- beskriva och tillämpa olika metoder för prioritetshantering vid multipla avbrottskällor (mjukvarubaserad och hårdvarubaserad prioritering, avbrottsmaskering, icke-maskerbara avbrott).

- beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart.



Fatala fel, kräver RESET av CPU

POR, Power On Reset, vid opålitlig tillgång
RESET, utsignal till processorn aktiveras.
COP, Computer Operating Properly, så kallad watchdog-funktion
CMON, Clock Monitor Reset, övervakare för klockan, om frekvensen sjunker under 10 kHz genererar RESET.

Adress (hex)	Funktion
FFFF	RESET Startvektor
FFFC	Clock Monitor Fail
FFFB	COP Watchdog Timeout
FFFA	Illegal Op Code
FFF9	Illegal Op Code
FFF8	Illegal Op Code
FFF7	Illegal Op Code
FFF6	Illegal Op Code
FFF5	Illegal Op Code
FFF4	Illegal Op Code
FFF3	Illegal Op Code
FFF2	Illegal Op Code
FFF1	Illegal Op Code
FFF0	Illegal Op Code

Internt genererade avbrott

Adress (hex)	Funktion
FFFF	RESET Startvektor
FFFC	Clock Monitor Fail
FFFB	COP Watchdog Timeout
FFFA	Illegal Op Code
FFF9	Illegal Op Code
FFF8	Illegal Op Code
FFF7	Illegal Op Code
FFF6	Illegal Op Code
FFF5	Illegal Op Code
FFF4	Illegal Op Code
FFF3	Illegal Op Code
FFF2	Illegal Op Code
FFF1	Illegal Op Code
FFF0	Illegal Op Code

Interna undantag

Om processorn avkodar en otillåten operationskod kallas detta *Illegal Opcode Fetch* (IOF).
Processorn avbryter då, sparar registerinnehåll på stacken, läser autovektorn för IOF och utför undantagshantering.
Instruktionens Software Interrupt (SWI) fungerar på samma sätt, men har en annan autovektor och en bestämd operationskod.

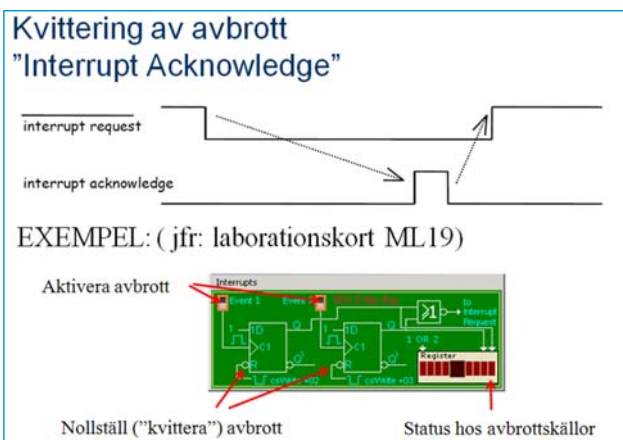
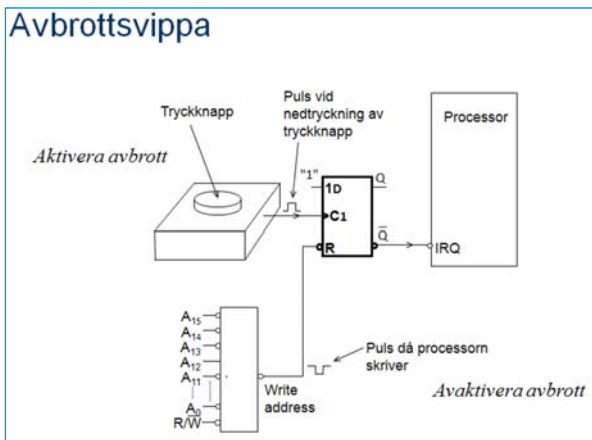
Adress (hex)	Funktion
FFFF	RESET Startvektor
FFFC	Clock Monitor Fail
FFFA	COP Watchdog Timeout
FFF9	Illegal Op Code
FFF8	Illegal Op Code
FFF7	Illegal Op Code
FFF6	Illegal Op Code
FFF5	Illegal Op Code
FFF4	Illegal Op Code
FFF3	Illegal Op Code
FFF2	Illegal Op Code
FFF1	Illegal Op Code
FFF0	Illegal Op Code

Externt genererade avbrott

Adress (hex)	Funktion
FFFF	RESET Startvektor
FFFC	Clock Monitor Fail
FFFB	COP Watchdog Timeout
FFFA	Illegal Op Code
FFF9	Illegal Op Code
FFF8	Illegal Op Code
FFF7	Illegal Op Code
FFF6	Illegal Op Code
FFF5	Illegal Op Code
FFF4	Illegal Op Code
FFF3	Illegal Op Code
FFF2	Illegal Op Code
FFF1	Illegal Op Code
FFF0	Illegal Op Code

Sammanfattning

- konstruera enklare avbrottssystem med användning av digitala komponenter.



Sammanfattning

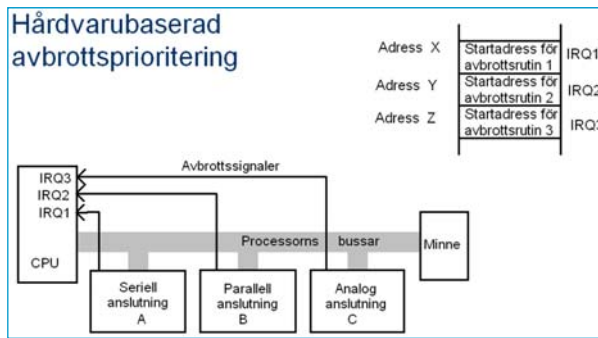
- beskriva och tillämpa olika metoder för prioritetshantering vid multipla avbrottskällor (mjukvarubaserad och hårdvarubaserad prioritering, avbrottsmaskering, icke-maskerbara avbrott).

Maskering av avbrott

Maskera avbrott:
SEI
Alternativt
ORCC #00010000

Demaskera avbrott:
CLI
Alternativt
ANDCC #011101111

Demaskera X-avbrott:
ANDCC #101111111
OBS: Kan INTE maskeras ("Non Maskable Interrupt")



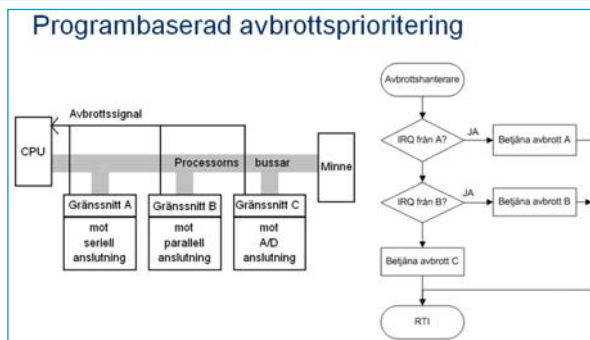
Intern avbrottsprioritering

Högre prioritet

För avbrott från interna kretsar bestäms prioriteten av avbrottsvektorns adress. Ju högre adress, desto högre prioritet.

Det finns vissa möjligheter att ändra detta programmatiskt.

Lägre prioritet



Av speciell vikt: "maskinorienterad programmering..."

2.24 En strömbrytare och en sju-sifferindikator (se figur) är anslutna till adresser 0x400 respektive 0x600 i ett MC12 mikrodatorsystem.

Konstruera en funktion `void DisplayNBBCD(void)` som hela tiden läser från strömbrytarna och skriver värden till sju-sifferindikatorn.

När bit 7 på inporten är ettställd skall sifferindikatorn släckas helt. När bit 7 på inporten är nollställd skall sifferindikatorn tändas enligt följande beskrivning:

- Bit 3-0 på inporten anger vad som skall visas på sifferindikatorn.
 - Om indata är i intervallet [0,9] skall motsvarande decimala siffra visas på sifferindikatorn.
 - Om indata är i intervallet [A,F] skall ett 'E' (Error) visas på sifferindikatorn.
- Bitarna 6-4 på inporten kan anta vilka värden som helst.

Du har tillgång till en tabell i minnet med segmentkoder för de hexadecimala siffrorna [0..F] (mönster för sifferindikatorn) enligt

```
unsigned char SegCodes[]={ 0x77,0x22,0x5B,0x6B,0x2E,0x6D,0x7D,0x23,
    0x7F,0x6F,0x3F,0x7C,0x55,0x7A,0x5D,0x18 };
```

Segmentkoden för bokstaven 'E' ges av:

```
#define ERROR_CODE 0x5D
```

Läsa/skriva på fasta adresser (portar)

Datatyper, storlek (8,16 eller 32 bitar...)

Heltalstyper, med eller utan tecken, vad innebär typkonverteringarna?

Bitoperationer &, |, ^ (AND, OR, XOR)

Skiftoperationer <<, >> (vänster, höger)

Kodningskonventioner

Program som kräver källtexter både i 'C' och assemblyspråk...

2.31 Inledningen (parameterlistan och lokala variabler) för en funktion ser ut på följande sätt:

```
void function( char *b, char a )
{
    char *c, *d;
    .....
}
```

- Visa hur utrymme för lokala variabler reserveras i funktionen (*prolog*).
- Visa funktionens aktiveringspost, ange speciellt offset för parametrar och lokala variabler.

Kompilatorkonvention XCC12:

- Parametrar överförs till en funktion via stacken.
- Då parametrarna placeras på stacken bearbetas parameterlistan från höger till vänster.
- Utrymme för lokala variabler allokeras på stacken. Variablerna behandlas i den ordning de påträffas i koden.
- **Prolog** kallas den kod som reserverar utrymme för lokala variabler.
- **Epilog** kallas den kod som återställer (återlämnar) utrymme för lokala variabler.
- Den del av stacken som används för parametrar och lokala variabler kallas *aktiveringspost*.

2.31: a) LEAS -4, SP
b)

Parameter/ variabel	adressering
a	8, SP
b	6, SP
c	2, SP
d	0, SP

Pekare och dess användning...

```
/*
  strpbrk.c
  C-library function "strpbrk"
*/
#include <string.h>
char *strpbrk(char *s, char *breakat)
{
    char *sscan, *bscan;

    for (sscan = s; *sscan != '\0'; sscan++) {
        for (bscan = breakat; *bscan != '\0';)
            if (*sscan == *bscan++)
                return sscan;
    }
    return((char *) 0 );
}
```

```
/*
  memcpy.c
  C-library function "memcpy"
*/
#include <string.h>
void *memcpy(void *dst, void *src, size_t size)
{
    char *d, char *s, size_t n;
    if (size <= 0)
        return(dst);
    s = (char *) src;
    d = (char *) dst;
    if (s <= d && s + (size - 1) >= d) {
        /* Overlap, must copy right-to-left */
        s += size - 1;
        d += size - 1;
        for (n = size; n > 0; n--)
            *d-- = *s--;
    } else
        for (n = size; n > 0; n--)
            *d++ = *s++;
    return(dst);
}
```

Assemblerprogrammering...

```
# define DATA      *( char *) 0x700
# define STATUS     *( char *) 0x701
void printerprint( char *s )
{
    while( *s )
    {
        while( STATUS & 1 )
        {}
        DATA = *s;
        s++;
    }
}
```



```
; void printerprint( char *s )
_printerprint:
; {
;   while( *s )
;       LDX 2,SP
printerprint1:
;   TST ,X
;   BEQ printerprint2
;   {
;       while( !( STATUS & 1 ) )
;           {}
printerprint3:
;   LDAB $0701
;   ANDB #$01
;   BEQ printerprint3
;   DATA = *s;
;   LDAB 1,X+      (även 's++' nedan)
;   STAB $0700
;   S++;
;   BRA printerprint1
printerprint2:
;   }
; }
RTS
```