

# Programmering av inbyggda system 2014/2015

## Sammanfattning

### Kursens syften är

- att vara en introduktion till konstruktion av små inbyggda system och
- att ge en förståelse för hur imperativa styrstrukturer översätts till assembler
- att ge en förståelse för de svårigheter som uppstår vid programmering av händelsestyrda system med flera indatakällor.

Vi repeterar kursens "lärandemål"

# 1. Programutveckling i C och assemblyspråk

Kunna utföra programmering i C och assemblyspråk 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  
returvärden.*

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

```
int a;
static int b;
```

*Lagrings-  
klass och  
synlighet.*

```
; 1 | int a;
SEGMENT bss
_a: RMB $2
EXPORT _a [r,2]
```

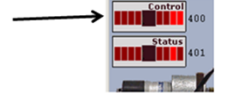
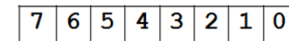
```
; 2 | static int b;
1: RMB $2
```

(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
* borrhmaskinens styrord på adress
* DCCopy. Ettställer 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     OUTONE
* Utdata:         Inga
* Registerpåverkan: Ingen
* 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	Återhoppadress, 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
```

## Returvärden via register

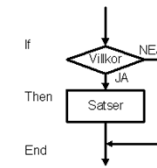
Register väljs, beroende på returvärdets typ (storlek), HCS12-exempel

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.*

If (...) {...}



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

Bättre kodning...

```
DipSwitch EQU $600
HexDisp EQU $400

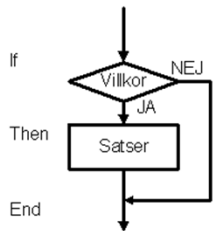
...
TST DipSwitch
BEQ end

LDAB DipSwitch
STAB HexDisp

end:
```

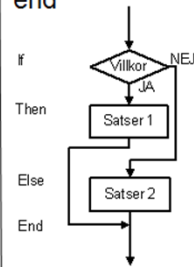
## Kontrollstrukturer

If( Villkor ) then ...



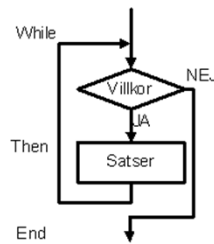
```
if (Villkor)
{
    Satsen;
}
```

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



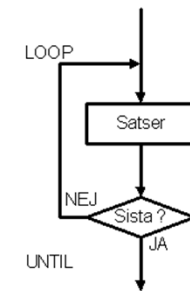
```
if (Villkor)
{
    Satsen1;
} else {
    Satsen2;
}
```

while( Villkor )  
loop



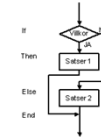
```
while (Villkor)
{
    Satsen;
}
```

loop  
...  
until( Villkor )



```
do {
    Satsen;
}
while (Villkor);
```

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



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

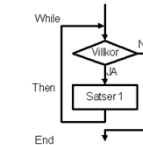
<b>BNE</b>	"Hopp" om ICKE zero	Z=0
<b>BEQ</b>	"Hopp" om zero	Z=1

```
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:
```

<b>BEQ</b>	"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
```

Test av tal utan tecken		
<b>BHI</b>	Villkor: R>M	C + Z = 0
Test av tal med tecken		
<b>BGT</b>	Villkor: R>M	Z + (N @ V) = 0

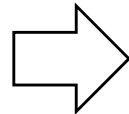
- *beskriva och använda sammansatta datatyper (fält och poster) och enkla datatyper (naturliga tal, heltal och flyttal).*

```

/*
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]

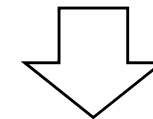
;      4 | int      intvec[10];
;      _intvec:   RMB    $14
;      EXPORT    _intvec  [r,20]

;      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".

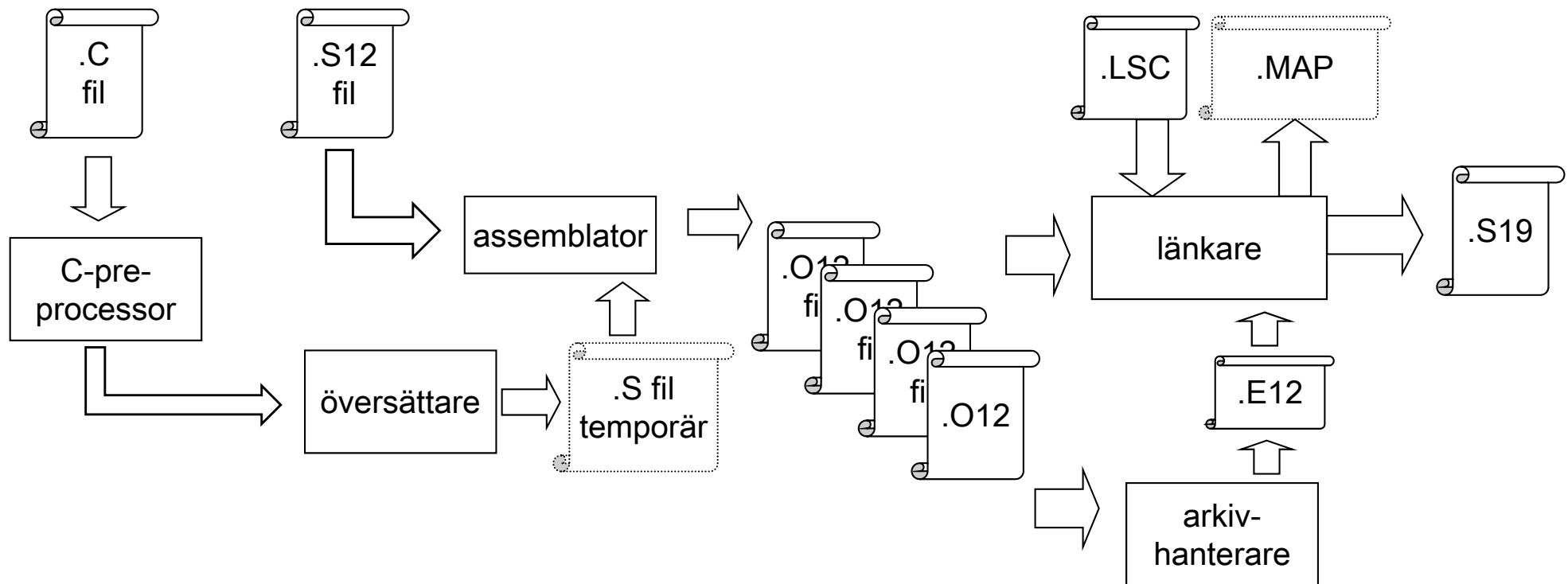
## 2. Programutvecklingsteknik

Att 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.*

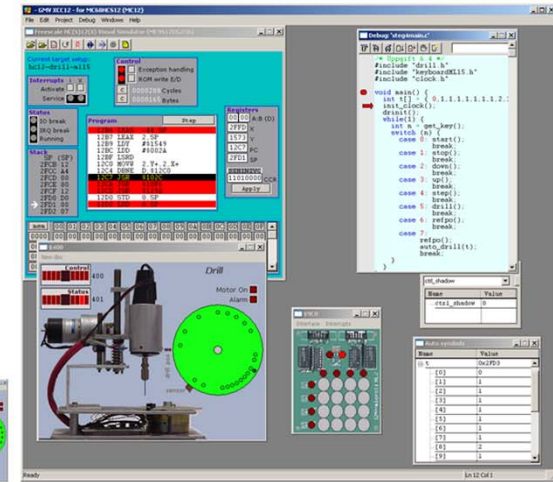
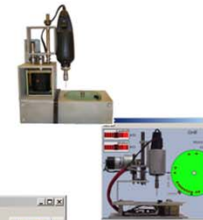




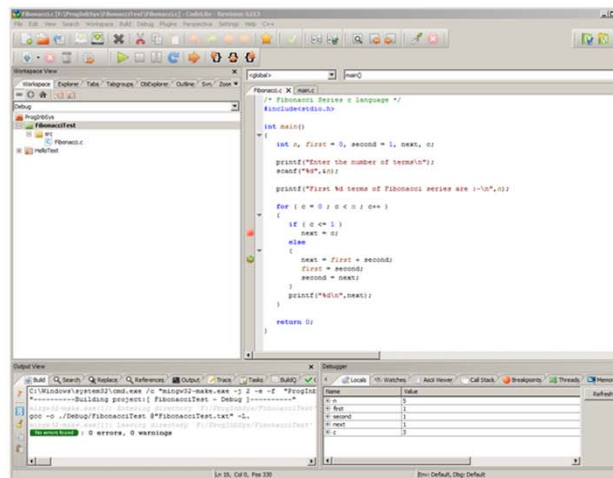
- konstruera, redigera och översätta (kompilera och assemblera) program
- testa, felsöka och rätta programkod med hjälp av avsedda verktyg.

Moment 5:

XCC12  
för Simulator  
och laborations-  
system



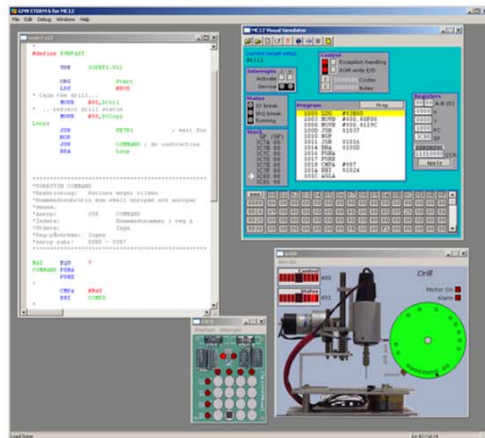
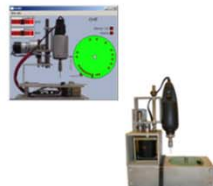
CodeLite för  
moment 3  
"Morsealfabetet"  
och moment 4  
"Prioritetskö".



Dessa lärandemål har vi kontrollerat under laborationer.

Moment 1 och 2:

ETERM för  
Simulator och  
laborationssystem

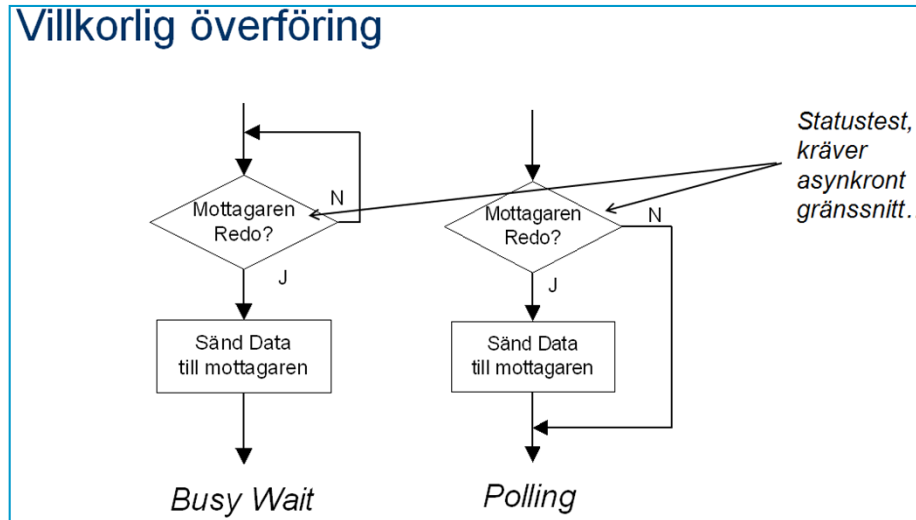
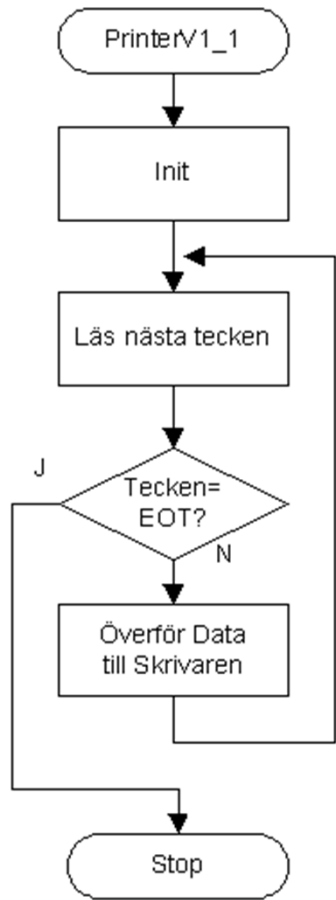


## 3. Systemprogrammerarens bild

Att 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.
- beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart samt prioritetshantering vid undantag.
- 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, statustest och rundfrågning.*



Händelser i Datorsystemet	Händelser i skrivaren
InväntarREADY=1	Skrivaren är upptagen med att skriva ut ett tecken.READY=0.
	Skrivaren är redo för nästa tecken och sätter READY=1
När READY=1 skrivs nästa tecken till skrivarens dataregister. Sätter DAV=1	InväntarDAV=1
InväntarREADY=0	Ser att DAV=1. Läser nytt tecken från skrivarbussen. Signalerar upptagen, READY=0.
När READY=0 nollställs DAV som indikation på att det inte finns giltigt tecken på skrivarbussen	Skrivaren är upptagen med att skriva ut ett tecken.READY=0.

```

* Printer V3
PRINTER EQU $0800
PSTATUS EQU $0801
PCONTROL EQU $0802
EOT EQU 4

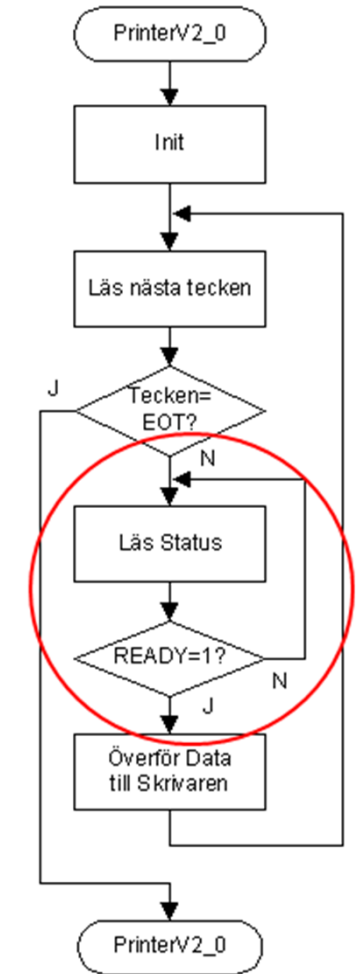
ORG $1000
LDX #Text
Loop: LDAA 1,X+
      CMPA #EOT
      BEQ Stop

Ready: BRCLR PSTATUS,#4,Ready
      STAA PRINTER
      BSET PCONTROL,#2

NotReady: BRSET PSTATUS,#4,NotReady
          BCLR PCONTROL,#2
          BRA Loop

Stop: NOP
      BRA Stop

Text: ORG $3000
      FCS "Hej Du Kalle!"
      FCB EOT
          
```



- *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

...

...

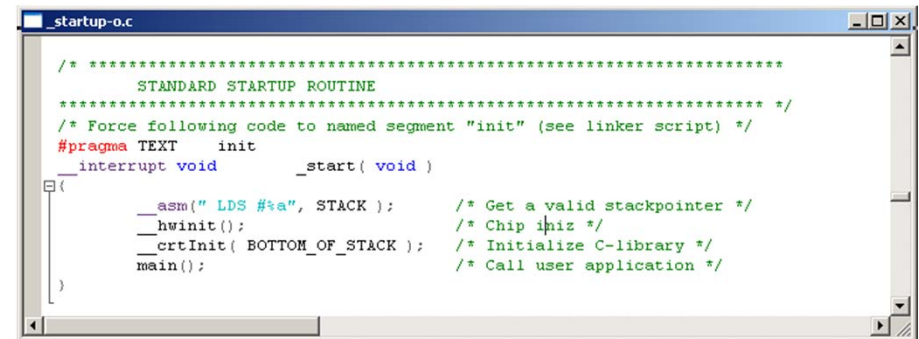
ANDCC   #$FE      ; nollställ I-flagga

JSR     _main
    
```

Vår slutliga "appstart" blir nu:

```

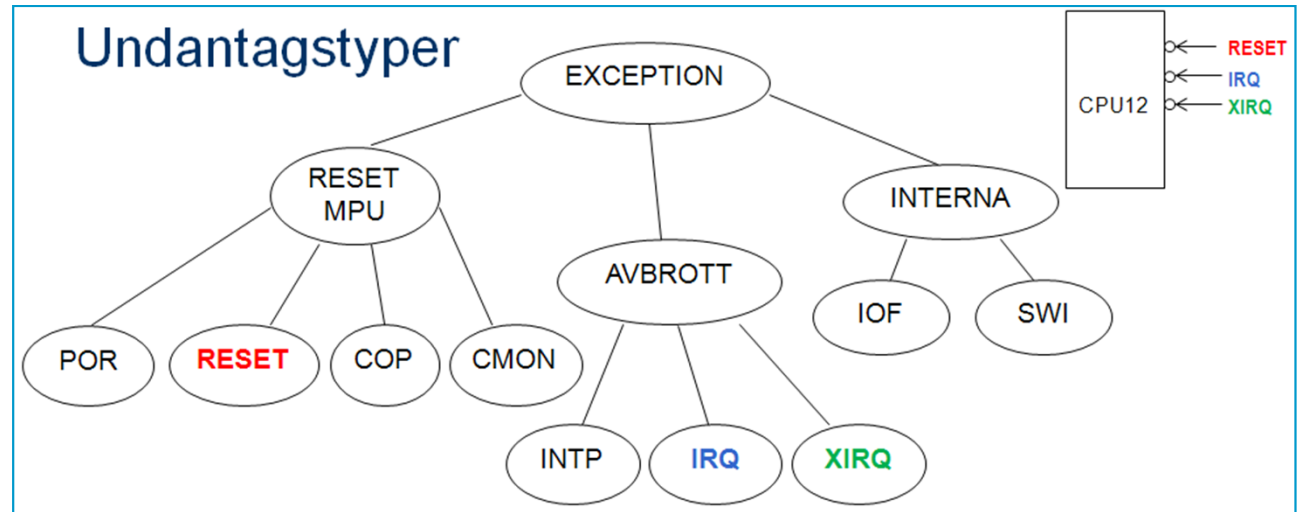
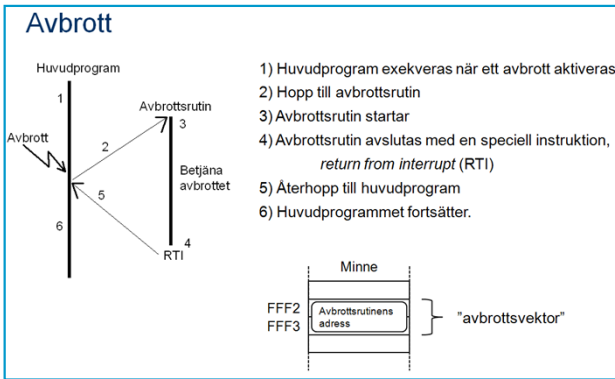
segment    init
export    _exit
import    _main
function  __start, __start_end
*   Här börjar exekveringen...
__start
        LDS      #$2FFF
        JSR      _main
_exit:  NOP
        BRA      _exit
__start_end
    
```



```

_startup-o.c
/* *****
   STANDARD STARTUP ROUTINE
   ***** */
/* Force following code to named segment "init" (see linker script) */
#pragma TEXT init
__interrupt void __start( void )
{
    __asm(" LDS #a", STACK ); /* Get a valid stackpointer */
    __hwin(); /* Chip initialization */
    __crtInit( BOTTOM_OF_STACK ); /* Initialize C-library */
    main(); /* Call user application */
}
    
```

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



### Fatala fel, kräver RESET av CPU

POR, *Power On Reset*, vid spänningstillslag  
 RESET, insignal till processorn aktiveras.  
 COP, *Computer Operating Properly*, så kallad *watchdog-funktion*.  
 CMON, *Clock Monitor Reset*, övervakar E-klockan, om frekvensen sjunker under 10 kHz genererar RESET.

Adress (hex)	Funktion
FFF8	RESET, Startvektor
FFFC	Clock Monitor Fail
FFFA	COP Watchdog Timeout
FFF8	Illegal Op Code
FFF6	SWI
FFF4	XIRQ
FFF2	IRQ
FF00-FFF0	Enhetsspecifika vektorer, skiljer sig något beroende på olika varianter

### Internt genererade avbrott

Adress (hex)	Funktion
FFF0	Real Time Interrupt
FFEE	Enhanced Capture Timer channel
FFEC	Enhanced Capture Timer channel 1
FFEA	Enhanced Capture Timer channel 2
FF8E	Port P Interrupt
FF8C	PWM Emergency Shutdown
FF8A-FF80	Reserverade

### 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 undantagshandling.

Instruktionen *SoftWare Interrupt (SWI)* fungerar på samma sätt, men har en annan autovektor och en bestämd operationskod.

Adress (hex)	Funktion
FFF8	RESET, Startvektor
FFFC	Clock Monitor Fail
FFFA	COP Watchdog Timeout
FFF8	Illegal Op Code
FFF6	SWI
FFF4	XIRQ
FFF2	IRQ
FF00-FFF0	Enhetsspecifika vektorer, skiljer sig något beroende på olika varianter

### Externt genererade avbrott

Adress (hex)	Funktion
FFF8	RESET, Startvektor
FFFC	Clock Monitor Fail
FFFA	COP Watchdog Timeout
FFF8	Illegal Op Code
FFF6	SWI
FFF4	XIRQ
FFF2	IRQ
FF00-FFF0	Enhetsspecifika vektorer, skiljer sig något beroende på olika varianter



- 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 # %11101111

Demaskera X-avbrott:  
ANDCC # %10111111  
OBS: Kan INTE maskeras ("Non Maskable Interrupt")

**CONDITION CODE REGISTER**

S	CARRY
X	OVERFLOW
H	ZERO
I	NEGATIVE
N	MASK (DISABLE) IRQ INTERRUPTS
Z	HALF-CARRY (USED IN BCD ARITHMETIC)
V	MASK (DISABLE) XIRQ INTERRUPTS RESET OR XIRQ SET X INSTRUCTIONS MAY CLEAR X BUT CANNOT SET X
C	STOP DISABLE (IGNORE STOP OPCODES) RESET DEFAULT IS 1

### Hårdvarubaserad avbrottsprioritering

Adress X	Startadress för avbrottsrutin 1	IRQ1
Adress Y	Startadress för avbrottsrutin 2	IRQ2
Adress Z	Startadress för avbrottsrutin 3	IRQ3

### Intern avbrottsprioritering

För avbrott från interna kretsar bestäms prioriteten av avbrottsvektorns adress. Ju högre adress, desto högre prioritet. Det finns vissa möjligheter att ändra detta programmsätt.

Adress (hex)	Funktion
FFF0	Real Time Interrupt
FFEE	Enhanced Capture Timer channel
FFEC	Enhanced Capture Timer channel 1
FFEA	Enhanced Capture Timer channel 2
FFEB	Enhanced Capture Timer channel 3
FFED	Enhanced Capture Timer channel 4
FFE4	Enhanced Capture Timer channel 5
FFE2	Enhanced Capture Timer channel 6
FFE0	Enhanced Capture Timer channel 7
FFDE	Enhanced Capture Timer overflow
FFDC	Pulse accumulator A overflow
FFDA	Pulse accumulator input edge
FFD8	SPI0
FFD6	SCID
FFD4	SC1I
FFD2	ATD0
FFD0	ATD1
FFD6	Port A
FFC0	Port H
FFCA	Modulus Down Counter overflow
FFC6	Pulse Accumulator B overflow
FFC6	PLL lock
FFC4	CRG Self Clock Mode
FFC2	Advanced (B)LCU
FFC0	IIC Bus
FFBE	SPI1
FFBC	Reserverad
FFBA	EEPROM 1-Bit
FFB8	PLASH 1-Bit
FFB6	CAN0 wake-up
FFB4	CAN0 errors
FFB2	CAN0 receive
FFB0	CAN0 transmit
FFA6	CAN4 wake-up
FFA4	CAN4 errors
FFA2	CAN4 receive
FFA0	CAN4 transmit
FF9E	Port H interrupt
FF9C	PVIM Emergency Shutdown
FF9A	Reserverade
FF90	

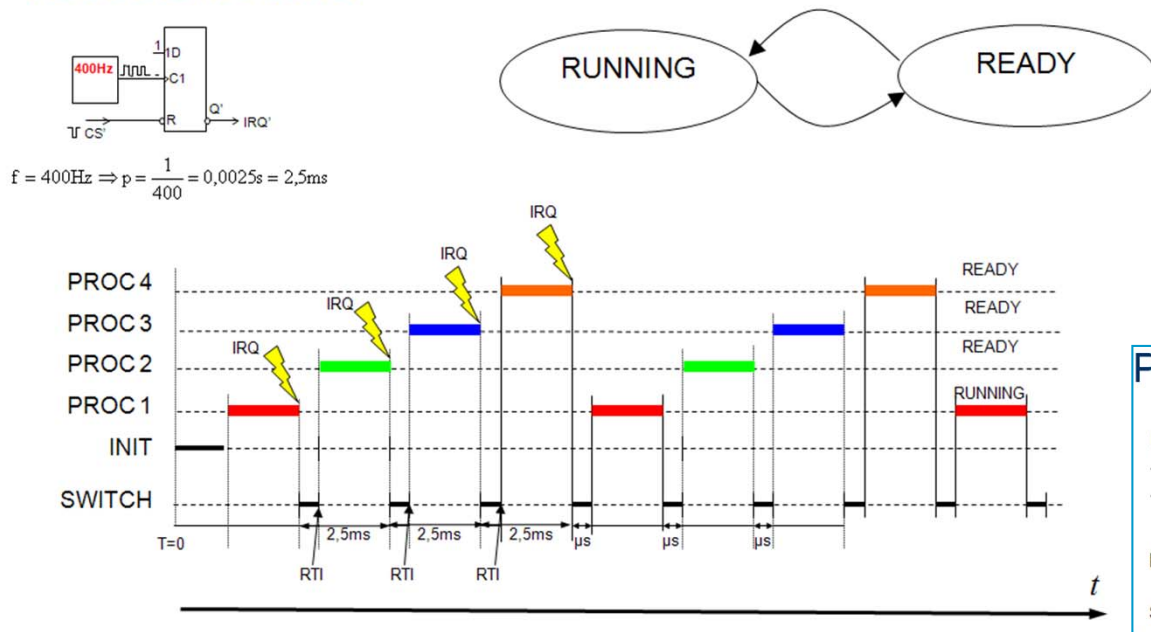
### Programbaserad avbrottsprioritering

```

    graph TD
      CPU[CPU] -- Avbrottsignal --> Handler[Avbrottskrets]
      Handler --> CheckA{IRQ från A?}
      CheckA -- JA --> ServA[Bejåna avbrott A]
      CheckA -- NEJ --> CheckB{IRQ från B?}
      CheckB -- JA --> ServB[Bejåna avbrott B]
      CheckB -- NEJ --> ServC[Bejåna avbrott C]
      ServC --> RTI[RTI]
  
```

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

## Processtillstånd

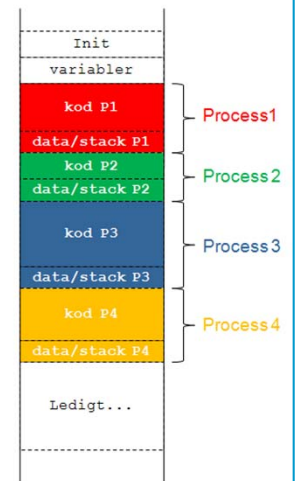
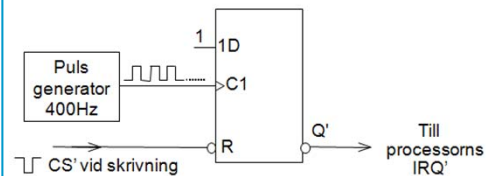


## Processbyte

En processor  
 – flera program  
 – körs "samtidigt" (pseudoparallellt)

HDW krav: En avbrottskälla som ger regelbundna avbrott (Ex Timer)

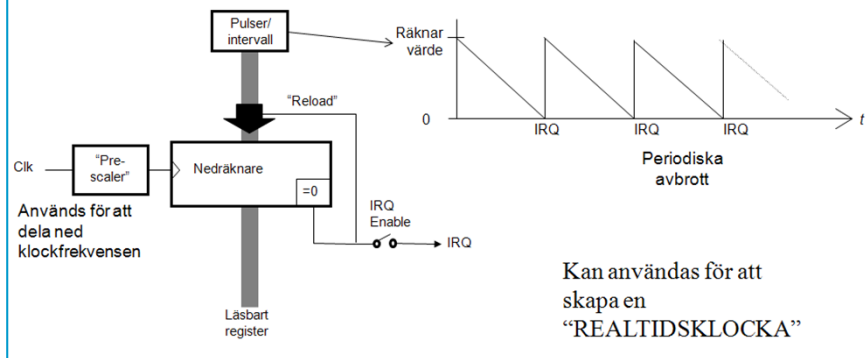
SW krav: En avbrottsrutin (SWITCH) som växlar process





- *beskriva och använda kretsar för tidmätning.*

## Räknarkrets ("timer"), principiell funktion



## Realtidsklocka i HCS12

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

Tre olika register används för realtidsklockan

### NOTES:

1. CTFLG is intended for factory test purposes only.
2. FORBYP is intended for factory test purposes only.
3. CTCTL is intended for factory test purposes only.

## .. Program för initiering..

```

; Adressdefinitioner
CRGINT EQU    $38
RTICTL EQU    $3B

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

; Skriv tidbas för avbrottsintervall till RTICTL
    MOVW    #$10,RTICTL    ; För simulator
    
```

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

## Realtidsklocka i HCS12, avbrottshantering

```

; Adressdefinition
CRGFLG EQU    $37

timer_interrupt:
; Kvittera avbrott från RTC
    BSET    CRGFLG,$80
    RTI

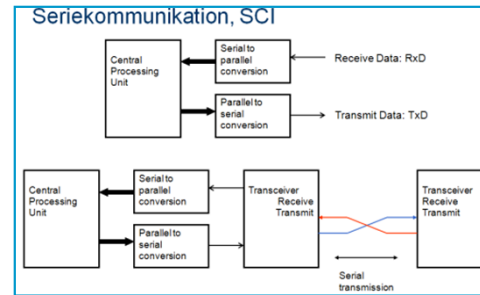
; Avbrottsvektor på plats..
    ORG    $FFF0
    FDB    timer_interrupt
    
```

Adress (hex)	Funktion
FF00	Real Time Interrupt
FF04	Enhanced Capture Timer channel
FF08	Enhanced Capture Timer channel 1
FF0C	Enhanced Capture Timer channel 2
....	....
FF8E	PortP Interrupt
FF8C	PWM Emergency Shutdown
FF8A-FF80	Reserverade

- 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

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

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

9 600	$\frac{48 \times 10^6}{16 \times 9600} = 312,5$	$\frac{48 \times 10^6}{16 \times 312} = 9615$	$\frac{48 \times 10^6}{16 \times 313} = 9585$
57 600	$\frac{48 \times 10^6}{16 \times 57600} \approx 52,08333$	$\frac{48 \times 10^6}{16 \times 52} = 57692$	
256 000	$\frac{48 \times 10^6}{16 \times 256000} = 11,71875$	$\frac{48 \times 10^6}{16 \times 12} = 250000$	

**Eclock:** EQU 8000000 ; 8 MHz  
 ; BaudRate register värden, baserad på PLL-klocka  
**Baud9600:** EQU (Eclock/(16\*9600))

### Exempel 4.50

Ange i såväl assemblerspråk 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

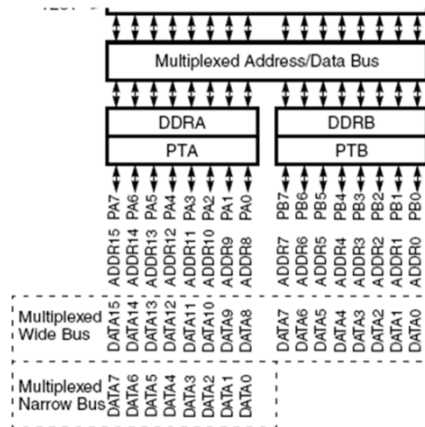
...
CLR DDRA
MOVB #$FF, DDRB
...
  
```

```

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"

Basadress = SC8

Algorithm:

1. Initiera BAUDRATE
2. Aktivera Transmitter Receiver

Offset	7	6	5	4	3	2	1	0	Mnemonic	Namn			
\$00	R	0	0	0	0	0	0	0	SCIBDH	Baud Rate Register High			
\$01	R	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SCIBDL	Baud Rate Register Low			
\$02	R	LOOPS	SCI0WAL	RSRC	M	WAKE	ILIE	FE	PF	SCICR1	Control Register 1		
\$03	R	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SRK	SCICR2	Control Register 2		
\$04	R	TDRE	TC	RDRF	IDLE	OR	NP	FE	PF	SCISR1	Status Register 1		
\$05	R	0	0	0	0	0	0	0	BRR13	TXDIR	RAF	SCISR2	Status Register 2
\$06	R	RB	TE	0	0	0	0	0	0	0	0	SCIDRH	Data Register High
\$07	R	R7	RE	R5	R4	R3	R2	R1	R0	0	0	SCIDL	Data Register Low
	W	R7	TE	R5	R4	R3	R2	R1	R0	0	0	0	0

```

SCI0BD: EQU $C8 ; SCI 0 baudrate-register (16 bit).
SCI0CR2: EQU $CB ; SCI 0 styr-register 2.
; Bitdefinitioner, styrregister
TE: EQU $08 ; Transmitter enable.
RE: EQU $04 ; 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 SCI0
; Inparameter, register B: tecken.
out: BRCLR SCI0SR1, #TDRE, out ; vänta till TDRE=1
      STAB SCI0DRL ; skicka tecken ...
      RTS
  
```

```

; IN tecken rutin
; Läs tecken från SCI0
; Returnera i register B
in: BRCLR SCI0SR1, #RDRF, in ; vänta till RDRF=1
     LDAB SCI0DRL ; läs tecken
     RTS
  
```

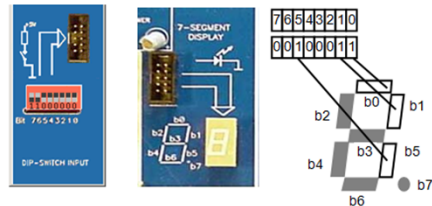
# Av speciell vikt: "maskinorienterad programmering..."

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 DisplayNBCD( 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 assemblerspråk...

### 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 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 offseter för parametrar och lokala variabler.

```
/*
    Men även tilldelningar och
    enklare typkonverteringar!
*/
.....
{
    char a;
    int b,c;
    a = b;
    c = a;
    ...
}
```

2.31: a) LEAS -4,SP  
b)

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

## Pekare och deras användning

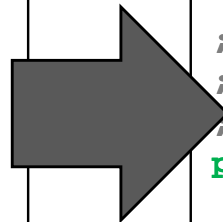
```
/*
  strpbrk.c
  C-library function "strpbrk"
*/
#include <string.h>
char *strpbrk(char *s, char *breakat)
{
  char *sscan, *bscan;

  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
```

# Maskinorienterad programmering 2014/2015

*Sammanfattad...*

Tisdag 13/1

Tentamen, 14.00-18.00 (V-salar)

Tillåtna hjälpmedel:

**Instruktionslista för CPU12**

***Quick guide: C-reference card ANSI***

En av böckerna:

**Vägen till C**

**The C programming language**