Maskinorienterad Programmering 2010/2011

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 returvärden.

Subrutiner för att manipulera styrregistret OUTONE och OUTZERO

```
* Subrutin OUTONE. Läser kopian av
* hosrmaskinens 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 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Lagrings-klass och synlighet.

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

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

```
;       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.
```

```
**************************************
* 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.
**************************************
```


- *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);
```

översättas 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);
```

Översättas 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	
1c.msb		6,SP
1b.lsb	Parameter 1b	
1b.msb		4,SP
1a.lsb	Parameter 1a	
1a.msb		2,SP
PC.lsb	Återhoppsadress, placeras här vid BSR	
PC.msb		0,SP

```
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        ; återhoppsadress->X
    INX                ; modifiera ...
    STX    0,SP        ; .. tillbaks till stack
    . . .
    . . .
    . . .
    RTS
```

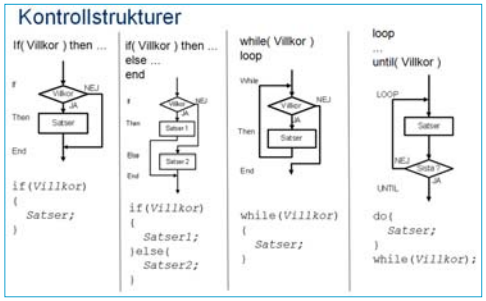
Returvärden via register

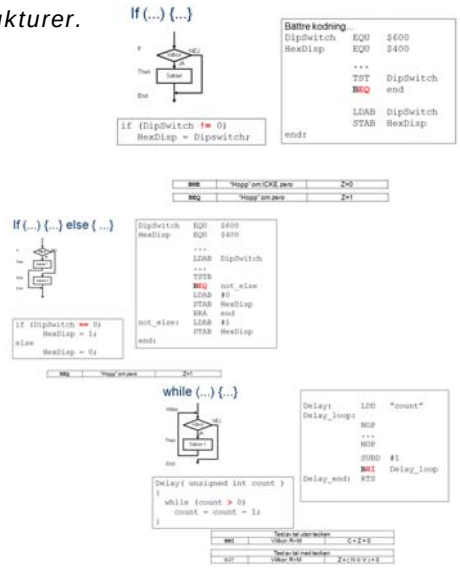
Register väljs, beroende på returvärdets typ (storlek), HCS12-exempel

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

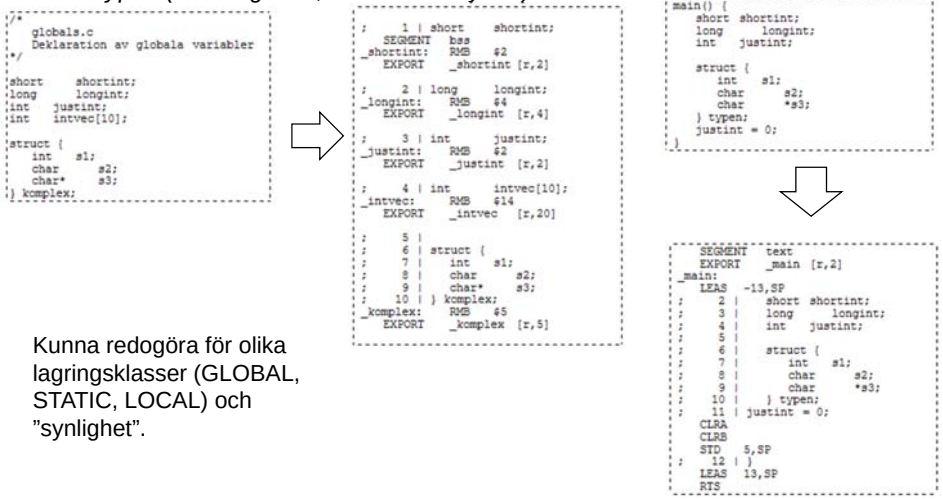
En regel (konvention) bestäms och följs därefter vid kodning av samtliga subrutiner

- *beskriva och använda olika kontrollstrukturer.*





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



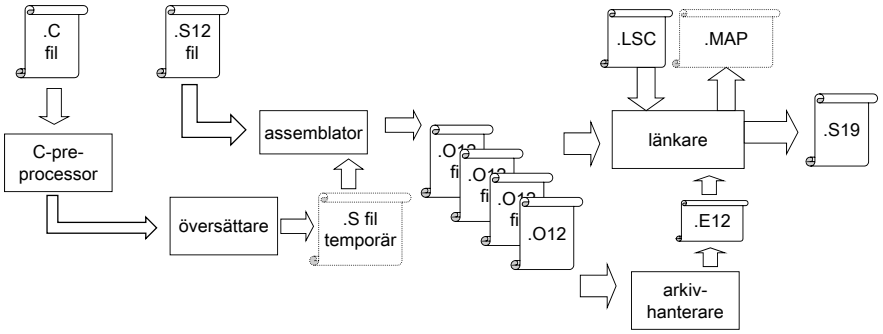
Kunna redogöra för olika lagringsklasser (GLOBAL, STATIC, LOCAL) och ”synlighet”.

2. Programutvecklingteknik

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.*

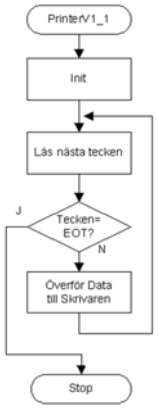
Dessa lärandemål har vi kontrollerat under laborationer.

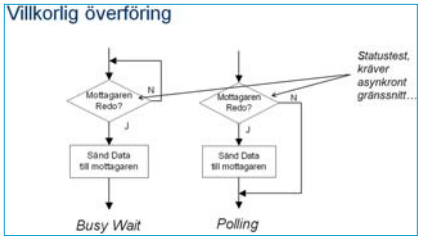
3. Systemprogrammerarens bild av inbäddade system

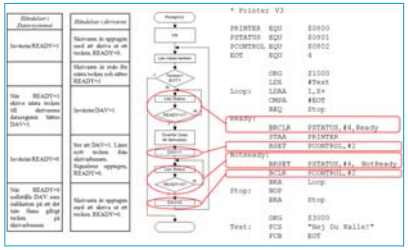
Att självständigt kunna:

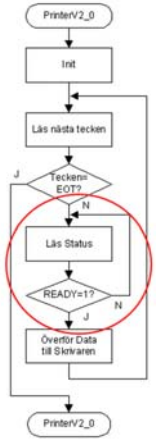
- beskriva och tillämpa olika principer för överföring mellan centralenhet och kringenheter så som: ovillkorlig eller villkorlig överföring, statustest och rundfrågning.
- konstruera program för systemstart och med stöd för avbrottshantering från olika typer av kringenheter.
- kunna beskriva metoder 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, statustest och rundfrågning.*









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

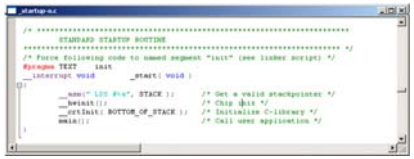
Exempel 4.43 Placering av Exceptionvektorer, assemblerkod

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

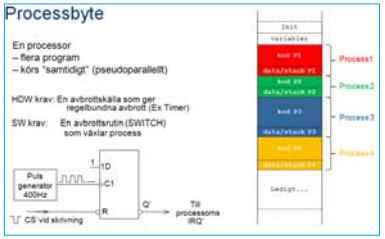
```
      ORG       $FFF2
      FDB       irq_service_routine
      FDB       xirq_service_routine
      FDB       software_interrupt_service_routine
      FDB       illegal_opcode_service_routine
      FDB       cop_service_routine
      FDB       clock_monitor_fail_service_routine
      FDB       Application_Start

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

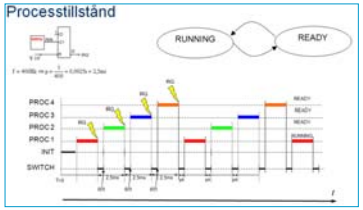
Vår slutliga "appstart" blir nu:

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

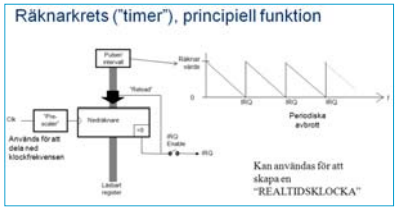


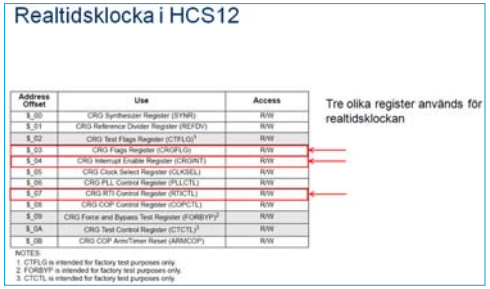
- *kunna beskriva metoder och mekanismer som är centrala i systemprogramvara så som pseudoparallell exekvering och hantering av processer.*

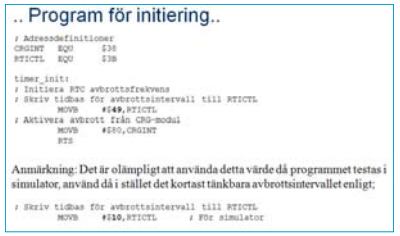


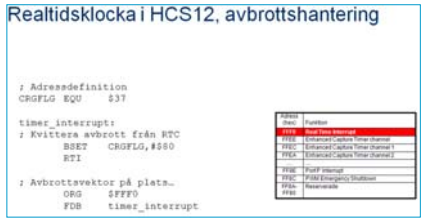


- *beskriva och använda kretsar för tidmätning.*

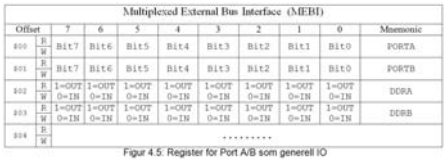




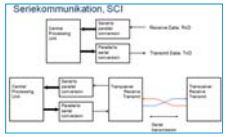


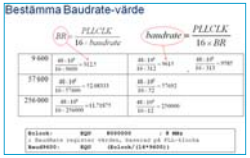


- *beskriva och använda kretsar för parallell respektive seriell överföring.*



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



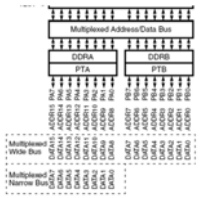


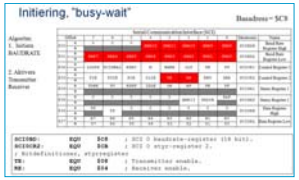
Exempel 4.50

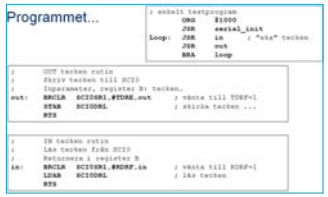
Ange i såväl assemblerspråk som C, programkonstruktioner som initierar port A för användning som inport samt port B för användning som utport.

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;
```





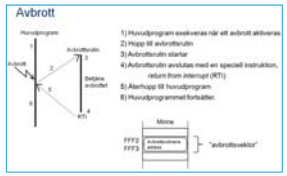


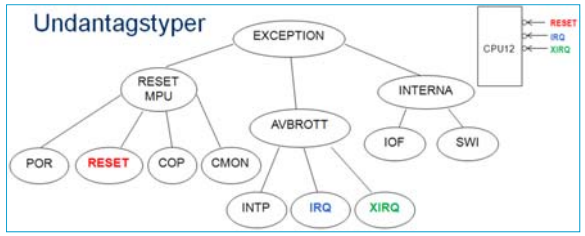
4. Undantagshantering i datorsystem

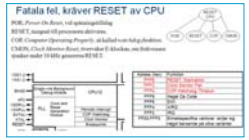
Att självständigt kunna:

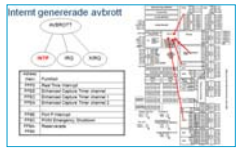
- beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart.
- konstruera enklare avbrottssystem med användning av digitala komponenter.
- beskriva och tillämpa olika metoder för prioritetshantering vid multipla avbrottskällor (mjukvarubaserad och hårdvarubaserad prioritering,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

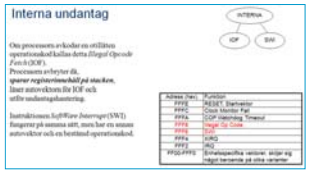
- *beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart.*

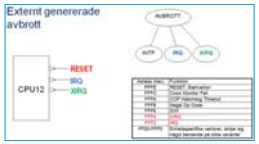




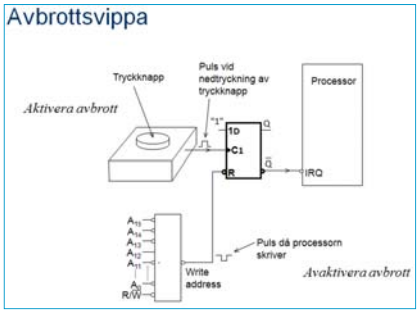


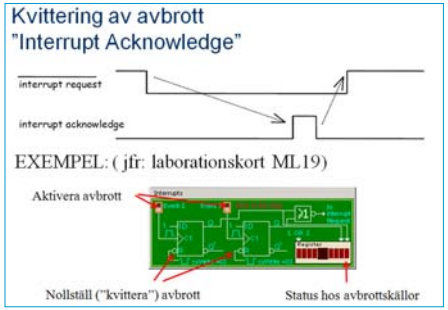






- *konstruera enklare avbrottssystem med användning av digitala komponenter.*





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