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"
- ❑ 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
* bitarmaskinens 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
```

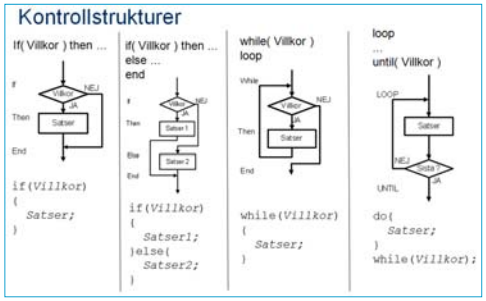
Returvärden via register

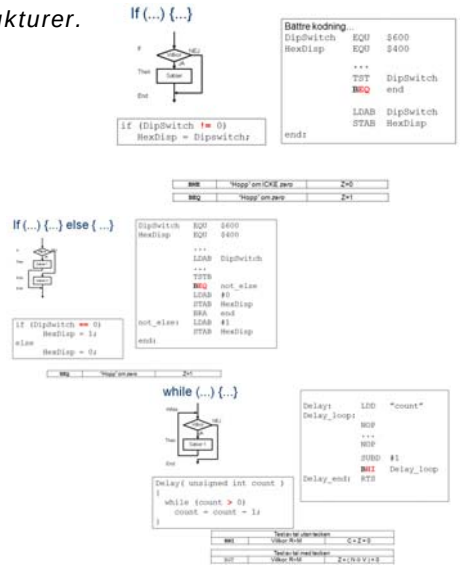
Register väljs, beroende på returvärdets typ (storlek), HCS12-exempel

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

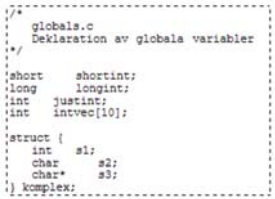
En regel (konvention) bestäms och följs därefter vid kodning av samtliga subrutiner

- *beskriva och använda olika kontrollstrukturer.*

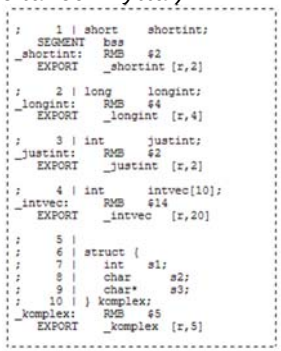




- *beskriva och använda sammansatta datatyper (fält och poster) och enkla datatyper (naturliga tal, heltal och flyttal).*



```
/*
   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
```

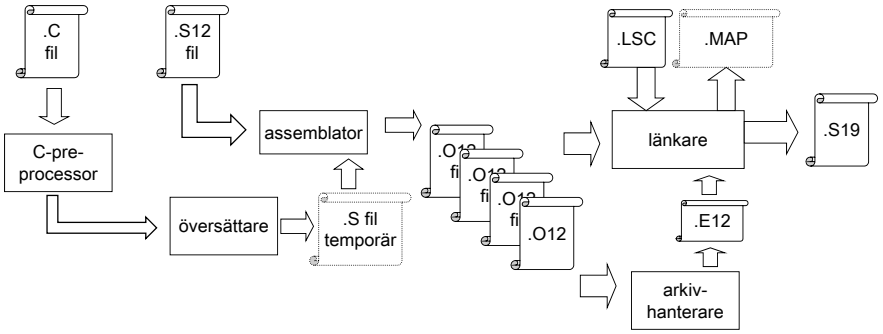
Kunna redogöra för olika lagringsklasser (GLOBAL, STATIC, LOCAL) och ”synlighet”.

2. 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.*

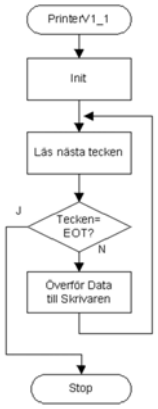
Dessa lärandemål har vi kontrollerat under laborationer.

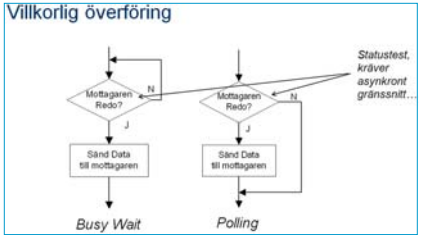
3. Systemprogrammerarens bild av inbäddade system

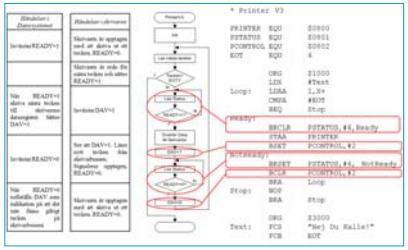
Att självständigt kunna:

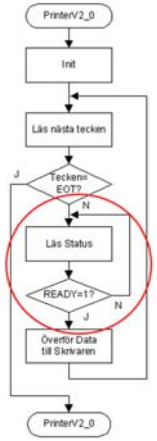
- beskriva och tillämpa olika principer för överföring mellan centralenhet och kringenheter så som: ovillkorlig eller villkorlig överföring, statustest och rundfrågning.
- konstruera program för systemstart och med stöd för avbrottshantering från olika typer av kringenheter.
- kunna beskriva metoder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

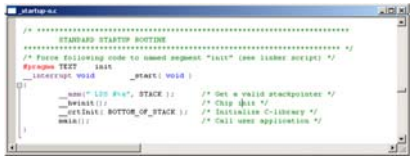
Exempel 4.43 Placering av Exceptionvektorer, assemblerkod

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

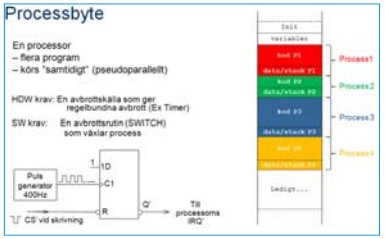
```
      ORG       $FFF2
      FDB       irq_service_routine
      FDB       xirq_service_routine
      FDB       software_interrupt_service_routine
      FDB       illegal_opcode_service_routine
      FDB       cop_service_routine
      FDB       clock_monitor_fail_service_routine
      FDB       Application_Start

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

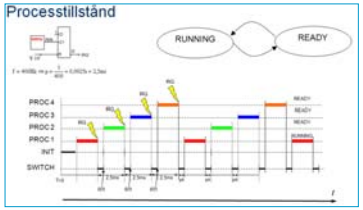
Vår slutliga "appstart" blir nu:

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

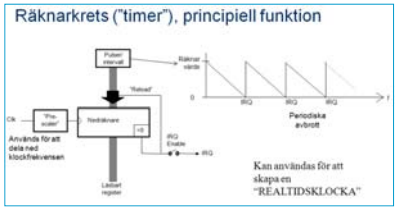


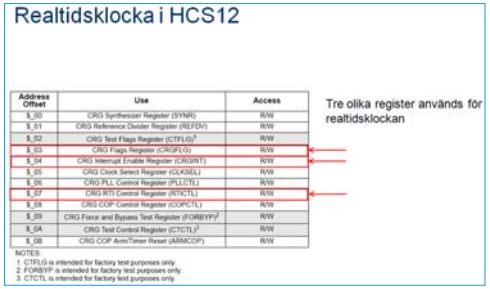
- *kunna beskriva metoder och mekanismer som är centrala i systemprogramvara så som pseudoparallell exekvering och hantering av processer.*





- *beskriva och använda kretsar för tidmätning.*





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)[1]	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)[2]	R/W
$_0A	CRG Test Control Register (CTCTL)[3]	R/W
$_0B	CRG COP Arm/Timer Reset (ARMCOP)	R/W

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
        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    #$10,RTICTL     ; För simulator
```

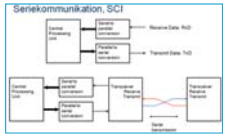
```
Realtidsklocka i HCS12, avbrottshantering
; Adressdefinition
CRGFLG  EQU     $37

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

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

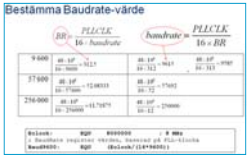

- *beskriva och använda kretsar för parallell respektive seriell överföring.*

Multiplexed External Bus Interface (MEBI)

Offset		7	6	5	4	3	2	1	0	Mnemonic
$00	R	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	PORTA
	W									
$01	R	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	PORTB
	W									
$02	R	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	DDRA
	W	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	
$03	R	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	DDRB
	W	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	
$04	R	.....								
	W									

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



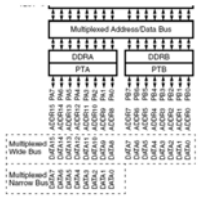


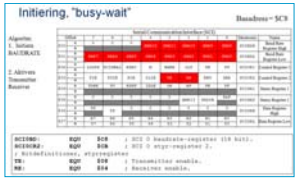
Exempel 4.50

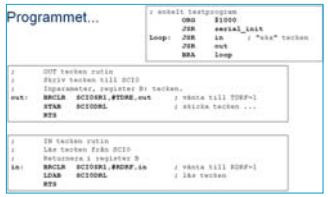
Ange i såväl assemblerspråk som C, programkonstruktioner som initierar port A för användning som inport samt port B för användning som utport.

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;
```





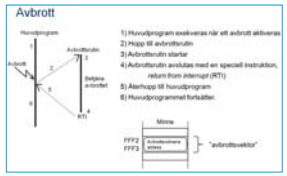


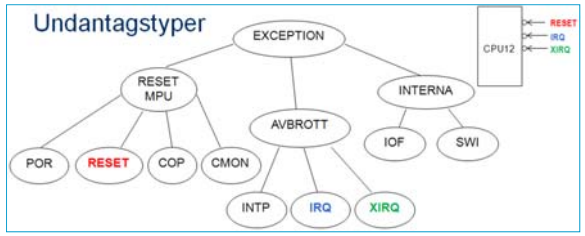
4. Undantagshantering i datorsystem

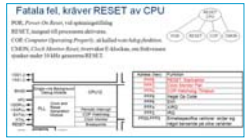
Att självständigt kunna:

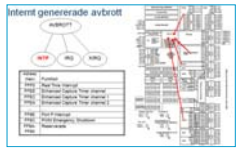
- beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart.
- konstruera enklare avbrottssystem med användning av digitala komponenter.
- beskriva och tillämpa olika metoder för prioritetshantering vid multipla avbrottskällor (mjukvarubaserad och hårdvarubaserad prioritering,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

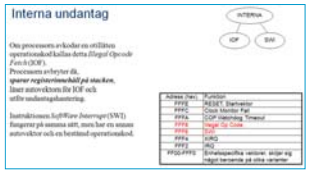
- *beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart.*

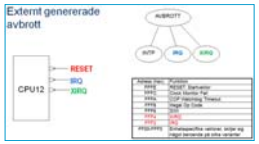




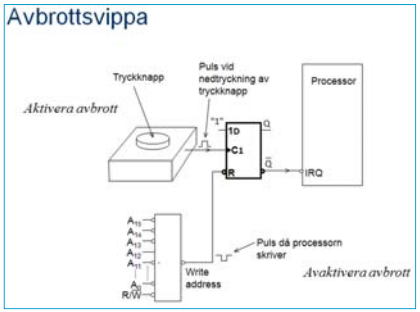


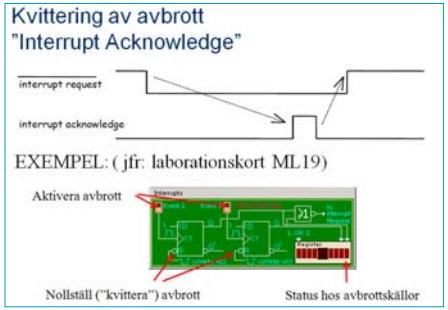




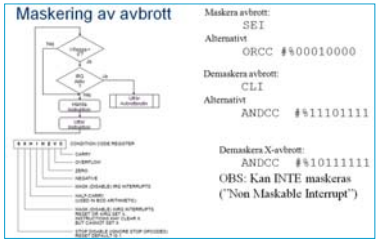


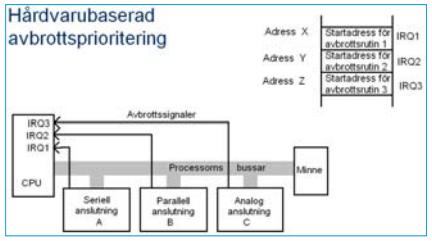
- *konstruera enklare avbrottssystem med användning av digitala komponenter.*

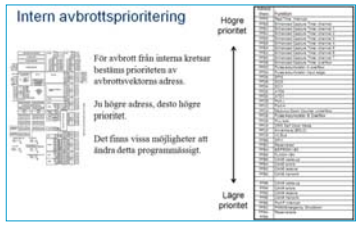


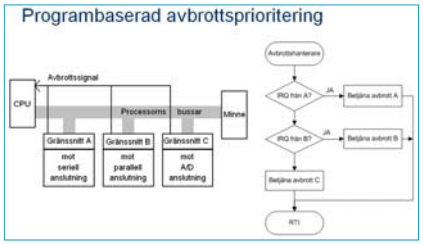


- *beskriva och tillämpa olika metoder för prioritetshantering vid multipla avbrottskällor (mjukvarubaserad och hårdvarubaserad prioritering, avbrottsmaskering, icke-maskerbara avbrott).*

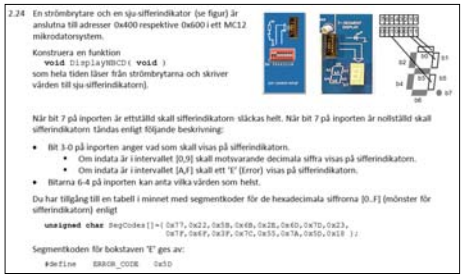








Av speciell vikt: ”maskinorienterad programmering...”



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...

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

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

a) Visa hur utrymme för lokala variabler reserveras i funktionen (*prolog*).
b) Visa funktionens aktiveringspost, ange speciellt offseter 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
```


Maskinorienterad Programmering 2011/2012

Sammanfattad...

Torsdag 1/3 (i morgon)
"Öppet hus" i lab 4220 (hela dagen)
Måndag 5/3
Tentamen, 14.00-18.00