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 returvärden.

Subrutiner för att manipulera styrregistret OUTONE och OUTZERO

```
* Subrutin OUTONE. Läser kopian av
* borrmaskinens styrord på adress
* DCCopy. Ettställer en av bitarna och
* skriver det nya styrordet till
* utporten DCTRL samt tillbaka till
* kopian DCCopy.
* Biten som 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

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.
```

```
*************************************
* 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(la,lb,lc);
```

översättas till:

```
        LDD    la
        LDX    lb
        LDY    lc
        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(la,lb,lc);
```

Översättas till:

```
        LDD    lc
        PSHD
;(alternativt STD   2,-SP)
        LDD    lb
        PSHD
        LDD    la
        PSHD
        BSR    dummyfunc
        LEAS   6,SP
```

Innehåll	Kommentar	Adressering via SP i subrutinen
lc.lsb	Parameter lc	
lc.msb		6,SP
lb.lsb	Parameter lb	
lb.msb		4,SP
la.lsb	Parameter la	
la.msb		2,SP
PC.lsb	Återhoppsadress,	
PC.msb	placeras här vid BSR	0,SP

```
dummyfunc:
        ..
        LDD     2,SP
; parameter la till register D
        ..
        LDD     4,SP
; parameter lb till register D
        ..
        LDD     6,SP
;parameter lc 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
```

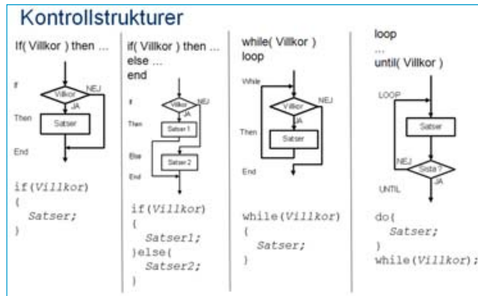
Returvärden via register

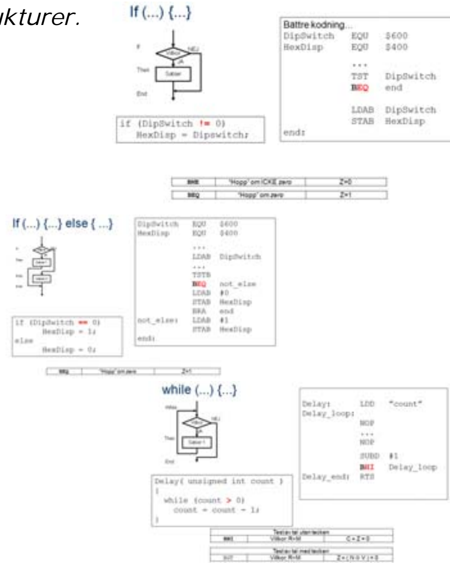
Register väljs, beroende på returvärdets typ (storlek), HCS12-exempel

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

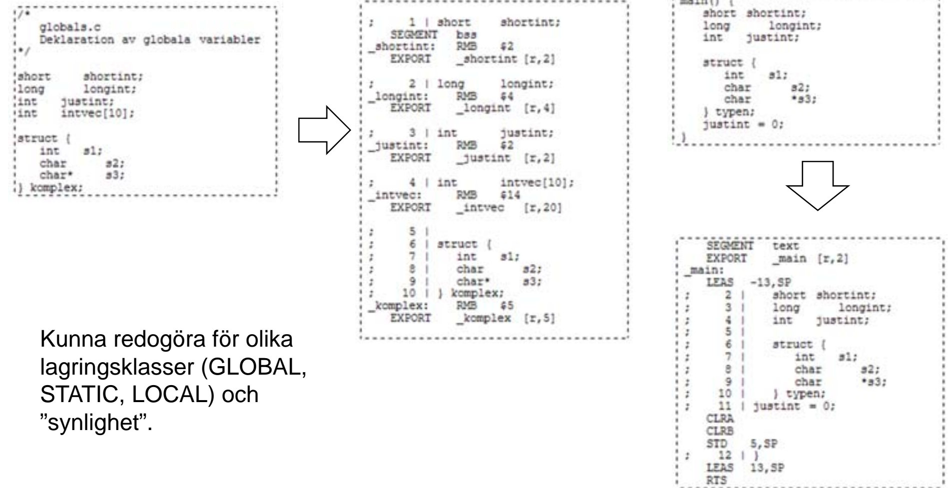
En regel (konvention) bestäms och följs därefter vid kodning av samtliga subrutiner

- *beskriva och använda olika kontrollstrukturer.*





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



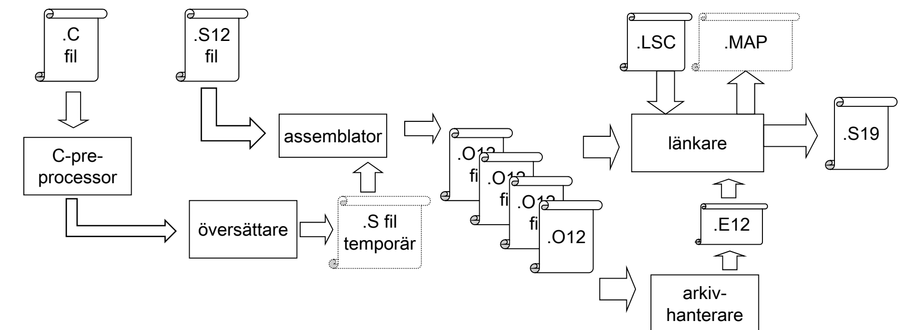
Kunna redogöra för olika lagringsklasser (GLOBAL, STATIC, LOCAL) och ”synlighet”.

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.*

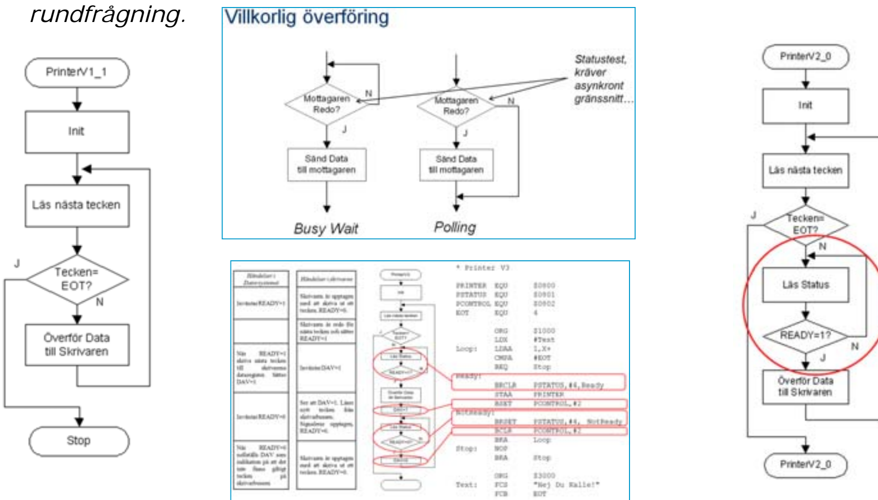
Dessa lärandemål har vi kontrollerat under laborationer.

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

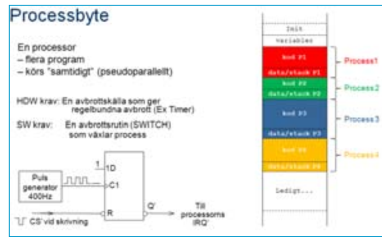
Följande programskelett illustrerar hur några avbrottsrutiner respektive avbrottsvektorer kan definieras i en fristående HCS12-applikation.

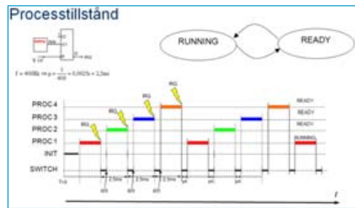
```
        ORG     $FFF2
        FDB     irq_service_routine
        FDB     xirq_service_routine
        FDB     software_interrupt_service_routine
        FDB     illegal_opcode_service_routine
        FDB     cop_service_routine
        FDB     clock_monitor_fail_service_routine
        FDB     Application_Start

; 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.*





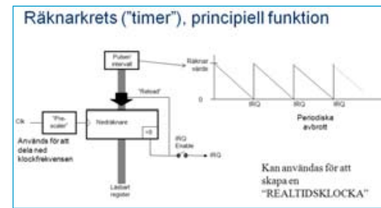
En realtidskärna

Jan Skansholm

```
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);
}
```


CHALMERS
Chalmers University of Technology
Maskinorienterad Programmering – DAT015, 2011/2012

- *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    #$10,RTICTL      ; För simulator
```

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)[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.

Tre olika register används för realtidsklockan

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
```


CHALMERS
Chalmers University of Technology
Maskinorienterad Programmering – DAT015, 2011/2012

- *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	....								

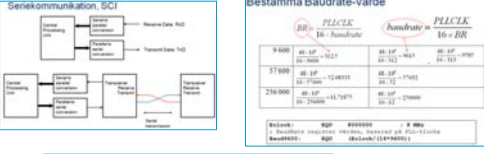
Figur 4.5: Register för Port A/B som generell IO

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"

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 TDRF=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
```

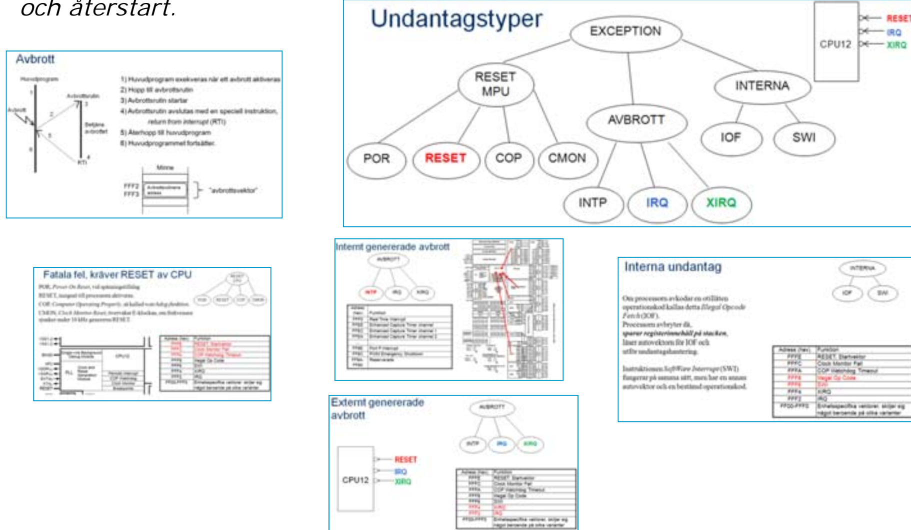

CHALMERS
Chalmers University of Technology
Maskinorienterad Programmering – DAT015, 2011/2012

4. Undantagshantering i datorsystem

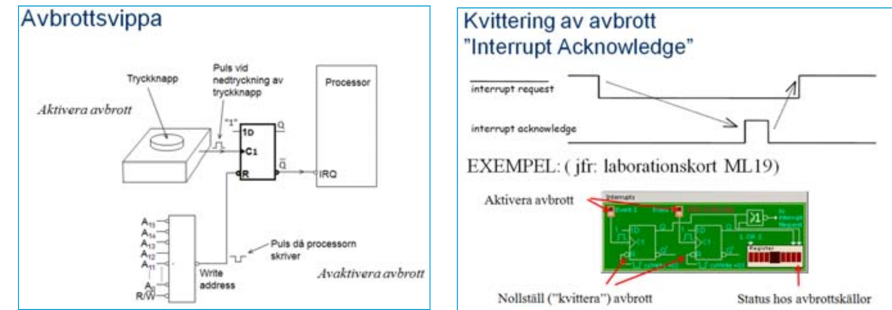
Att självständigt kunna:

- 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).

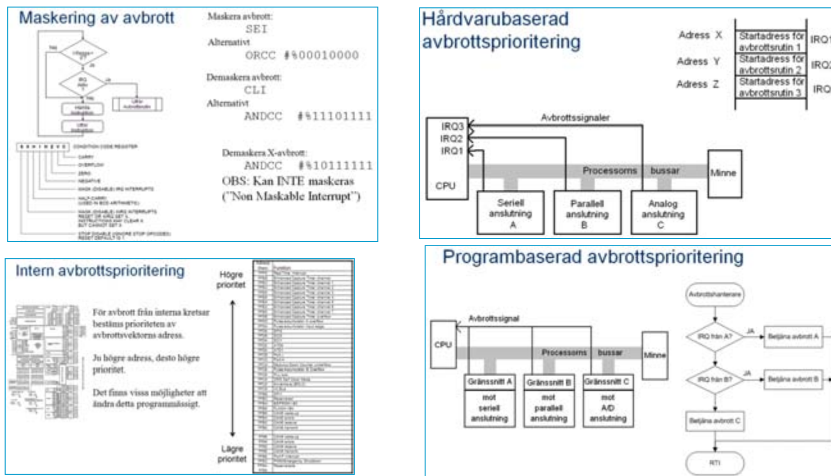
- *beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart.*



- *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..."

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 DisplayHBCD( 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...

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

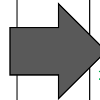
Pekare och dess användning...

```
/*
  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
```