

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;
}
ACC12 genererar följande kod:
SEGMENT text
EXPORT _callfunc [r,2]
_callfunc:
  2 | |
  3 | | aa = 1;
  LDD #1 aa = 1;
  STD 2,SP
  4 | #2 ab = 2;
  LDD #2 ab = 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;

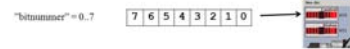
; 1 | int a;
SEGMENT bss
_a: RMB $2
EXPORT _a [r,2]

; 2 | static int b;
; 1: RMB $2
; symbolen 1 existerar endast under
; assemblering och motsvarar då
; symbolen 'b' i programmet. Symbolen
; 'b' exporteras inte.
    
```

Lagringsklass och synlighet.

Subrutiner för att manipulera styrregistret OUTONE och OUTZERO

- * Subrutin OUTONE. Läser kopien av
- * horisontalens styrgod på adress
- * DCopy. Ettställer en av bitarna och
- * skriver det nya styrgodet till
- * utporten DCTM samt tillbaka till
- * kopien DCopy.
- * Risen som motsvarar ges av innehållet
- * i R-registret (0-7) vid anrop.
- * Om (R) > 7 utförs ingenting.
- * Anrop: JSR #ritnummer
- * JSR OUTONE
- * Utdata: Inga
- * Registerpåverkan: Ingen
- * Anropad subrutin: Inga



- 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 "In Line"

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

```

SUB dummyfunc
FCB 10
...

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

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 la
PSHD
BSR dummyfunc
LEAS 6,SP
    
```

Innehåll	Kommentar	Adressering via SP i subrutinen
lc,lab	Parameter lc	6, SP
lb,lab	Parameter lb	4, SP
la,lab	Parameter la	2, SP
PC,lab	Återhoppadress, 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
    
```

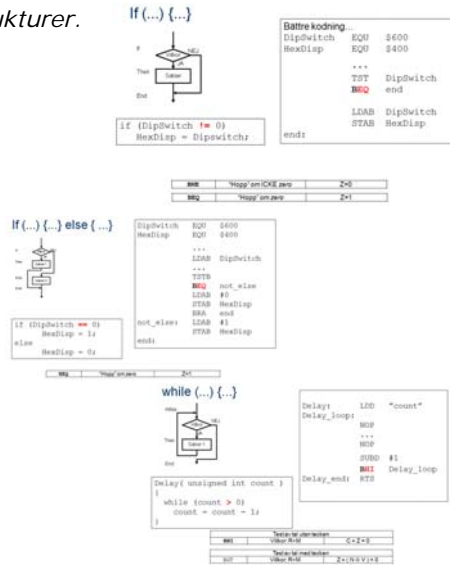
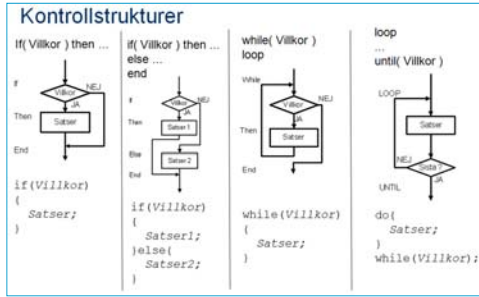
Returvärdet via register

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

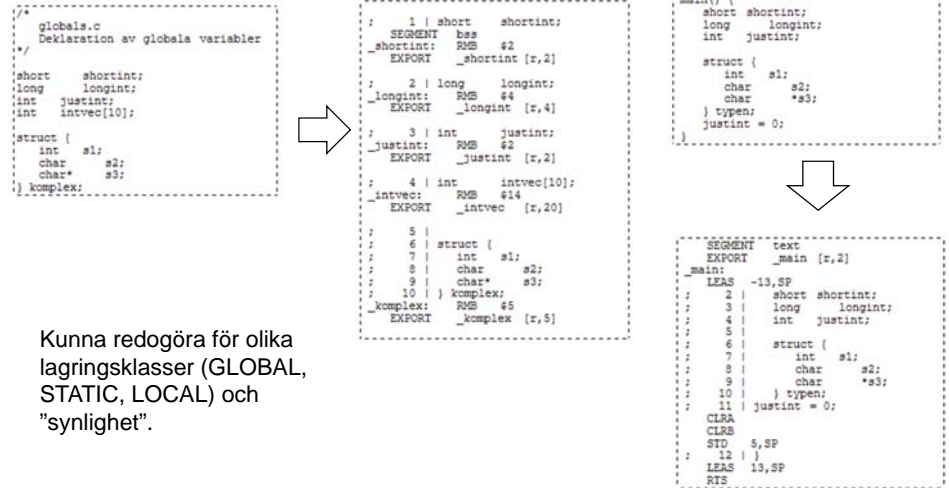
Storlek	Benämning	C-typ	Register
8 bitar	byte	char	B
16 bitar	word	short int	D
32 bitar	long	long int	Y/D

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

- beskriva och använda olika kontrollstrukturer.



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



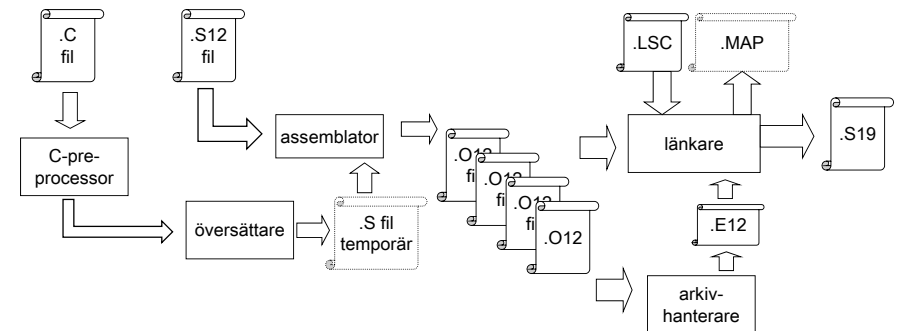
Kunna redogöra för olika lagringsklasser (GLOBAL, STATIC, LOCAL) och "synlighet".

2. Programutvecklingsteknik

Att självständigt kunna:

- beskriva översättningsprocessen, dvs. assemblatorns arbets sätt, 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 arbets sätt, preprocessorns användning, separatkompilering och länkning.



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

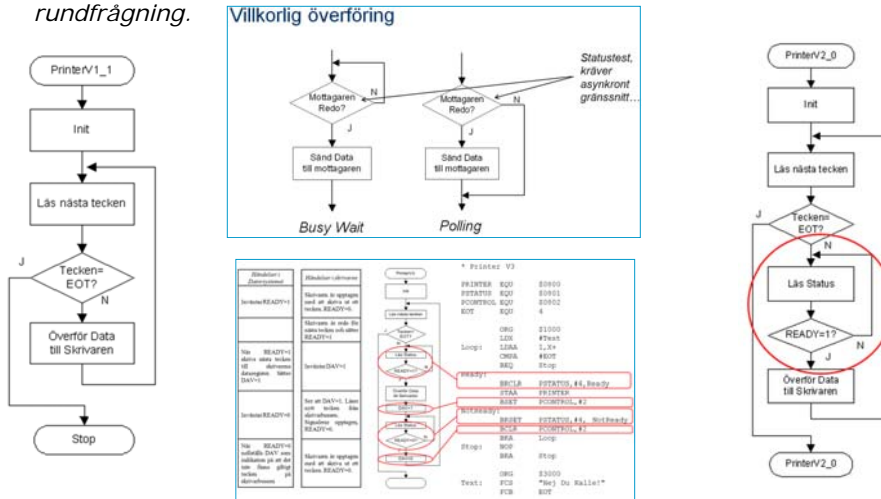
Dessa lärandemål har vi kontrollerat under laborationer.

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, statusstest 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.



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

Exempel 4.43 Placering av Exceptionvektorer, assemblerkod

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

```

ORG SFFF
FDB irq_serviceRoutine
FDB irq_serviceRoutine
FDB softwareInterruptServiceRoutine
FDB illegalOpcodeServiceRoutine
FDB copServiceRoutine
FDB clockMonitorFailServiceRoutine
FDB Application_Start

; Symbolen "Application_Start_Address" kan vara godtycklig.
Application_Start:
LDS #topOfStack
...
ANDCC #SFE ; nollställ I-flagga
JSR _main

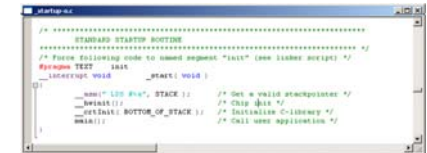
```

Vår slutliga "appstart" blir nu:

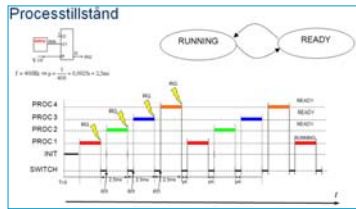
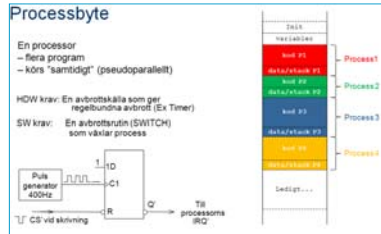
```

segment init
export _exit
import _main
function _start,_start_end
* Här börjar exekveringen...
_start
LDS #S2FFF
JSR _main
_exit: NOP
BRB _exit
_start_end

```



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



En realtidskärma

Jan Skansholm

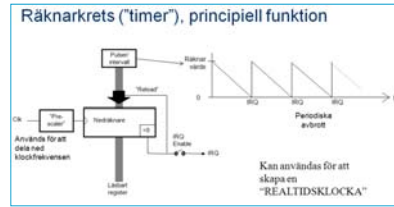
```

void watch(int channel_no, unsigned long int interval) {
    while (1) {
        int i;
        wait(0); // begär exklusiv tillgång till AD-omvandlaren
        adc_read(channel_no);
        do {
            delay(50);
            i = adc_get_value();
        } while (i == B00F);
        signal(0); // frislår AD-omvandlaren
        if (i == ERROR) // En felmeddelande
            warning(msg[channel_no]);
        else if (i <= 20 || i >= 40)
            warning(i1_msg[channel_no]);
        delay(interval);
    }
}

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

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

- beskriva och använda kretsar för tidmätning.



.. Program för initiering..

```

; Adressdefinitioner
CRIFLG EQU $38
RTICTL EQU $3B

timer_init:
; Initiera RTC avbrottsfrekvens
; Skriv tillbas för avbrottsintervall till RTICTL
    MOVW #49,RTICTL
; Aktivera avbrott från CRIF-modul
    MOVW #80,CRIFLG
    RTS
    
```

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

Address Offset	Use	Access
1_00	CRG Symbolic Register (SYMR)	R/W
1_01	CRG Reference Divider Register (REFDN)	R/W
1_02	CRG Test Flags Register (CTFLG)	R/W
1_03	CRG Flags Register (CRIFLG)	R/W
1_04	CRG Interrupt Enable Register (CRINTL)	R/W
1_05	CRG Clock Select Register (CLKSEL)	R/W
1_06	CRG PLL Control Register (PLLCCTL)	R/W
1_0F	CRG RTI Control Register (RTICTL)	R/W
1_08	CRG CRIP Control Register (CRIPCTL)	R/W
1_09	CRG Force and Release Test Register (FOREFPT)	R/W
1_0A	CRG Test Control Register (CTCTL)	R/W
1_0B	CRG CRIP Auto/Timer Counter (ARICRCP)	R/W

Tre olika register används för realtidsklockan

Realtidsklocka i HCS12, avbrotts hantering

```

; Adressdefinition
CRIFLG EQU $38

timer_interrupt:
; Kvittera avbrott från RTC
    BSET CRIFLG,$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		
800	R	W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	PORTA
801	R	W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	PORTB
802	R	W	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	1=OUT	DDRA
803	R	W	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	DDRB
804	R	W	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	0=IN	DDRB

Seriell Kommunikation, SCI

Bestämna Baudrate-värde

Baudrate	CRIF	CRIFLG	CRIFLG	CRIFLG
9600	0x1F	0x1F	0x1F	0x1F
37500	0x1F	0x1F	0x1F	0x1F
230400	0x1F	0x1F	0x1F	0x1F

Exempel 4.50

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

```

Lösning:
PORTA EQU 0
PORTB EQU 1
DDRA EQU 2
DDRB EQU 3
...
CLR DDRA
MOVW #0xFF,DDRB
...

typedef struct sMEBI {
    volatile unsigned char porta;
    volatile unsigned char portb;
    volatile unsigned char dira;
    volatile unsigned char dirb;
}MEBI, *PMEBI;

#define MEBI_BASE 0
{ ( ( PHEBI ) ( MEBI_BASE ) )->dira) = 0;
  ( ( PHEBI ) ( MEBI_BASE ) )->dirb) = 0xFF;
}
    
```

Initiering, "busy-wait"

```

; Adressdefinitioner
PORTA EQU $00
PORTB EQU $01
DDRA EQU $02
DDRB EQU $03

; Adressdefinitioner
PORTA EQU $00
PORTB EQU $01
DDRA EQU $02
DDRB EQU $03

; Adressdefinitioner
PORTA EQU $00
PORTB EQU $01
DDRA EQU $02
DDRB EQU $03
    
```

Programmet..

```

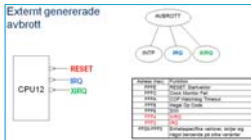
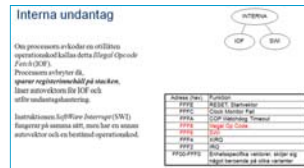
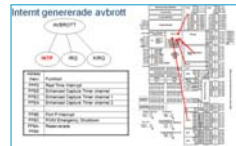
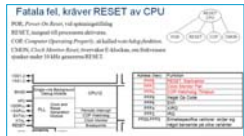
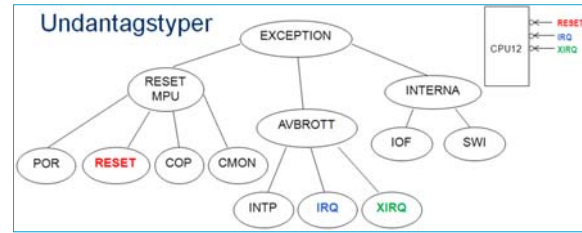
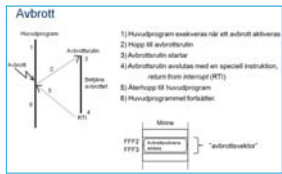
; Initialiseringsprogram
ORG $1000
; Skriv tillbas för "data" tecken
    MOVW #0,PORTA
    MOVW #0,PORTB
    MOVW #0,DDRA
    MOVW #0,DDRB
; Skriv tecken till PORTA
    MOVW #0,PORTA
; Skriv tecken till PORTB
    MOVW #0,PORTB
; Skriv tecken till DDRA
    MOVW #0,DDRA
; Skriv tecken till DDRB
    MOVW #0,DDRB
    
```

4. Bestämning av undantagshantering i datorsystem

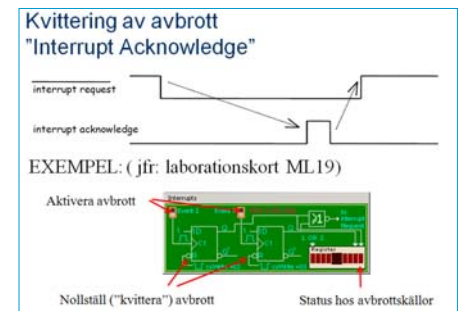
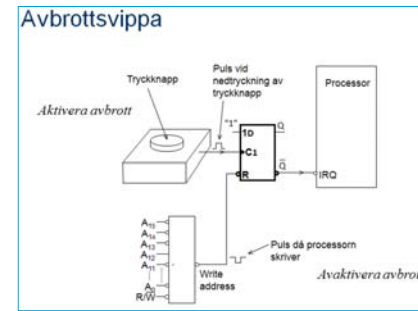
Att självständigt kunna:

- 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.



- 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).

