Maskinorienterad Programmering 2010/11

Maskinnära programmering – C

Ur innehållet:

- Kodningskonventioner
- Att tänka på då man programmerar i 'C'

Kodningskonventioner (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*.

Beroende på datatyp används för returparameter HC12's register enligt följande tabell:

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

Exempel (ES 2.13):

Skriv en egen version av standardfunktionen `strcpy`.

a) Använd pekare.
b) Använd indexering.
c) Använd XCC12, kompilera de båda versionerna till assemblerkod och jämför resultaten.

Lösning (ES 2.13):

Specifikation av strcpy:

Synopsis

```
#include <string.h>
void strcpy(char * s1, const char * s2);
```

Description

The `strcpy` function copies the string pointed to by `s2` (including the terminating null character) into the array pointed to by `s1`. If copying takes place between objects that overlap, the behavior is undefined.

Returns

The `strcpy` function returns nothing.

Lösning (ES 2.13):

a) Med pekare…

```
void strcpy (char *s1, const char *s2)
{
            while (*s1++ = *s2++)
                  ;
}
```

b) Med indexering…

```
void strcpy (char *s1, const char *s2)
{
            int   i = 0;
            while (s1[i] = s2[i])
                  i++;
}
```



```
// pekare
;           void strcpy (char *s1, const char *s2)
_strcpy:
;           {
;                       while (*s1++ = *s2++)
_1:
            LDX         2,SP
            LDY         4,SP
            LDAB        1,Y+
            STAB        1,X+
            STX         2,SP
            STY         4,SP
            TSTB
            BNE         _1
;                                   ;
;           }
            RTS
```

```
// indexering
;           void strcpy (char *s1, const char *s2)
_strcpy:
;           {
            LEAS        -3,SP
;                       int         i = 0;
            CLRA
            CLRB
            STD         1,SP
;                       while (s1[i] = s2[i])
_1:
            LDD         1,SP
            ADDD        7,SP
            TFR         D,X
            LDAB        0,X
            STAB        0,SP
            LDD         1,SP
            ADDD        5,SP
            TFR         D,X
            LDAB        0,SP
            STAB        0,X
            TSTB
            BEQ         _2
;                                   i++;
            LDX         1,SP
            INX
            STX         1,SP
            BRA         _1
_2:
;           }
            LEAS        3,SP
            RTS
```


Exempel (ES 2.26):

Följande C-deklarationer har gjorts på ”toppnivå” (global synlighet):

```
char    a,b,c;
char    min( char a, char b );
```

a) Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12.

b) Visa hur följande sats översätts till assemblerkod för HCS12:

```
c = min( a , b );
```

Vi löser på tavlan...

Exempel:

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

```
void  function( long c, char b, unsigned int a )
{
      char  d;
      long  e;
.....
```

a) Visa hur utrymme för lokala variabler reserveras i funktionen (*prolog*).

b) Visa funktionens *aktiveringspost*, ange speciellt offseter för parametrar och lokala variabler.

Vi löser på tavlan...

Exempel

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

```
void  function( long c, char b, unsigned int a )
{
      char  d;
      long  e;
.....
```

a) Visa hur tilldelningen `d = b;` utförs i assemblerspråk

b) Visa hur tilldelningen `e = a;` utförs i assemblerspråk

Vi löser på tavlan...

Exempel: Utgå från följande översättning av `strcpy` och försök förbättra koden

```
// pekare
;       void strcpy (char *s1, const char *s2)
_strcpy:
;       {
;               while (*s1++ = *s2++)
_1:
        LDX     2,SP
        LDY     4,SP
        LDAB    1,Y+
        STAB    1,X+
        STX     2,SP
        STY     4,SP
        TSTB
        BNE     _1
;                       ;
;       }
        RTS
```

Vi löser på tavlan...

Makroexpansion av preprocessor

```
#define   square(x) x*x
int  a,b;

b = square (a);/* b = a * a */

b = square (a+1);   /* b = a+1 * a+1 */
```

.. sätt ut parenteser...

```
#define   square(x) (x)*(x)
```


Felaktig användning av ”boolean”

```
int  a,b,c, status;

status = 0;          /* ”false” */
status = 3 < 2 <1;   /* ”true” */

a=3; b=2; c=1;
status = a < b < c;
```

”false”->0

”true”->1

Svag typning

`char` är en heltalstyp i C...

`'1'` är samma sak som 49...
`'1'+'3'` är 100 ...

```
char c;
c = getchar();
while( c != EOF ){...
```

`EOF` är definierad som -1
`char` kan vara *signed* eller *unsigned* (ANSI-C)
dvs. -1 eller 255...

Sidoeffekter

`++` och `--` används i uttryck och satser

```
i = 3;
a[i] = i++;
```

Vad blir resultatet?

a[3] = 4 eller
a[4] = 4

a[3] = 3 eller
a[4] = 3

```
int x, m[2];
i = 0;
m[++i] = 0;

m[++i] = 0;
```

objekt	värde
x	0
m[1]	0
m[0]	

”Dolda” fel

```
a = 3;
if( a = 4 )
     a = a+1;
```

Resultat: `a` blir `5`

```
a = 3;
if( a == 4 );
     a = a+1;
```

Resultat: `a` blir `4`

```
while( a = 0 )
{
   utförs aldrig
}
```

```
while( a = 1 )
{
   utförs för evigt...
}
```


”Dolda” fel

```
int   a,b;
void  initialize()
{
      if (a>0) b = 1;
      else b = 0;
}
main()
{
      a=1;
      b=0;
      initialize;
      printf( "%d %d\n",a,b);
}
```

Utskrift: `1 0` i stället för `1 1` ...

```
;     13 | {
;     14 |     a=1;
        LDD    #1
        STD    _a
;     15 |     b=0;
        CLRA
        CLRB
        STD    _b
;     16 |     initialize;
;     17 |     printf( "%d..;
        PSHD
        LDD    _a
        PSHD
        LDX    #_5
        PSHX
        JSR    _printf
        LEAS   6,SP
;     18 | }
        RTS
```

Pekararitmetik

```
char  *p;
char  c;

      c = *(p++);
      c = (*p)++;
```

Vad betyder

```
      c = *p++;
```

Svar:

```
      c = *(p++);
            =
      c = *p++;
```

```
;  c = *(p++);
        LDX     _p
        LDAB    1,X+
        STX     _p
        STAB    _c

; c = (*p)++;
        LDAB    [_p,pcr]
        INC     [_p,pcr]
        STAB    _c
```


Konstanter...

Föregås konstant av 'O' tolkas den som oktal, dvs:

```
O31 == 25
```

Kan lätt förväxlas/misstolkas...

Konstanten 0 kan ha väldigt olika betydelser....

```
#define     NULL        ((void *) 0)
#define     FALSE       0
```

Och förstås, numeriskt värde `0` ...

```
char arr[3] = {'a','b','c'};

char arr[3] = {"abc"};
```

```
_arr:
        FCB     $61
        FCB     $62
        FCB     $63
```

```
_arr:
        FCS     "abc"
        FCB     0
```