

Programmering av inbyggda system 2012/2013

Kodningskonventioner och programbibliotek

Ur innehållet:

Kodningskonventioner

maskinnära programmering i C och
assemblerspråk

Programbibliotek

32-bitars operationer med 16 bitars processor

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

Övningsexempel:

Skriv en egen version av standardfunktionen `strncpy` med användande av **pekare** (indexering är inte tillåten...)

1. Använd XCC12, kompilera till assemblerkod.
2. Utgå från resultatet och förbättra assemblerkoden så långt du kan.

Specifikation av `strncpy`:

Synopsis

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

Description

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

Returns

The `strncpy` function returns nothing.

Lösning (rättfram implementering):

```
void strncpy(char *s1, char *s2, int n)
{
    while (*s2 != 0) {
        *s1++ = *s2++;
        if (--n <= 0)
            break;
    }
    *s1 = 0;
}
```

```
; void strncpy (char *s1, const char *s2, int n)
; _strncpy:
; 4 | {
; 5 |     while (*s2 != 0) {
_2:
    TST     [4, SP]
    BEQ     _3
; 6 |         *s1++ = *s2++;
    LDX     2, SP
    LDY     4, SP
    LDAB    1, Y+
    STAB    1, X+
    STX     2, SP
    STY     4, SP
; 7 |         if (--n == 0)
    LDY     6, SP
    DEY
    STY     6, SP
    TFR     Y, D
    CPD     #0
    BEQ     _3
; 8 |         break;
_4:
    BRA     2
_3:
; 9 |     }
; 10 |     *s1 = 0;
    CLR     [2, SP]
; 11 | }
RTS
```

Bestäm kandidater för registerallokering:
Parametrar och lokala variabler:

4, SP: (s2) typen 'pekare' 3 st referenser
2, SP: (s1) typen 'pekare' 3 st referenser
6, SP: (n) typen 'int' 2 st referenser

Register X och Y är självskrivna pekare

Register D kan hålla en 'int'
Men då kan inte B användas för
datakopieringen...

Lösning:
Allokera X till s1
Allokera Y till s2
Allokera D till n
Använd MOV för datakopiering...

```

; void strncpy (char *s1, const char *s2, int n)
_strncpy:
; 4 | {
;     LDD      6, SP
;     LDY      4, SP
;     LDX      2, SP
; 5 |     while (*s2 != 0) {
_2:
;     TST      ,Y
;     BEQ      _3
; 6 |         *s1++ = *s2++;
;     MOVB     1, Y+, 1, X+
; 7 |         if (--n == 0)
;     SUBD     #1
;     BGT      _2
; 8 |         break;
_3:
; 9 |     }
; 10 |     *s1 = 0;
; 11 |     CLR    ,X
;     RTS
    
```

Med lämplig registerallokering kan koden förbättras betydligt...

n -> D
 s2 -> Y
 s1 -> X

Exempel (ES 2.26):

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

```

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

- Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12.
- Visa hur följande sats översätts till assemblerkod för HCS12:

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

Vi löser på tavlan...

Exempel (ES 2.32):

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;
    . . . . .
```

- Visa hur utrymme för lokala variabler reserveras i funktionen (*prolog*).
- Visa funktionens *aktiveringspost*, ange speciellt offseter för parametrar och lokala variabler.
- Visa hur tilldelningen `d = b;` utförs i assemblerspråk
- Visa hur tilldelningen `e = a;` utförs i assemblerspråk

Vi löser på tavlan...

Programbibliotek, HCS12 ordlängder och datatyper

7	A	0	7	B	0	8-bitarsackumulatorer A och B
15	D	0				eller
15	X	0				Index register X
15	Y	0				Index register Y
15	SP	0				Stackpekare SP
15	PC	0				Programräknare PC
S X H I N Z V C						Statusregister CCR

```
char    c;    /* 8-bitars datatyp, storlek byte */
short   s;    /* 16-bitars datatyp, storlek word */
long    l;    /* 32-bitars datatyp, storlek long */
int     i;    /* storlek implementationsberoende */
```

Lämpliga arbetsregister för **short** och **char** är **D** respektive **B**

32 bitars datatyper ryms ej i något CPU12-register.

Operationer implementeras i programbibliotek.

Aritmetiska operatorer (+,-,×,/,%)

Addition
Subtraktion
Multiplikation
Heltalsdivision
Restdivision

I den bästa av världar kan alla operatorer användas på godtyckliga sifferkombinationer.

Dagens programspråk lever dessvärre inte i den bästa världen...

Operator OCH datatyp avgör val av maskininstruktion, eller sekvens av instruktioner...

Addition ...

Pseudo språk:

```
char  ca,cb,cc;
...
ca = cb + cc;
```

8 bitar

```
...
LDAB    cb        ; operand 1
ADDB    cc        ; adderas
STAB    ca        ; skriv i minnet
```

Pseudo språk:

```
short sa,sb,sc;
...
sa = sb + sc;
```

16 bitar

```
LDD     sb        ; operand 1
ADDD    sc        ; adderas
STD     sa        ; skriv i minnet
```

32 bitar

Pseudo språk:

```
long  la,lb,lc;
...
la = lb + lc;
```

```
LDD     lb+2      ; minst signifikanta "word" av b
ADDD    lc+2      ; adderas till minst signifikanta "word" av c
STD     la+2      ; tilldela, minst signifikanta "word"
LDD     lb        ; mest signifikanta "word" av b
ADCB    lc+1      ; adderas till låg byte av mest signifikanta
          ; "word" av c
ADCA    lc        ; adderas till hög byte av mest signifikanta
          ; "word" av c
STD     la        ; tilldela, mest signifikanta "word"
```

Subtraktion ...

Pseudo språk:

```
char  ca,cb,cc;
      ...
ca = cb - cc;
```

8 bitar

```
...
LDAB    cb      ; operand 1
SUBB    cc      ; subtraheras
STAB    ca      ; skriv i minnet
```

Pseudo språk:

```
short sa,sb,sc;
      ...
sa = sb - sc;
```

16 bitar

```
LDD     sb      ; operand 1
SUBD    sc      ; subtraheras
STD     sa      ; skriv i minnet
```

32 bitar

Pseudo språk:

```
long  la,lb,lc;
      ...
la = lb - lc;
```

```
LDD     lb+2    ; minst signifikanta "word" av b
SUBD    lc+2    ; subtraheras från minst signifikanta "word" av c
STD     la+2    ; tilldela, minst signifikanta "word"
LDD     lb      ; mest signifikanta "word" av b
SBCB    lc+1    ; subtrahera låg byte av mest signifikanta
          ; "word" av c
SBCA    lc      ; subtrahera hög byte av mest signifikanta
          ; "word" av c
STD     la      ; tilldela, mest signifikanta "word"
```

Kodningskonventioner

13

Multiplikation ($P=X \times Y$) tal utan tecken, med addition/skift, $X > 0$, $Y > 0$.

$X=6=$ $=0110$ (multiplikator)

$Y=5=y_3y_2y_1y_0$ $=0101$ (multiplikand)

PP(0) 0000 $y_0=1 \Rightarrow \text{ADD } X$

+ 0110

0110

skifta

PP(1) 0011 0

$y_1=0 \Rightarrow \text{ADD } 0$

+ 0000

0011 0

skifta

PP(2) 0001 10

$y_2=1 \Rightarrow \text{ADD } X$

+ 0110

0111 10

skifta

PP(3) 0011 110

$y_3=0 \Rightarrow \text{ADD } 0$

+ 0000

0011 110

skifta

$P=PP(4)=$ $00011110 = 2^4+2^3+2^2+2^1 = 30$

Kodningskonventioner

14

Multiplikation ($P=X \times Y$) tal med tecken, med addition/aritmetiskt skift, $X < 0$, $Y < 0$.

$X = -6 = 1010$ (multiplikator) $-X = 0110$
 $Y = -5 = y_3y_2y_1y_0 = 1011$ (multiplikand)
Observera hur vi här använder en extra teckenbit, (5-bitars tal i operationen)

PP(0)	00000	$y_0=1 \Rightarrow \text{ADD } X$
	+ 11010	
	11010	skifta aritmetiskt
PP(1)	111010	$y_1=1 \Rightarrow \text{ADD } X$
	+ 11010	
	101110	skifta aritmetiskt
PP(2)	1101110	$y_2=0 \Rightarrow \text{ADD } 0$
	+ 00000	
	1101110	skifta aritmetiskt
PP(3)	11101110	$y_3=1 \Rightarrow \text{ADD } -X$
	+ 00110	
	00011110	skifta aritmetiskt
P=PP(4)=	00011110	= 30

Slutsatser:

- Multiplikation utförs enkelt med grundläggande operationer addition och skift.
- Vi kan **inte** använda exakt samma algoritm på tal med respektive utan tecken.
- Resultatet av en "unsigned" multiplikation av 2 st. n -bitars tal kräver $2 \times n$ bitars register.
- Resultatet av en "signed" multiplikation av 2 st n -bitars tal kräver $(2 \times n) - 1$ bitars register.

Implementering "papper och penna-metod"

```
long mul32 ( long a, long b)
{
    /* Multiplikation med "skift/add" */
    long result, mask;
    int i;
    mask = 1;
    result = 0;
    for( i = 0; i<32; i++ )
    {
        if ( mask & a )
            result = result + b ;
        b = b << 1;
        mask = mask << 1;
    }
    return result;
}
```

Fungerande men långsam metod...

HCS12, 8-bitars multiplikation...

Pseudo språk:

```
unsigned char  ca,cb,cc;
...
ca = cb * cc;
```

Assemblerspråk:

```
LDAB  cb      ; operand 1
LDAA  cc      ; adderas
MUL           ; multiplicera
STAB  ca      ; skriv i minnet
```

Endast "unsigned"...

HCS12, 16-bitars multiplikation (unsigned)...

Pseudo språk:

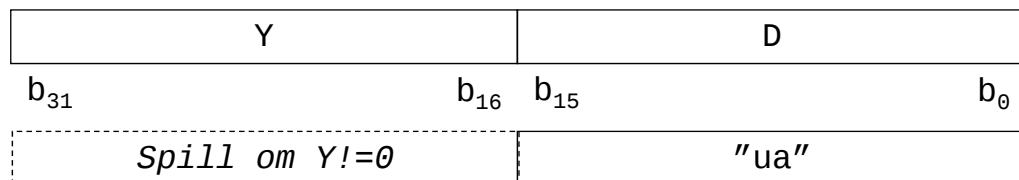
```
unsigned short ua,ub,uc;
```

```
ua = ub * uc;
```

Assemblerspråk:

```
LDD    ub    ; operand 1
LDY    uc    ; operand 2
EMUL      ; multiplicera
STD     ua    ; skriv i minnet
```

Resultat (2n bitar)



HCS12, 16-bitars multiplikation (signed) ...

Pseudo språk:

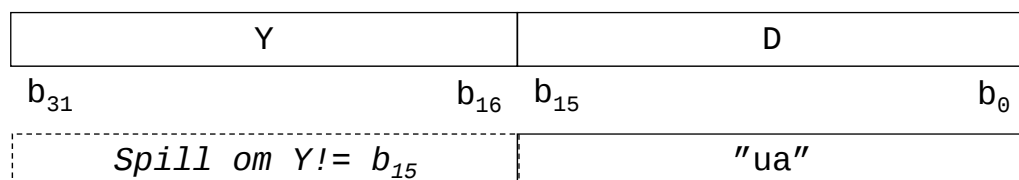
```
short ua,ub,uc;
```

```
ua = ub * uc;
```

Assemblerspråk:

```
LDD    ub    ; operand 1
LDY    uc    ; operand 2
EMULS     ; multiplicera
STD     ua    ; skriv i minnet
```

Resultat (2n bitar)



HCS12, 32-bitars multiplikation, med EMUL

Antag a , b 32-bitars tal, skriv:

$$a = ah \times 2^{16} + al, \quad ah = \text{MSW}(a) \text{ och } al = \text{LSW}(a)$$

$$b = bh \times 2^{16} + bl, \quad bh = \text{MSW}(b) \text{ och } bl = \text{LSW}(b)$$

Det gäller då att:

$$\begin{aligned} a \times b &= (ah \times 2^{16} + al) \times (bh \times 2^{16} + bl) = \\ &2^{32} (ah \times bh) + 2^{16} (ah \times bl + al \times bh) + (al \times bl) \end{aligned}$$

Detta är samma sak som:

$$(ah \times bh) \ll 32 + (ah \times bl + al \times bh) \ll 16 + (al \times bl)$$

Eftersom vi bara kräver 32 bitars resultat kan vi därför skriva:

$$a \times b = (ah \times bl + al \times bh) \ll 16 + (al \times bl)$$

Endast "unsigned"...

Implementering...

```
unsigned long mulu32 ( unsigned long a, unsigned long b)
{
    unsigned long  result;
    unsigned short ah,al,bh,bl;
    ah = (unsigned short )( a >> 16 );
    al = (unsigned short ) a ;
    bh = (unsigned short )( b >> 16 );
    bl = (unsigned short ) b ;
    result = (((unsigned long)( ah*bl + al*bh ))<< 16 ) + ( al*bl );
    return result;
}
long muls32 (long a, long b)
{
    long r;
    r = mulu32 ( ((a < 0) ? -a : a),((b < 0) ? -b : b) );
    if ( (a < 0) ^ (b < 0))
        return -r;
    else
        return r;
}
```

Division

En division kan skrivas som:

$$\frac{X}{Y} = Q + \frac{R}{Y}$$

Där:

- X är dividend
- Y är divisor
- Q är kvot, resultatet av heltalsdivisionen X/Y .
- R är resten, resultatet av modulusdivisionen $X \bmod Y$.

Av sambandet framgår att resten kan uttryckas: $R = X - Q \times Y$

Exempel: Decimal division, återställning av resten, 3967/15

$$X = 3967$$

$$Y = 15$$

$$R = X - Q \times Y$$

$ \begin{array}{r} 0264,4 \\ 15 \overline{) 3967,0} \\ \underline{-0} \\ 3967,0 \\ \underline{-30} \\ 967,0 \\ \underline{-90} \\ 67,0 \\ \underline{-60,0} \\ 7,0 \\ \underline{-6,0} \\ 1,0 \end{array} $	$ \begin{aligned} 3967 &= 3967 - 0 \times 15 \\ 3967 &= 3967 - (0 \times 10^3) \times 15 \\ 967 &= 3967 - (0 \times 10^3 + 2 \times 10^2) \times 15 \\ 67 &= 3967 - (0 \times 10^3 + 2 \times 10^2 + 6 \times 10^1) \times 15 \\ 7 &= 3967 - (0 \times 10^3 + 2 \times 10^2 + 6 \times 10^1 + 4 \times 10^0) \times 15 \\ 1 &= 3967 - (0 \times 10^3 + 2 \times 10^2 + 6 \times 10^1 + 4 \times 10^0 + 4 \times 10^{-1}) \times 15 \end{aligned} $	Utgångsläge steg 1 steg 2 steg 3 steg 4 steg 5
--	---	--

$$\text{DVS. } 3967/15 = 264,4 + 10/15 \times 10^{-1}$$

En divisionsalgoritm

Algoritm: *Division med återställning*

$$R = X - Q \times Y$$

$$Q = q_0 q_1 q_2 \dots q_{n-1}$$

n = antal kvotbitar att beräkna

$$R_0 = X$$

$$R_1 = R_0 - q_0 \times Y$$

$$q_0 = 1 \text{ om } R_1 \geq 0, q_0 = 0 \text{ annars}$$

för $i = 2..n$

$$R_i = 2 \times R_{i-1} - q_i \times Y$$

$$q_i = 1 \text{ om } R_i \geq 0, q_i = 0 \text{ annars}$$

Anm: För binära tal reduceras operationen $q_i \times Y$ till ADD Y.

Implementering: 32-bitars "unsigned" heltalsdivision

```
unsigned long divu32 (unsigned long a, unsigned long b)
{
    unsigned long rest = 0L;
    unsigned char count = 31;
    unsigned char c;

    do{
        if( a & 0x80000000 )
            c = 1;
        else
            c = 0;
        a = a << 1;
        rest = rest << 1;
        if(c)
            rest = rest | 1L;

        if(rest >= b){
            rest = rest - b;
            a = a | 1L;
        }
    } while(count--);
    return a;
}
```

Implementering: 32-bitars "signed" heltalsdivision

```
long divs32 (long a, long b)
{
    long r;

    r = divu32((a < 0 ? -a : a), (b < 0 ? -b : b));

    if ( (a < 0) ^ (b < 0))
        return -r;
    else
        return r;
}
```

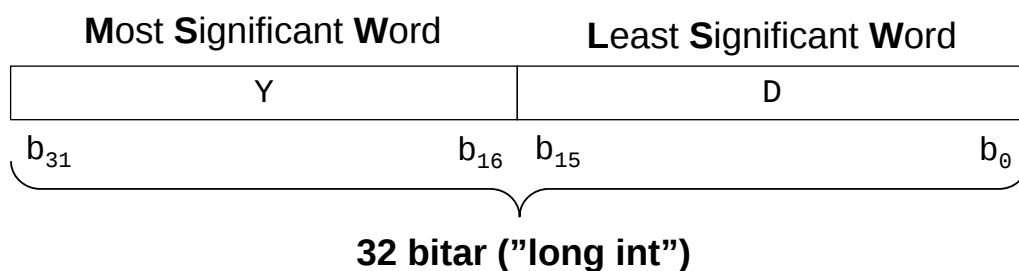
Implementering: 32-bitars "unsigned" restdivision

```
unsigned long modu32 (unsigned long a , unsigned long b)
{
    unsigned long c = a/b; /* heltalsdivision */
    return ( a - b * c );
}
```

Resultatet vid "signed" restdivision är implementationsberoende....

Sammanfattning, 32-bitars aritmetik för HCS12

Anropskonvention (L op R)	32 bitar op	Unsigned	Signed
PUSH LSW(R)			
PUSH MSW(R)	L+R (ADD)	add32(L, R)	
PUSH LSW(L)	L-R (SUB)	sub32(L, R)	
PUSH MSW(L)	L*R (MUL)	mulu32(L, R)	mul32(L, R)
JSR op32	L/R (DIV)	divu32(L, R)	div32(L, R)
(MSW i Y, LSW i D)	L%R (MOD)	modu32(L, R)	mod32(L, R)



Logik- och skiftoperationer

32 bitar op	Unsigned	Signed
L & R (AND)	and32(L, R)	
L R (OR)	or32(L, R)	
L ^ R (EOR)	eor32(L, R)	
L<<R (SHIFT LEFT)	shl32(L, R)	
L>>R (SHIFT RIGHT)	shru32(L, R)	shrs32(L, R)

