# Kodningskonventioner

Viktor Kämpe

# Varför kodningskonventioner?

- Förståelse för
  - Skillnaden mellan lokala/globala variabler.
  - Funktionsargument.
  - Returvärde.
- Möjliggör
  - Mix av assembler och C.

# Funktionsanrop utan argument

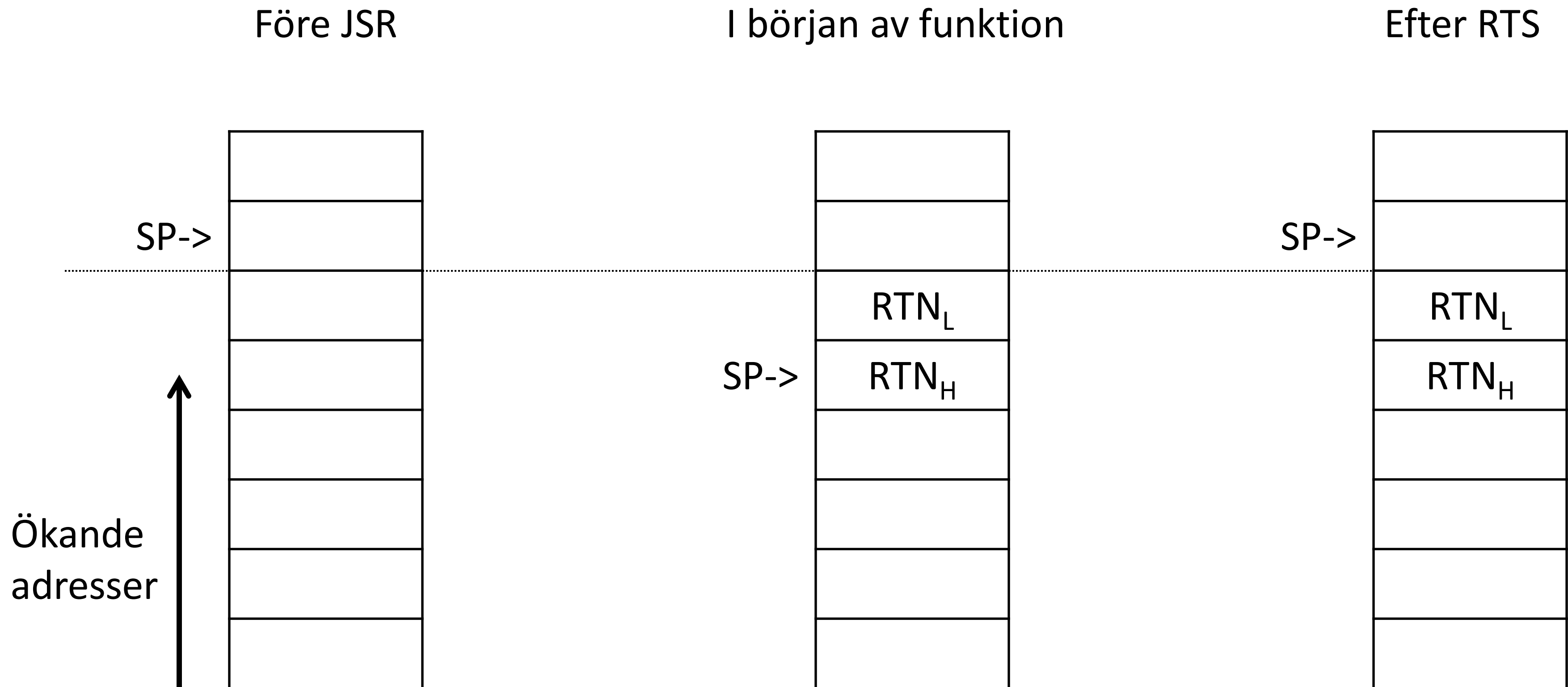
```
        JSR     delay




delay:
        LDAB    #$FF
loop_delay:
        DECB
        BNE     loop_delay
        RTS
```

CPU12 Assembler

```
delay();




void delay()
{
    unsigned char tmp = 0xFF;
    do {
        tmp--;
    } while(tmp);
}
```

C – program

# Stackens utseende



# Funktionsanrop med argument

- Anropskonventioner för XCC12:
  - Kallas också parameteröverföring.
  - Parametrarna överförs via stacken.
    - (Värdet av) argumenten pushas på stacken.
    - Ordningen är höger-till-vänster.

Detaljer i *Arbetsbok för MC12* på sidan 80

# Returvärde från funktion

- Returvärdet sker enligt XCC12's konventioner i:
  - Register **D**, om ej beskrivet nedan.
  - Register **B**, om returtyp är `char`.
  - Register **Y/D** om returtyp är `long` eller `float`.
    - MSW i **Y**, LSW i **D**.
  - Via (en pekare till) minnet om returtyp är en `struct`.

Se *Arbetsbok för MC12* på sidan 81.

# Ett enkelt C—program

```
#include <stdio.h>

int g_var;    //global synlighet

int myAdd(int x, int y)
{
     // lokal variabel
     int sum;
     sum = x+y;
     return sum;
}

int main()
{
     // lokala variabler
     int var1, var2;

     var1 = 4;
     g_var = 1;

     var2 = myAdd(g_var, var1);

     return 0;
}
```

Funktionsdeklaration

Funktionsanrop

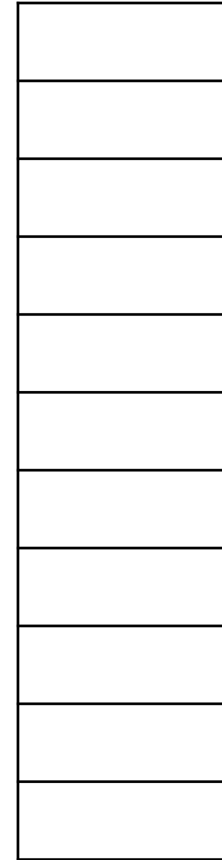
I följande slides använder vi konventioner för XCC12 och 16 bitars `int`

# Passa första parametern

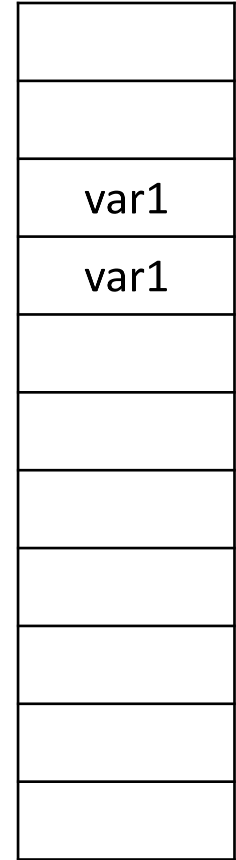
```c
int myAdd(int x, int y)
{
    // lokal variabel
    int sum;
    sum = x+y;
    return sum;
}


var2 = myAdd(g_var, var1);
```

Push

Före




Efter



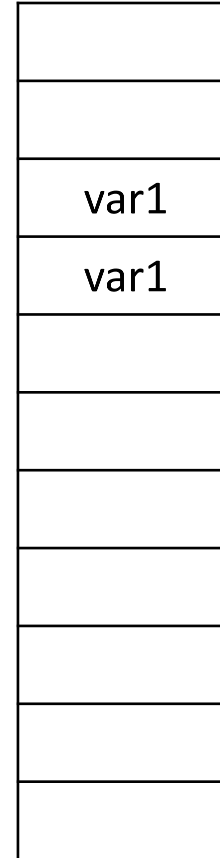
# Passa andra parametern

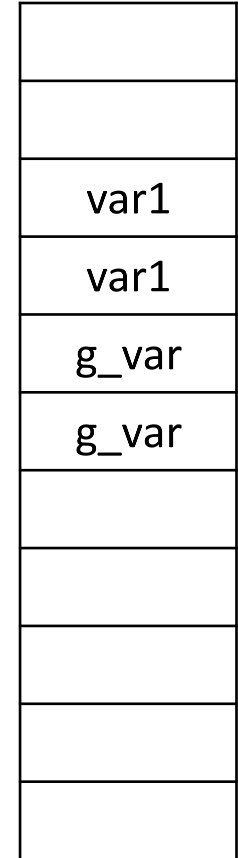
```
int myAdd(int x, int y)
{
    // lokal variabel
    int sum;
    sum = x+y;
    return sum;
}


var2 = myAdd(g_var, var1);
```

Push

Före

Efter



# Spara returadress på stacken

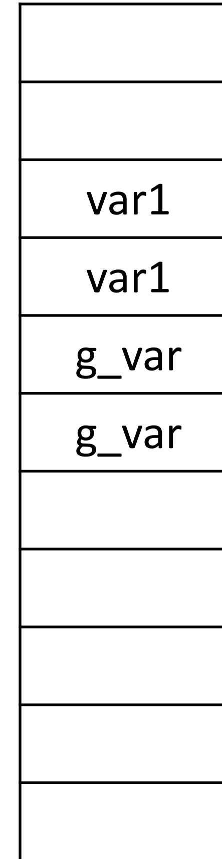
```
int myAdd(int x, int y)
{
    // lokal variabel
    int sum;
    sum = x+y;
    return sum;
}


var2 = myAdd(g_var, var1);
```

JSR

Före

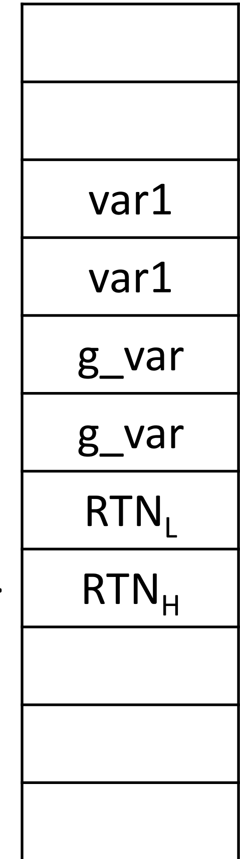
Ökande adresser

SP->




Efter

SP->



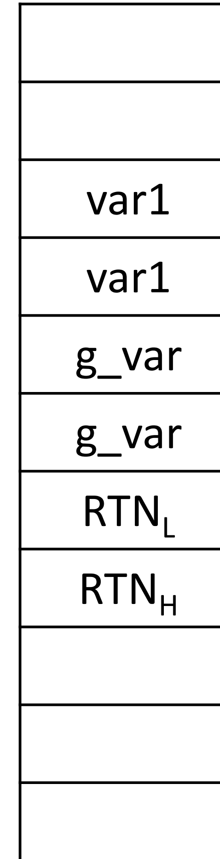
# Lokala variabler på stacken

- Lokala variabler ligger på stacken.
- Utrymme skapas genom att flytta stack pekaren (SP).

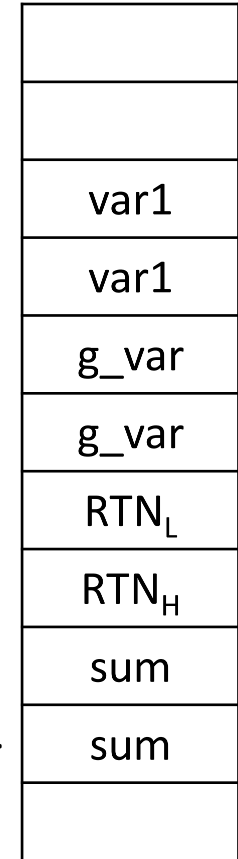
# Lokala variabler på stacken

```
int myAdd(int x, int y)
{
    // lokal variabel
    int sum;
    sum = x+y;
    return sum;
}


var2 = myAdd(g_var, var1);
```

Före

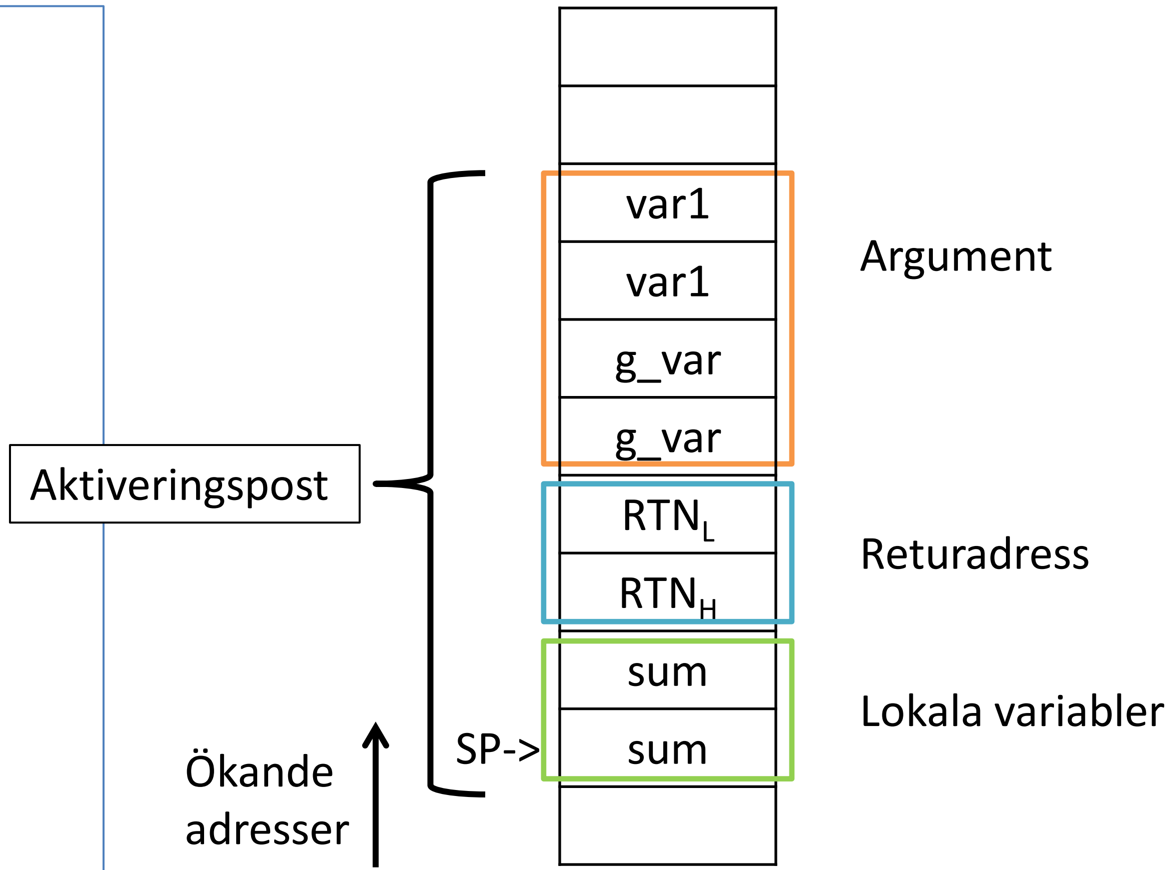



Efter



# Aktiveringspost / Stack frame

```
int myAdd(int x, int y)
{
    // lokal variabel
    int sum;
    sum = x+y;
    return sum;
}


var2 = myAdd(g_var, var1);
```



# Prolog

- Prologen av en funktion skapar utrymmet för lokala variabler
- På CPU12 så görs det genom att flytta SP
- LEAS
  – Load Effective Address to SP

# Epilog

- Epilogen av en funktion återlämnar minnet för lokala variabler.
- Görs på motsvarande sätt som prologen.

# [myAdd till assembler för hand]

# Ett funktionsanrop

1. Pusha argumenten på stacken.
2. JSR (pusha återhoppsadressen på stacken).
3. Prolog: skapar utrymme för lokala variabler.
4. { Funktionskroppen }
5. Lägg returvärde i rätt register.
6. Epilog: återlämnar utrymme för lokala variabler.
7. RTS (pop av programräknaren PC)
8. Återställ stacken till tillståndet innan argumenten pushades.

[assemblerkod från kompilator]

# Kombinera C och assembler

- Anropa assembler-rutiner från C
- Anropa C-rutiner från assembler
- Kodgenerering sker från olika filer, så ”ihopkopplingen” sker i länkningen.

# XCC genererar assembler från C

- Från C till assembler
- Från assembler till maskinkod
- Symbolerna i assemblern är samma som i C fast med ett understreck:

`myAdd()` i C

`_myAdd:` i assembler

# Import

IMPORT i <u>assembler</u> säger att symbolen kommer från en annan fil.

```
        IMPORT      _main
        ...
        JSR         _main
```

Symbol från annan fil ( t ex .c-fil)

Börja exekvera main-funktionen

# Extern

extern i C säger att symbolen kommer från en annan fil.

```
#include <stdio.h>

extern int g_var;
void myCLI();

void main()
{
    myCLI();
}
```

g_var definerad i annan fil ( t ex .c-fil)

Extern ej nödvändig för funktioner, prototypen säger: finns vid länkning.

```
        EXPORT    _myCLI
_myCLI:
        CLI
        RTS
```

# Synlighet av symboler

- Alla symboler synliga per-default i C
- Inga symboler synliga per-default i assembler
  - Gör symboler (i .s12 filen) synliga med `EXPORT`
  - Gör symboler från andra filer synliga med `IMPORT`

# static

`static` i C kan göra två saker

1. Ta bort synlighet (från andra filer) av symboler.
2. Allokera minne för lokala variabler som om de vore globala variabler (men fortfarande med lokal synlighet).

# Static variant 1

```
static int var;
```

Variabeln *var* kan ej ses från andra filer, oavsett om man använder `extern` eller `IMPORT`.

# Static variant 2

```
#include <stdio.h>

void testFkt()
{
    int var1 = 0;
    static int var2 = 0;
    var1++;
    var2++;
    printf("var1: %i, var2: %i \n", var1, var2);
}

int main()
{
    testFkt();
    testFkt();
    testFkt();
}
```

Utskrift:

```
var1: 1, var2: 1
var1: 1, var2: 2
var1: 1, var2: 3
```

var2 initialiseras till noll endast första gången vi anropar funktionen, men behåller sedan sitt värde mellan anrop.

# Viktiga koncept

- Aktiveringspost
- Prolog/Epilog av funktion
- Keywords:
  - Static (C)
  - Extern (C)
  - Import, Export (assembler)