Assemblerprogrammering, ARM-Cortex M4 – del 2

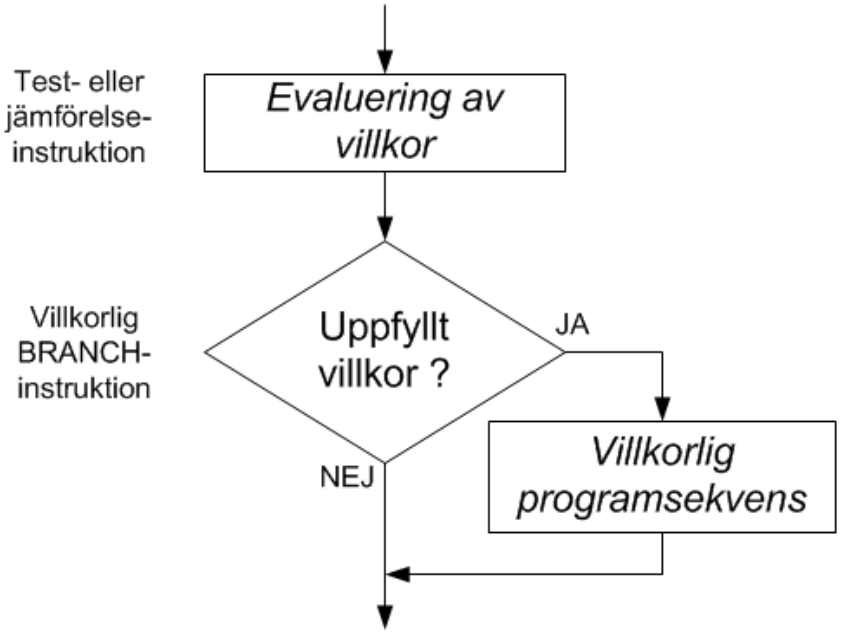
Ur innehållet

- Programflöde
- Subrutiner, parametrar, returvärden och lokala variabler
- Startup-sekvensen
- Läsanvisningar:
  - Arbetsbok kap 2
  - Quick-guide, instruktionslistan

Kvar från föregående föreläsning:

- Programflöde
- Enkla subrutiner

Villkorsblock – kan ändra programflöde



EXEMPEL:

```
if( a==b )
   L1;
L2;
```

Assemblerspråk:

```
    LDR   R1,a
    LDR   R2,b
    CMP   R1,R2
    BEQ   L1
    B     L2
L1:
    ...
L2:
```

Instruktioner för villkorlig programflödeskontroll

C-operator	Betydelse	Datatyp	Instruktion
==	Lika med	signed/unsigned	BEQ
!=	Skild från	signed/unsigned	BNE
<	Mindre än	signed	BLT
		unsigned	BCS
<=	Mindre än eller lika	signed	BLE
		unsigned	BLS
>	Större än	signed	BGT
		unsigned	BHI
>=	Större än eller lika	signed	BGE
		unsigned	BCC

Instruktioner för programflödesändring

C-operator	Betydelse	Operation	Instruktion	RTN
`goto`	ovillkorlig	Branch	`B <label>`	`PC←label`
	ovillkorlig	Branch and exchange	`BX Rx`	`PC←Rx`

Flaggsättning

Instruktioner för aritmetik- logik- och skiftoperationer påverkar flaggsättningen:

```
ADD     SUB     MUL ...
AND     ORR     EOR ...
ASR     LSL     ROR ...
```

Se Quick Guide...

EXEMPEL

Antag följande deklarationer på ”toppnivå”:

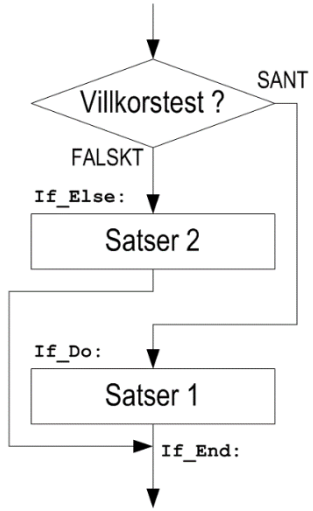
```
unsigned short    a,b;
signed short      c,d;
```

Koda följande programsekvens i assembler:

```
if( a < b )
{
   if ( c < d )
      L1;
}
L2;
```

Vi löser på tavlan...

If (...) {...} else { ...}

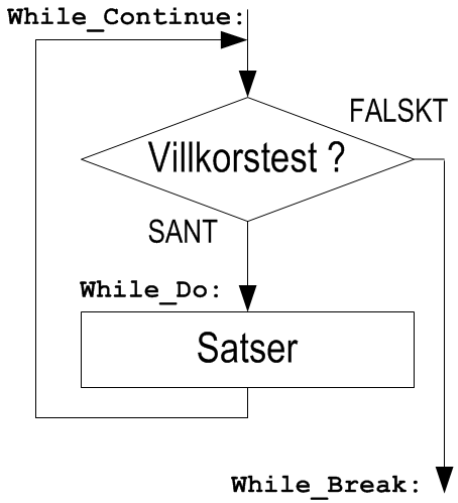



```
int   a,b,c;
if ( a > b )
   c = a;
else
   c = b;
```

```
        LDR    R1,a
        LDR    R2,b
        CMP    R1,R2
        BGT    If_Do
If_Else:
        MOV    R0,R2
        B      If_End
If_Do:
        MOV    R0,R1
If_End:
        ...
```

>	Större än	signed	BGT
		unsigned	BHI

while (...) {...}



Vid kodning av ”while”-iteration används det *komplementära* villkoret

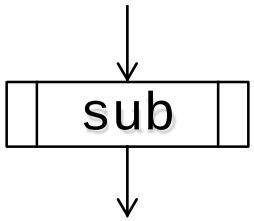
```
While_Continue:
        LDR    R1,a
        CMP    R1,#20
        BGE    While_Break
While_Do:
        ...
B       While_Continue
While_Break:
```

```
while ( a < 20 )
{
   ...
}
```

>=	Större än eller lika	signed	BGE
		unsigned	BCC

Subrutiner (”funktioner”)

C-operator	Betydelse	Operation	Instruktion	RTN
`f()`	funktionsanrop	Branch and link	`BL <label>`	`LR←PC, PC←label`
`“return”`		Branch and exchange	`BX Rx`	`PC←Rx`



C:

```
...
   sub ();
...
```

EXEMPEL:

```
   BL   sub
@ returadress -> LR
   ...

sub:
   ...
   BX   LR
```

Returadressen sparas i register. Snabbt, men klarar ej rekursion

Subrutiner (”funktioner”) ett exempel

```
unsigned int g = 0x55555555;


void f(){
  g = ~g;
}


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

Vi löser ihop på tavlan...

Subrutiner (”funktioner”) ett exempel: lösningsförslag

```
start:
      bl  f

done:  b done

f:
      ldr r3,=g
      ldr r4,[r3]
      mvn r4,r4
      str r4,[r3]
      bx lr

      .align
g:    .word 0x55555555
```

Subrutiner, parametrar, returvärden och lokala variabler

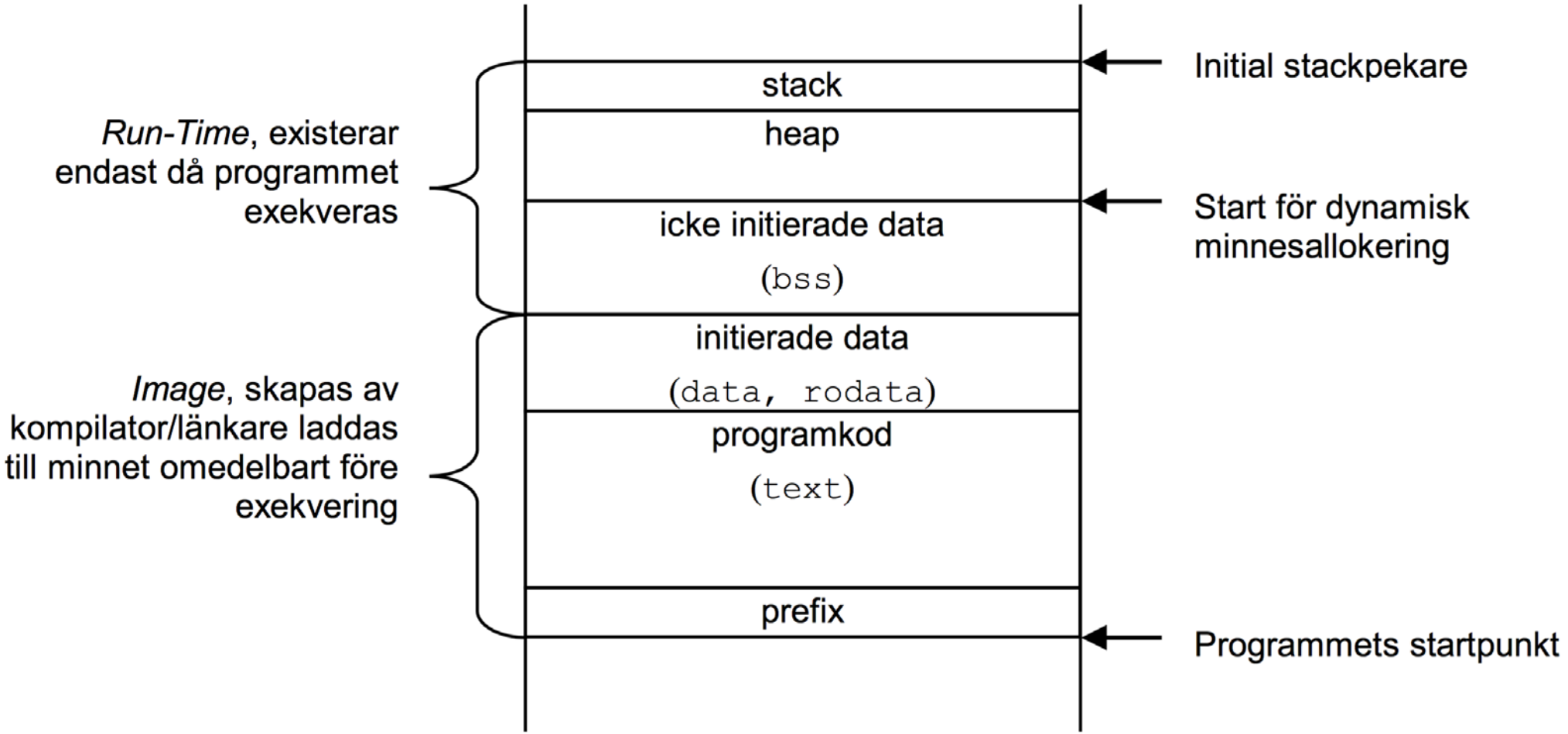
Konventioner:

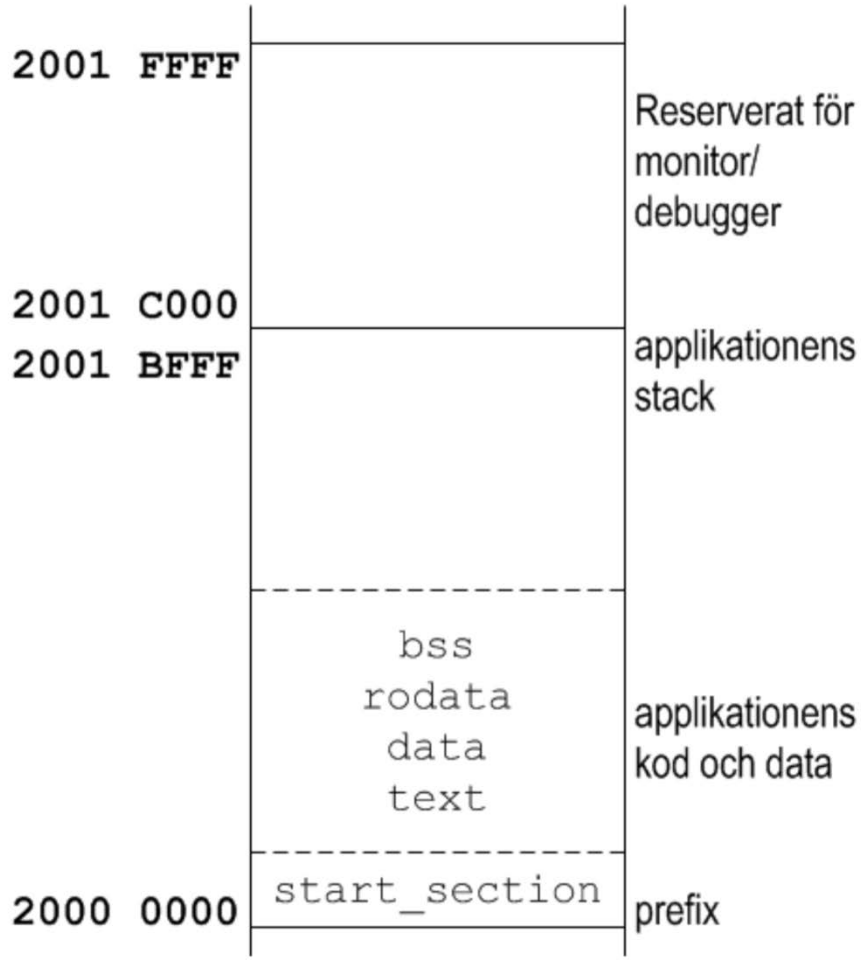
Register	Användning	
R15 (PC)	Programräknare	
R14 (LR)	Länkregister	
R13 (SP)	Stackpekare	
R12 (IP)	Dessa register är avsedda för variabler och som temporära register. Om dom används måste dom sparas och återställas av den anropade (*callee*) funktionen	
R11		
R10		
R9		
R8		
R7	Speciellt använder GCC R7 som pekare till aktiveringspost (*stack frame*) Också dessa register är avsedda för variabler och temporärbruk Om dom används måste dom sparas och återställas av den anropade (*callee*) funktionen	
R6		
R5		
R4		
R3	parameter 4 / temporärregister	Dessa register sparas normalt sett inte över funktionsanrop men om, så är det den anropande (*caller*) funktionens uppgift
R2	parameter 3 / temporärregister	
R1	parameter 2 / resultat 2 /temporärregister	
R0	parameter 1 / resultat 1/ temporärregister	

Subrutiner: Learning by doing

- Maximum
- Manhattan distance
- String length
- Power

Vi löser så många vi hinner ihop...


stack
heap
icke initierade data
(bss)
initierade data
(data, rodata)
programkod
(text)
prefix
Initial stackpekare
Start för dynamisk minnesallokering
Programmets startpunkt
Run-Time, existerar endast då programmet exekveras
Image, skapas av kompilator/länkare laddas till minnet omedelbart före exekvering


2001 FFFF
Reserverat för monitor/ debugger
2001 C000
2001 BFFF
applikationens stack
bss
rodata
data
text
applikationens kod och data
start_section
2000 0000
prefix

```
/*
   Default linker script for MD407 (STM32F407)
   All code and data goes to RAM.
*/

/* Memory Spaces Definitions */
MEMORY
{
  RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 112K
}

SECTIONS
{
   .text :
   {
      . = ALIGN(4);
      *(.start_section) /* startup code */
      *(.text)          /* remaining code */
      *(.text.*)
      *(.data)          /* initialised data */
      *(.data.*)
      *(.rodata)        /* read-only data (constants) */
      *(.rodata*)
      *(.bss)           /* unintialised data */
      *(COMMON)
      . = ALIGN(4);
   } >RAM
}
```

Startup sekvensen

```
startup:
        ldr r0,=0x2001C000
        mov sp,r0
        bl  main
L1:     b  L1
```

Startup sekvensen

```
startup:
        ldr r0,=0x2001C000
        mov sp,r0
        bl  main
L1:     b  L1


main: …
```

GCC Inline Assembler

```
void startup(void){
  asm volatile (
    " ldr r0,=0x2001C000\n"
    " mov sp,r0\n"
    " bl main\n"
    ".L1: b .L1"
    );


int main(int argc, char **argv) {
  …
}
```