

# Maskinorienterad programmering

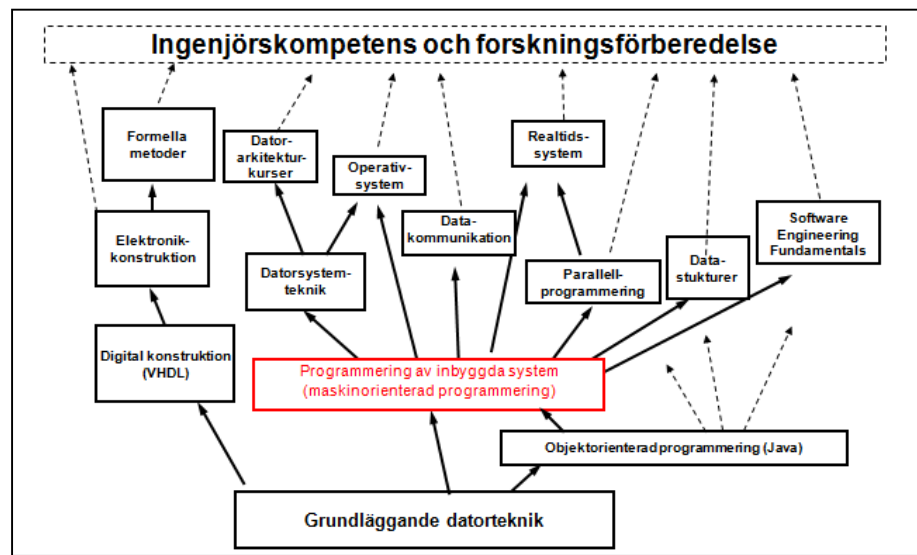
## Sammanfattning

### Ur innehållet:

- Vi rekapitulerar kursens syften
- Vi repeterar kursens "lärandemål"
- Vi belyser hur den skriftliga delen av examinationen genomförs.

# Kursens syften är

- ❑ att vara en introduktion till konstruktion av små inbyggda system och
- ❑ att ge en förståelse för hur imperativa styrstrukturer översätts till assembler
- ❑ att ge en förståelse för de svårigheter som uppstår vid programmering av händelsestyrda system med flera indatakällor.



# 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.

## Funktioners parametrar och returvärden.

**Permanenta och tillfälliga variabler**

Exempel:

```
char gc;

void sub( void )
{
    char lc;
    lc = 5;
}
```

→

```
.comm gc,1,1

sub:
    PUSH {R7,LR}
    SUB SP,SP,#8
    MOV R7,SP
    ADD R3,R7,#7 @ R3 = &lc
    MOV R2,#5
    STRB R2,[R3]
    MOV SP,R7
    ADD SP,SP,#8
    POP {R7,PC}
```

I stället för "registerallokering" av **lc** kan man reservera en position på stacken:  
I subrutinen refereras här då variabeln **lc** som **[x7,#7]**.

## Parameteröverföring till, och returvärden från subrutiner

### Parametrar, två metoder kombineras

Via register: snabbt, effektivt och enkelt, begränsat antal  
Via stacken: generellt

### Returvärden

Via register: för enkla datatyper som ryms i processorns register R0 och ev. R1  
Via stacken: sammansatta datatyper (poster och fält)

| Register | Användning                                                                          |
|----------|-------------------------------------------------------------------------------------|
| R15 (PC) | Programräknare                                                                      |
| R14 (LR) | Länkregister                                                                        |
| R13 (SP) | Stackpekare                                                                         |
| R12 (IP) |                                                                                     |
| R11      | Dessa register är avsedda för variabler och som temporära register.                 |
| R10      | Om den används måste den sparas och återställas av den anropade (callee) funktionen |
| R9       |                                                                                     |
| R8       | Speciellt anpassad GCC R7 som pekare till aktiveringspost (stack frame)             |
| R7       | Ökade dessa register är avsedda för variabler och temporärdata                      |
| R6       | Om den används måste den sparas och återställas av den anropade (callee) funktionen |
| R5       |                                                                                     |
| R4       | parameter 4 temporärregister                                                        |
| R3       | parameter 3 temporärregister                                                        |
| R2       | parameter 2 resultat 2 temporärregister                                             |
| R1       | parameter 1 resultat 1 temporärregister                                             |
| R0       | funktionens utgång                                                                  |

Detaljerna styrs av konventioner "application binary interface" (ABI)  
"Procedure call standard"

## Lagringsklasser och synlighet.

**Extern**

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

```
// main.c
#include "fractalmountain.h"
...

POBJECT objects[] = {&player, &fmountain};
unsigned int nObjects = 2;

void main()
{
    ...
}
```

```
// fractalmountain.h
#ifndef FRACALMOUNTAIN_H
#define FRACALMOUNTAIN_H

#include "object.h"
#include "types.h"

void someFunc1();
void someFunc2();
extern OBJECT fmountain;

#endif //FRACALMOUNTAIN_H
```

```
// fractalmountain.c
#include "fractalmountain.h"

OBJECT fmountain = {
    0, // geometri - ingen
    0, // riktningvektor
    0, // initial startposition
    drawMountain, // draw method
    0, // clear_object - unused
    moveMountain, // move method
    set_object_speed // set-speed method
};
```

**extern** = declare without defining  
Utan **extern** skulle vi skapa en **ny** variabel.

Poäng: Om ni kodar ett större program för lab5 så vill ni nog inte ha alla globala objekt/variabler deklarerade i main-filen. Då måste ni använda extern.

## Lagringsklasser och synlighet

### Variabler med permanent adress i minnet:

```
int gi; /* global synlighet */

static int si; /* synlig i denna kalltext */

void sub(void) {
    static int si; /* synlig i funktionen sub */
}
```

### I assemblerspråk:

```
.GLOBAL gi
gi: .SPACE 4
.si: .SPACE 4
.sub_si: .SPACE 4

alternativt sätt för global variabel:
.comm gi,4,4
(.comm sym,length,alignment)
```

Symboler med begränsad synlighet tilldelas *unika* namn av kompilatorn.  
Sådana kompilatorgenererade symbolnamn dyker ofta upp med inledande '.' i assemblerkälltexten.  
Samma C-symboler kan därför användas i olika sammanhang utan konflikt



- *beskriva och tillämpa olika metoder för parameteröverföring till och från subrutiner.*

Registeranvändning

I princip kan man använda de generella registren som man vill men det är mycket bättre att följa ARM's rekommendationer:

| Register | Användning                                                                                                                                              |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| R15 (PC) | Programräknare                                                                                                                                          |
| R14 (LR) | Länkregister                                                                                                                                            |
| R13 (SP) | Stackpekare                                                                                                                                             |
| R12 (IP) |                                                                                                                                                         |
| R11      | Dessa register är avsedda för variabler och som temporära register. Om dom används måste dom sparas och återställas av den anropade (callee) funktionen |
| R10      |                                                                                                                                                         |
| R9       |                                                                                                                                                         |
| R8       |                                                                                                                                                         |
| R7       | Speciellt använder GCC R7 som pekare till aktiveringspost (stack frame)                                                                                 |
| R6       | Också dessa register är avsedda för variabler och temporärbruk                                                                                          |
| R5       | Om dom används måste dom sparas och återställas av den anropade (callee) funktionen                                                                     |
| R4       |                                                                                                                                                         |
| R3       | parameter 4 / temporärregister                                                                                                                          |
| R2       | parameter 3 / temporärregister                                                                                                                          |
| R1       | parameter 2 / resultat 2 / temporärregister                                                                                                             |
| R0       | parameter 1 / resultat 1 / temporärregister                                                                                                             |

Returvärden via register

| Storlek    | C-typ          | Register |
|------------|----------------|----------|
| 8-32 bitar | char/short/int | R0       |
| 64 bitar   | long long      | R1:R0    |

Returvärden via stack

Krävs typiskt då en subrutin ska returnera en komplett post, eller ett helt fält. Den anropande funktionen ska då först reservera utrymme för returvärdet. En pekare till detta utrymme läggs sedan som första parameter vid anropet:

Funktionsparametrar

Konventionerna säger att vi använder register R0,R1,R2,R3, (i denna ordning) för parametrar som skickas till en subrutin. Övriga parametrar läggs på stacken (behandlas senare i kursen).

**Exempel:**  
Följande deklarationer är gjorda:  
int a,b,c,d;  
Visa hur följande funktionsanrop kodas i assemblerspråk:  
sub(a,b,c,d);

**Assembler:**  
LDR R0,a  
LDR R1,b  
LDR R2,c  
LDR R3,d  
BL sub

Tänkbar komplikation: "Registerspill"

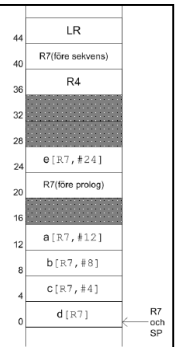
Dvs. register behövs i den anropade subrutinen. Lösning: Skapa "tillfälliga variabler" – använd stacken för temporär lagring

```
sub:
@ parametrar finns i register,
@ spara dessa på stacken
push {r7}
sub sp,sp,#20
add r7,sp,#0
str r0,[r7,#12]
str r1,[r7,#8]
str r2,[r7,#4]
str r3,[r7]
...
```

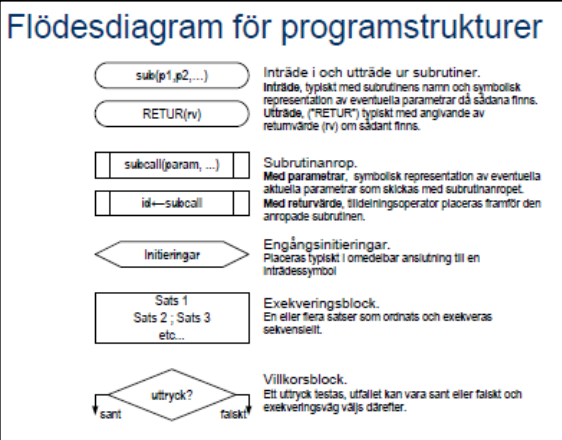
Stackens utseende i sub "aktiveringspost"

Funktionen:  
void sub(int a,int b,int c,int d,int e);

Aktiveringsposten kan sedan utvidgas ytterligare med lokala variabler.



- *beskriva och använda olika kontrollstrukturer.*



### Subrutiner ("funktioner")

| C-operator | Betydelse      | Operation           | Instruktion | RTN                |
|------------|----------------|---------------------|-------------|--------------------|
| f()        | funktionsanrop | Branch and link     | BL <label>  | LR←PC,<br>PC=label |
| "return"   |                | Branch and exchange | BX Rx       | PC←Rx              |

**EXEMPEL :**

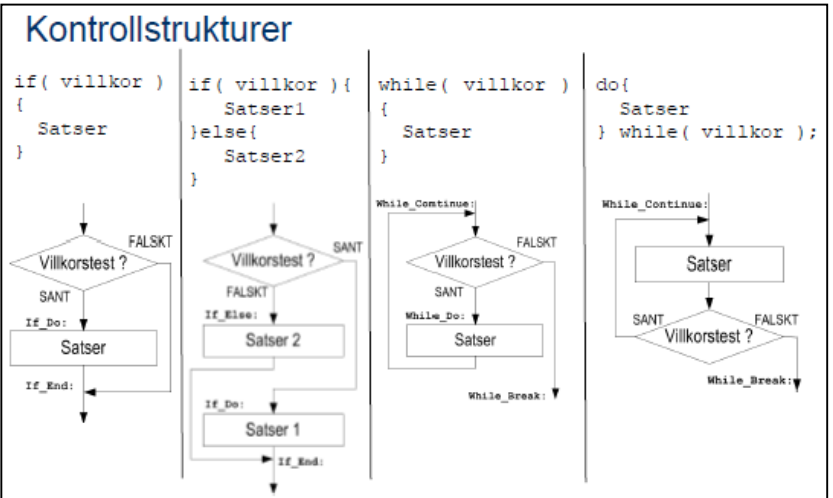
```
BL sub
@ returadress -> LR
...

sub:
...
BX LR
```

**C:**

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

Returadressen sparas i register. Snabbt, men klarar ej rekursion



### Instruktioner för villkorlig programflödeskontroll

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



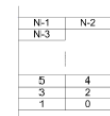
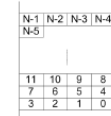
- *beskriva och använda sammansatta datatyper (fält och poster) och enkla datatyper (naturliga tal, heltal och flyttal).*

### Ordlängder och heltalstyper, ANSI C

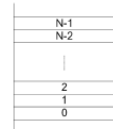
| Typ                | Förklaring                                                                                                                                                                                                                                 |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| char               | Minsta adresserbara enhet som kan rymma en basal teckenuppsättning. Det är dock en heltalstyp som kan vara signed eller unsigned, detta är implementationsberoende.                                                                        |
| signed char        | Tolkas som heltal med tecken. Måste <u>minst</u> kunna representera talområdet [-127, +127], dvs. minst 8 bitar.                                                                                                                           |
| unsigned char      | Tolkas som heltal utan tecken. Måste <u>minst</u> kunna representera talområdet [0, 255], dvs. minst 8 bitar.                                                                                                                              |
| short              | Kort heltalstyp med tecken. Måste <u>minst</u> kunna representera talområdet [-32767, +32767], dvs. minst 16 bitar.                                                                                                                        |
| short int          | Observera att talområdet [-32768, +32767] som fås vid 2-komplementrepresentation, också är tillåtet.                                                                                                                                       |
| signed short       |                                                                                                                                                                                                                                            |
| signed short int   |                                                                                                                                                                                                                                            |
| unsigned short     | Kort heltalstyp utan tecken. Måste <u>minst</u> kunna representera talområdet [0, +65535], dvs. minst 16 bitar.                                                                                                                            |
| unsigned short int |                                                                                                                                                                                                                                            |
| long               | Lång heltalstyp med tecken. Måste <u>minst</u> kunna representera talområdet [-2147483647, +2147483647], dvs. minst 32 bitar.                                                                                                              |
| long int           |                                                                                                                                                                                                                                            |
| signed long        |                                                                                                                                                                                                                                            |
| signed long int    |                                                                                                                                                                                                                                            |
| unsigned long      | Lång heltalstyp utan tecken. Måste <u>minst</u> kunna representera talområdet [0, +4294967295], dvs. minst 32 bitar.                                                                                                                       |
| unsigned long int  |                                                                                                                                                                                                                                            |
| int                | Implementationsbestämd heltalstyp med tecken. Måste <u>minst</u> kunna representera talområdet [-32767, +32767], dvs. minst 16 bitar. Observera att talområdet [-32768, +32767] som fås vid 2-komplementrepresentation, också är tillåtet. |
| signed             | Observera speciellt att int kan vara synonym med short eller long.                                                                                                                                                                         |
| signed int         |                                                                                                                                                                                                                                            |
| unsigned           | Typen int utan tecken.                                                                                                                                                                                                                     |
| unsigned int       |                                                                                                                                                                                                                                            |

### Fält – "vektorer" – N element, indexerats 0..N-1

```
char charvec[N];
sizeof(charvec) =
N bytes
```



```
short shortvec[N];
sizeof(shortvec) =
N*2 bytes
```



```
int intvec[N];
sizeof(intvec) =
N*4 bytes
```

**EXEMPEL:** Vi har deklarationerna:

```
int i, j, *ip, ivec[20];
```

Koda följande tilldelningar i assemblerspråk

a) `i = ivec[j];`

b) `ip = &ivec[j];`

Vi löser på tavlan...

|                        |               |                       |                  |
|------------------------|---------------|-----------------------|------------------|
| .. with register index | [Rx,Ri]       | LDR R0,[R1,R2]        | R0←M(R1+R2)      |
| .. with scaled index   | [Rx,Ri,shift] | LDR R0,[R1,R2,LSL #2] | R0←M(R1+(R2<<2)) |

### Typer för pekare, ARM-M4

```
char *cptr;      /* pekar på 8-bitars datatyp */
short *sptr;     /* pekar på 16-bitars datatyp */
int *iptr;       /* pekar på 32-bitars datatyp */
```

Adressutrymmet är 32 bitar

PC, SP och ALLA pekartyper är 32 bitar

**EXEMPEL:**

```
cptr = ( char *) 0x40021010
```

```
sptr = ( short *) 0x40021010
```

```
iptr = ( int *) 0x40021010
```

Vad är skillnaden?

### Structs (sammansatta datatyper)

I uppgift 39:

```
typedef struct tPoint{
    unsigned char x;
    unsigned char y;
} POINT;

#define MAX_POINTS 20

typedef struct tGeometry{
    int numpoints;
    int sizex;
    int sizey;
    POINT px[ MAX_POINTS ];
} GEOMETRY, *PGEOMETRY;
```

Skapa och initiera en variabel av typen GEOMETRY:

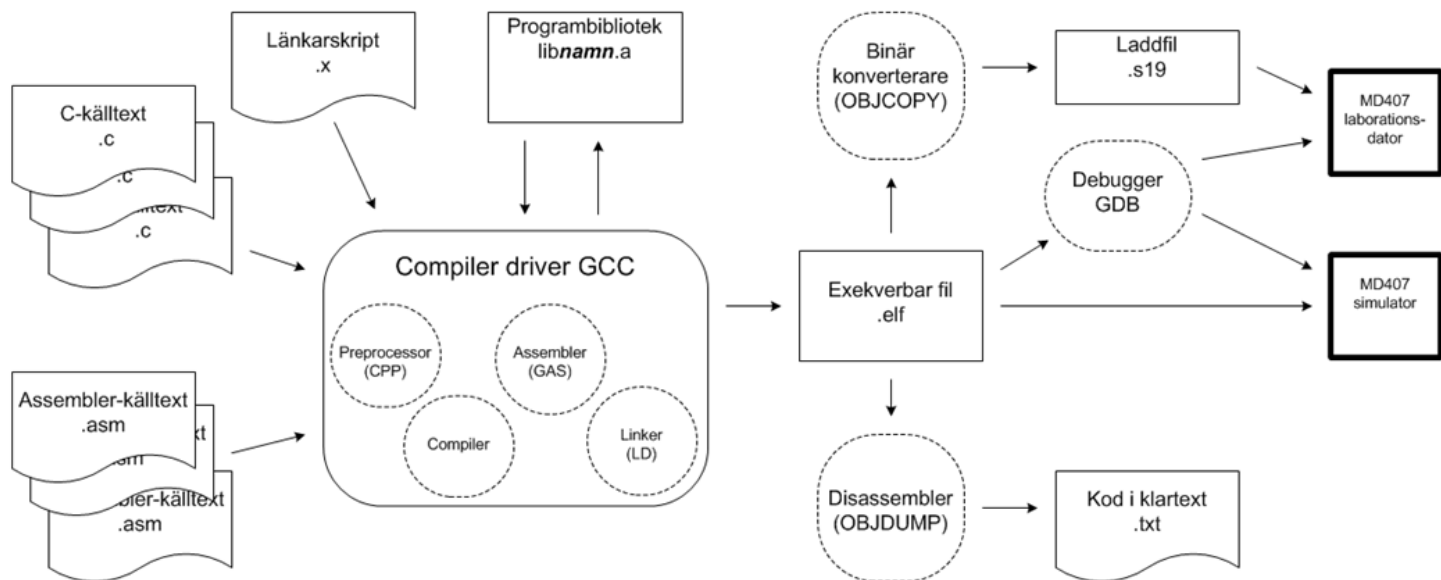
```
GEOMETRY ball_geometry = {
    12, 4, 4,
    { // POINT px[20]
        {0,1}, // px[...]
        {0,2},
        {1,0},
        {1,1},
        {1,2},
        {1,3},
        {2,0},
        {2,1},
        {2,2},
        {2,3},
        {3,1},
        {3,2} // Här utnyttjar vi ofullständig
              // initiering (12 av 20)
    }
};
```

## 2. Programutvecklingsteknik

Att kunna:

- beskriva översättningsprocessen, dvs. assemblatorns arbetssätt, 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 arbetssätt, preprocessorns användning, separatkompilering och länkning.*



### Preprocessor directives - #if, #ifdef, #ifndef

```
Preprocessor conditional inclusion
#define X 1 // syntax: #define [identifier name] [value], där [value] är optional

#if X == 0 // syntax: #if <value>, där 0=false och !0=true
    // any C-code
#elif X-1 == 1 // betyder else if
    // any C-code
#else
    // any C-code
#endif

#if 0 // bra för att temporärt kommentera bort stora block av kod.
    // any C-code
#endif

#define HW_DEBUG
#ifdef HW_DEBUG
    // any C-code, t ex:
    #define SIMULATOR
#endif

void delay_500ns(void)
{
    #ifndef SIMULATOR
        delay_250ns();
        delay_250ns();
    #endif
}
```

### Include guards

|                                                                                                                                                                                       |                                                                                                                                                                                                                                            |                                                                                                                             |                                                                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <pre>// main.c #include "player.h" #include "enemies.h"  void main() {     // ...     Vec2i add2i(Vec2i a, Vec2i b);     char isEqual(Vec2i a, Vec2i b);     #endif //VECMATH_H</pre> | <pre>// vecmath.h #ifndef VECMATH_H #define VECMATH_H  typedef struct {     union {         int v[2];         struct {int x, y;};     }; } Vec2i;  Vec2i add2i(Vec2i a, Vec2i b); char isEqual(Vec2i a, Vec2i b); #endif //VECMATH_H</pre> | <pre>// player.h #ifndef PLAYER_H #define PLAYER_H  #include "vecmath.h"  void movePlayer(Vec2i v); #endif //PLAYER_H</pre> | <pre>// enemies.h #ifndef ENEMIES_H #define ENEMIES_H  #include "vecmath.h"  void moveEnemy(int i, Vec2i v); #endif //ENEMIES_H</pre> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|

Utan include guards så inkluderas vecmath.h två gånger för main.c. Andra gången kommer kompilatorn klaga på att Vec2i redan är definierad.

.c-filer kompileras var för sig. Preprocessorn och kompilatorn exekveras individuellt per .c-fil.

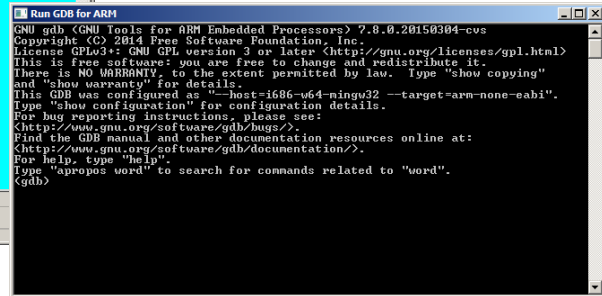
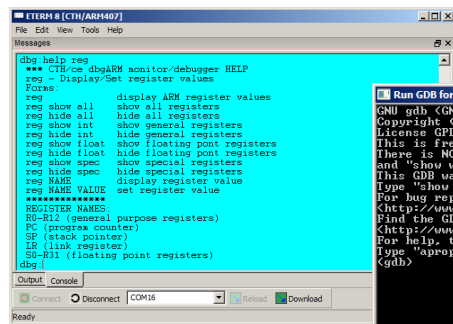
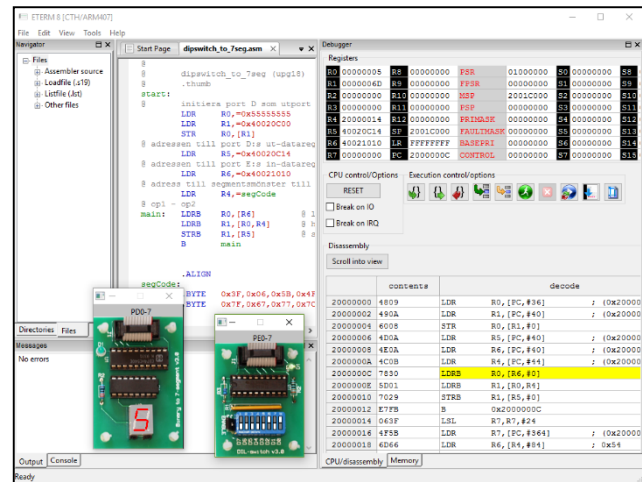
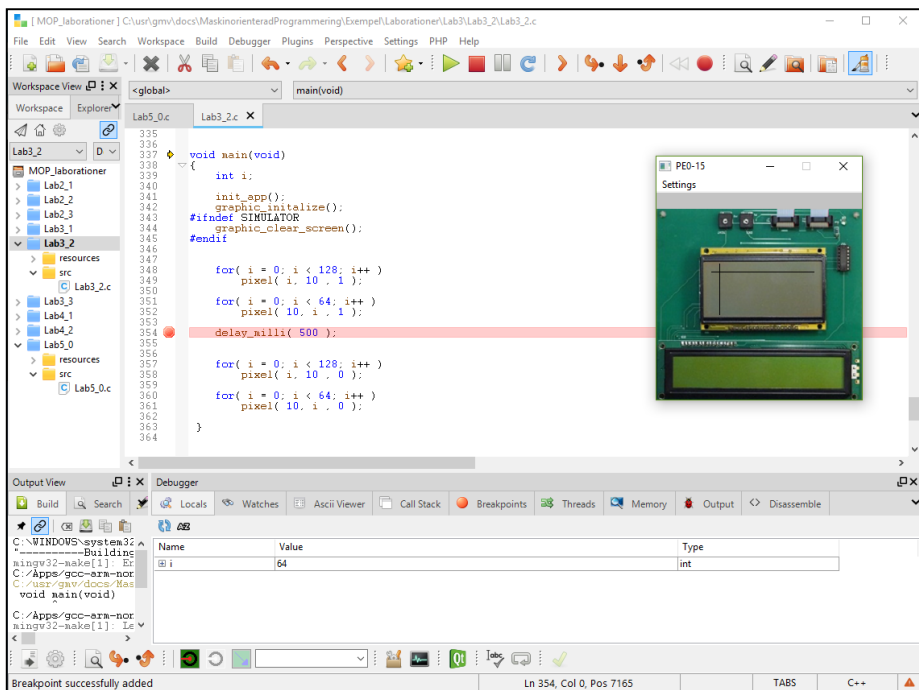
Vi bör ha include-guards på alla .h-filer – även om det i detta exemplet inte behövs för player.h samt enemies.h

### Programstruktur

|                                                                                                                             |                                                     |                                                                                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>// main.c #include &lt;stdio.h&gt; #include "foo.h"  int main() {     printf("x is %i", foo(0));     return 0; }</pre> | <pre>// foo.h int foo(int x);</pre>                 | <pre>// foo.c #include &lt;stdlib.h&gt;  int foo(int x) {     if( x == 0 ){         int x = 4;         return x;     }     return x; }</pre> |
| <b>c-fil</b><br>Inkluderar header-fil                                                                                       | <b>header-fil</b><br>Innehåller funktionsprototyper | <b>c-fil</b>                                                                                                                                 |

header-filen måste inte ha samma namn som c-filen, men det är enklare så.

- 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.*

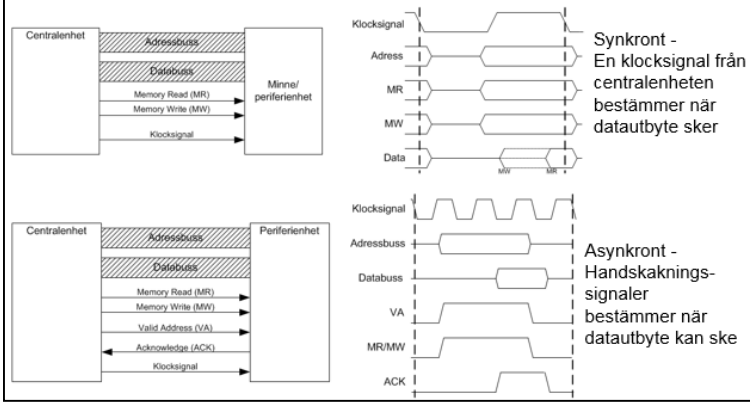
### 3. Systemprogrammerarens bild

Att kunna:

- beskriva och tillämpa olika principer för överföring mellan centralenhet och kringenheter så som: ovillkorlig eller villkorlig överföring, statustest och rundfrågning.
- konstruera program för systemstart och med stöd för avbrottshantering från olika typer av kringenheter.
- beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart samt prioritetshantering vid undantag.
- 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, statustest och rundfrågning.*

### Synkrona och asynkrona gränssnitt



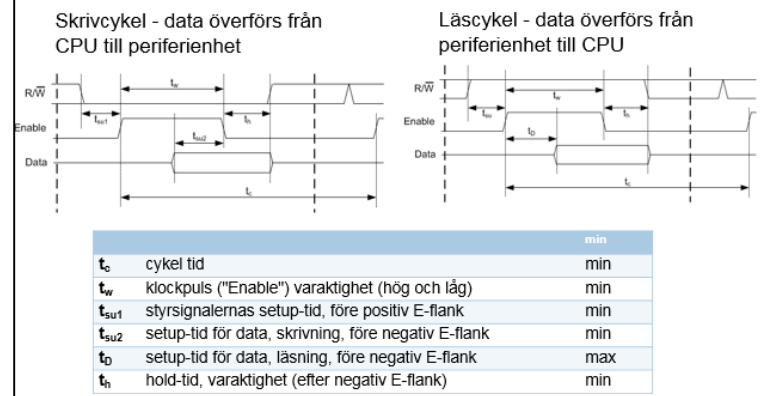
### Ovillkorlig överföring

Kräver samtidighet "synkron" - en speciell signal, "Enable", används då för att ange exakt när datautbyte ska ske.



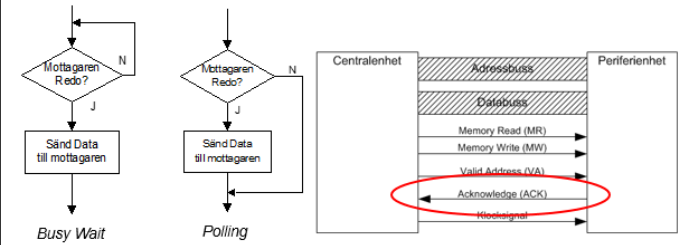
Flankerna definierar samtidigheten. Signalen kallas ofta "klocksignal".

### Tidsdiagram - underlag för synkronisering



### Villkorlig överföring

Periferienheten har ett register med en statusbit som anger om data finns eller ej.



Förutsätter inte samtidighet men kräver speciella "handskakningssignaler". Ett sådant gränssnitt kallar vi därför "asynkront".



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

Vektortabellen

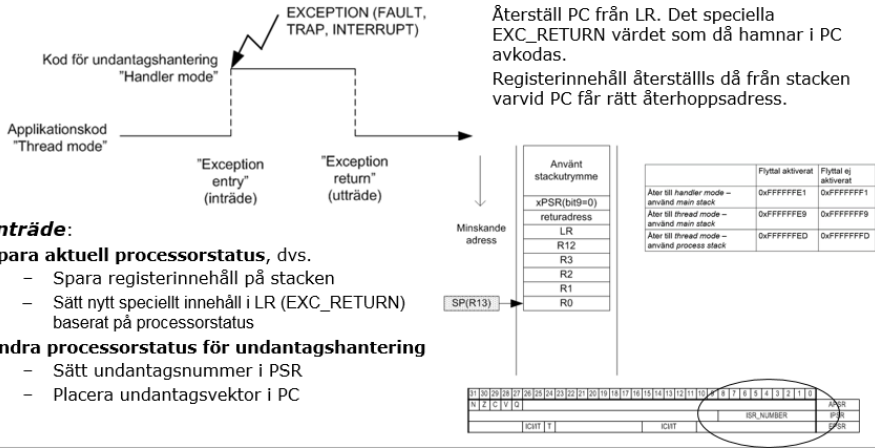
Anger position för de olika undantagsvektorer.

De 16 första positionerna är samma för alla Cortex-M4.

Resten av tabellen är specifik för den aktuella microcontrollern.

|    |          |               |                                                                                          |                           |
|----|----------|---------------|------------------------------------------------------------------------------------------|---------------------------|
|    |          |               | Reserverad stack                                                                         | 0x0000 0000               |
| -3 | fixed    | Reset         | Reset                                                                                    | 0x0000 0004               |
| -2 | fixed    | NMI           | Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector. | 0x0000 0008               |
|    |          |               | All class of fault                                                                       | 0x0000 000C               |
| -1 | fixed    | HardFault     | HardFault                                                                                | 0x0000 0010               |
| 0  | settable | MemManage     | Memory management                                                                        | 0x0000 0014               |
| 1  | settable | BusFault      | Pre-fetch fault, memory access fault                                                     | 0x0000 0018               |
| 2  | settable | UsageFault    | Undefined instruction or illegal state                                                   | 0x0000 001C               |
|    |          |               | Reserved                                                                                 | 0x0000 0020 - 0x0000 002B |
| 3  | settable | SVCall        | System service call via SWI instruction                                                  | 0x0000 002C               |
| 4  | settable | Debug Monitor | Debug Monitor                                                                            | 0x0000 0030               |
|    |          |               | Reserved                                                                                 | 0x0000 0034               |
| 5  | settable | PendSV        | Pendable request for system service                                                      | 0x0000 0038               |
| 6  | settable | SysTick       | System tick timer                                                                        | 0x0000 003C               |
| 7  | settable | WWDG          | Window Watchdog interrupt                                                                | 0x0000 0040               |
| 8  | settable | PVD           | PVD through EXTI line detection interrupt                                                | 0x0000 0044               |
| 9  | settable | TAMP_STAMP    | Tamper and TimeStamp interrupts through the EXTI line                                    | 0x0000 0048               |
| 10 | settable | RTC_WKUP      | RTC Wakeup interrupt through the EXTI line                                               | 0x0000 004C               |
| 11 | settable | FLASH         | Flash global interrupt                                                                   | 0x0000 0050               |
| 12 | settable | RCC           | RCC global interrupt                                                                     | 0x0000 0054               |
| 13 | settable | EXTI0         | EXTI Line0 interrupt                                                                     | 0x0000 0058               |
| 14 | settable | EXTI1         | EXTI Line1 interrupt                                                                     | 0x0000 005C               |
| 15 | settable | EXTI2         | EXTI Line2 interrupt                                                                     | 0x0000 0060               |
| 16 | settable | EXTI3         | EXTI Line3 interrupt                                                                     | 0x0000 0064               |
| 17 | settable | EXTI4         | EXTI Line4 interrupt                                                                     | 0x0000 0068               |
|    |          |               |                                                                                          |                           |
|    |          |               |                                                                                          |                           |
| 81 | 88       | FPU           | FPU global interrupt                                                                     | 0x0000 0184               |

Undantagshantering - detaljerna



Inträde:

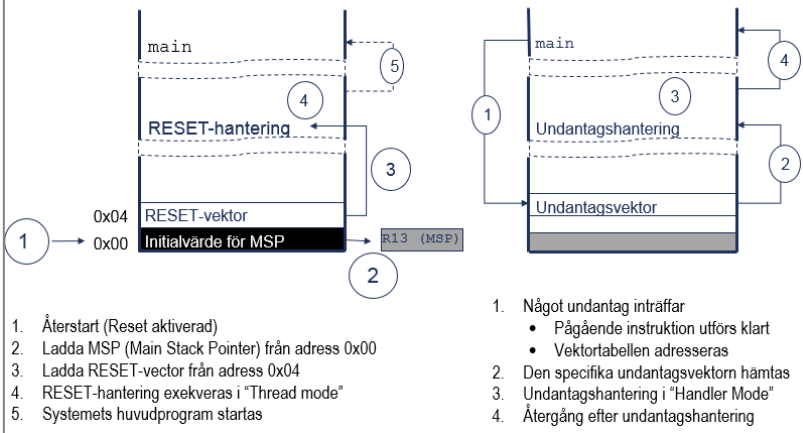
Spara aktuell processorstatus, dvs.

- Spara registerinnehåll på stacken
- Sätt nytt speciellt innehåll i LR (EXC\_RETURN) baserat på processorstatus

Ändra processorstatus för undantagshantering

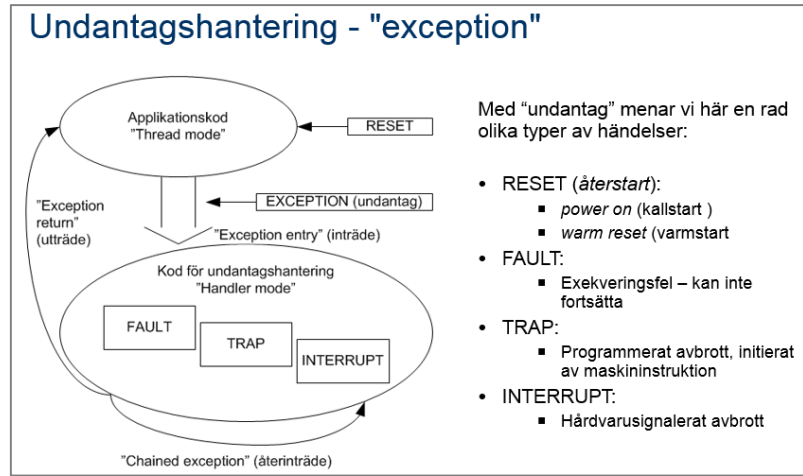
- Sätt undantagsnummer i PSR
- Placera undantagsvektor i PC

Exekvering vid undantag



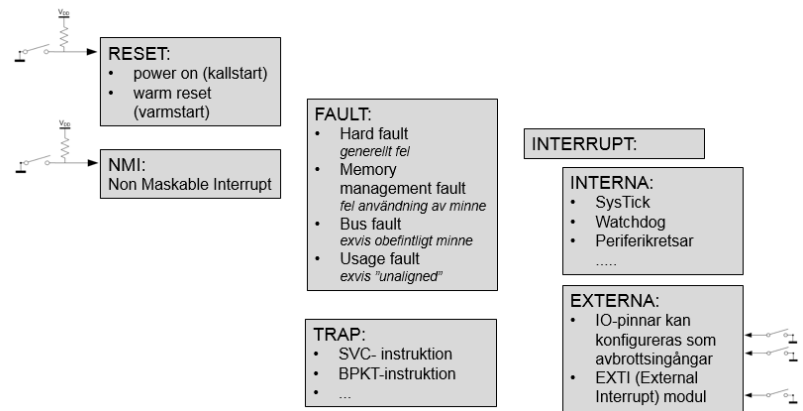


- *beskriva och exemplifiera olika undantagstyper: interna undantag, avbrott och återstart.*



### Undantagstyper

De olika undantagstyperna har en naturlig, fallande prioritet



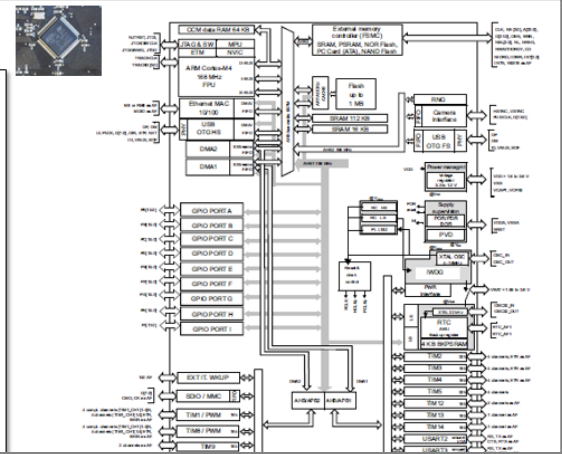
### Interna avbrott

**STM32F405xx**  
**STM32F407xx**

ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+1KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera

**Features**

- Core: ARM 32-bit Cortex-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from flash memory, independent up to 168 MHz, 1.25 DMIPS/MHz (Dyngstone 2.1), and DSP instructions
- Memories
  - Up to 1 Mbyte of Flash memory
  - Up to 192+4 Kbytes of SRAM including 64-Kbytes of DCM (core coupled memory) data RAM
- Flexible static memory controller supporting Compact Flash, SRAM, PSRAM, NOR and NAND memories
- LCD parallel interface, 6800/6800 modes
- Clock, reset and supply management
  - 1.8 V to 3.6 V application supply and I/Os
  - POR, PDR, PVD and BOR
  - 4-to-24 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - 32 kHz oscillator for RTC with calibration
- I/OCPWM or pulse counter and quadrature (incremental) encoder input
- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
  - Cortex-M4 Embedded Trace Macrocell™
- Up to 145 I/O ports with interrupt capability
  - Up to 136 fast I/Os up to 84 MHz
  - Up to 138 I/Os Universal I/Os
- Up to 15 communication interfaces
  - Up to 3 x PC interfaces (SMBus/PMBus)
  - Up to 4 USART/UARTs (10.5 Mbits, ISO 7816 interface, LIN, IrDA, modem control)
  - Up to 3 SPIs (45 Mbits), 2 with master full-duplex PS to achieve audio class accuracy via internal audio PLL or external clock
  - 2 x CAN interfaces (2 DB Active)
  - SIO interface
- Advanced connectivity
  - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
  - USB 2.0 high-speed full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and LPU
  - 10/100 Ethernet MAC with dedicated DMA, supports IEEE 1588 hardware, MII/RMII
  - 5- to 14-bit parallel camera interface up to 64 Mbytes
  - True random number generator
  - CRC calculation unit

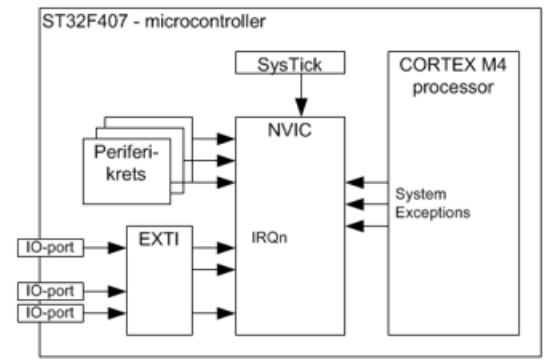


### Externa avbrott

Externa avbrott är en form av undantag.

Registerinnehåll (xPSR, PC, LR, R12, R3-R0) sparas och återställs automatiskt av processorn

Avbrotts hanterare kan skrivas uteslutande med 'C'-kod.

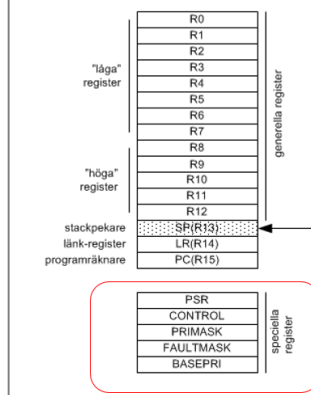


|    |          |          |                   |                                                       |
|----|----------|----------|-------------------|-------------------------------------------------------|
| 6  | settable | SysTick  | System tick timer |                                                       |
| 0  | 7        | settable | WWDG              | Window Watchdog interrupt                             |
| 1  | 8        | settable | PVD               | PVD through EXTI line detection interrupt             |
| 2  | 9        | settable | TAMP_STAMP        | Tamper and TimeStamp interrupts through the EXTI line |
| 3  | 10       | settable | RTC_WKUP          | RTC Wakeup interrupt through the EXTI line            |
| 4  | 11       | settable | FLASH             | Flash global interrupt                                |
| 5  | 12       | settable | RCC               | RCC global interrupt                                  |
| 6  | 13       | settable | EXTI0             | EXTI Line0 interrupt                                  |
| 7  | 14       | settable | EXTI1             | EXTI Line1 interrupt                                  |
| 8  | 15       | settable | EXTI2             | EXTI Line2 interrupt                                  |
| 9  | 16       | settable | EXTI3             | EXTI Line3 interrupt                                  |
| 10 | 17       | settable | EXTI4             | EXTI Line4 interrupt                                  |
| 11 | 18       | settable | DMA1_Stream0      | DMA1 Stream0 global interrupt                         |
| 12 | 19       | settable | DMA1_Stream1      | DMA1 Stream1 global interrupt                         |

- beskriva och tillämpa olika metoder för prioritetshantering vid multipla avbrottskällor (mjukvarubaserad och hårdvarubaserad prioritering, avbrottsmaskering, icke-maskerbara avbrott).

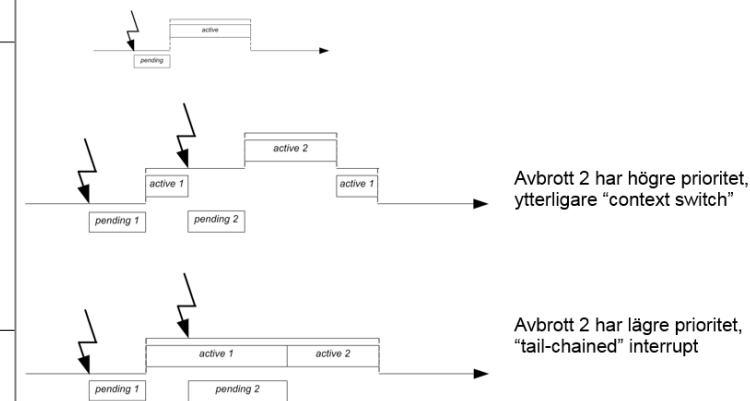
|    |          |               |                                            |
|----|----------|---------------|--------------------------------------------|
| -3 | fixed    | Reset         | Reserverad stack                           |
| -2 | fixed    | NMI           | Reset                                      |
| -1 | fixed    | HardFault     | Non maskable interrupt                     |
| 0  | settable | MemManage     | Clock Security System to the NMI vector.   |
| 1  | settable | BusFault      | Memory management                          |
| 2  | settable | UsageFault    | Pre-fetch fault, memory access             |
| -  | -        | -             | Undefined instruction                      |
| -  | -        | -             | Reserved                                   |
| 3  | settable | SVCall        | System service call via instruction        |
| 4  | settable | Debug Monitor | Debug Monitor                              |
| -  | -        | -             | Reserved                                   |
| 5  | settable | PendSV        | Pendable request for instruction           |
| 6  | settable | SysTick       | System tick timer                          |
| 7  | settable | WWDG          | Window Watchdog interrupt                  |
| 8  | settable | PVD           | PVD through EXTI line interrupt            |
| 9  | settable | TAMP_STAMP    | Tamper and TimeStamp through the EXTI line |
| 10 | settable | RTC_WKUP      | RTC Wakeup interrupt through EXTI line     |
| 11 | settable | FLASH         | Flash global interrupt                     |
| 12 | settable | RCC           | RCC global interrupt                       |
| 13 | settable | EXTI0         | EXTI Line0 interrupt                       |
| 14 | settable | EXTI1         | EXTI Line1 interrupt                       |
| 15 | settable | EXTI2         | EXTI Line2 interrupt                       |
| 16 | settable | EXTI3         | EXTI Line3 interrupt                       |
| 17 | settable | EXTI4         | EXTI Line4 interrupt                       |
| -  | -        | -             | -                                          |
| -  | -        | -             | -                                          |
| -  | -        | -             | -                                          |
| 81 | 88       | FPU           | FPU global interrupt                       |

### Register



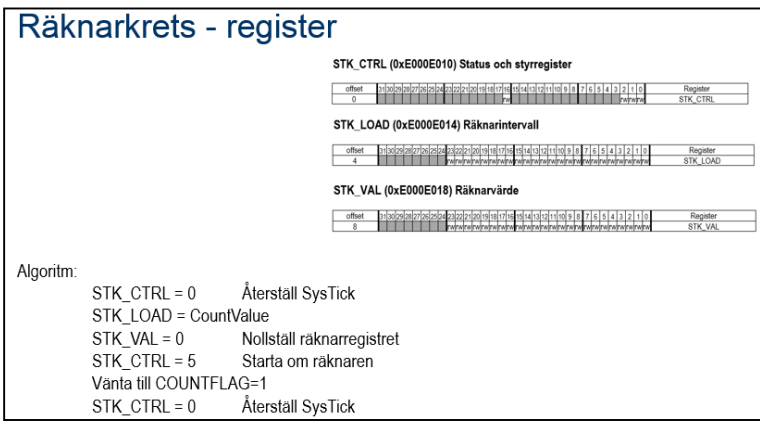
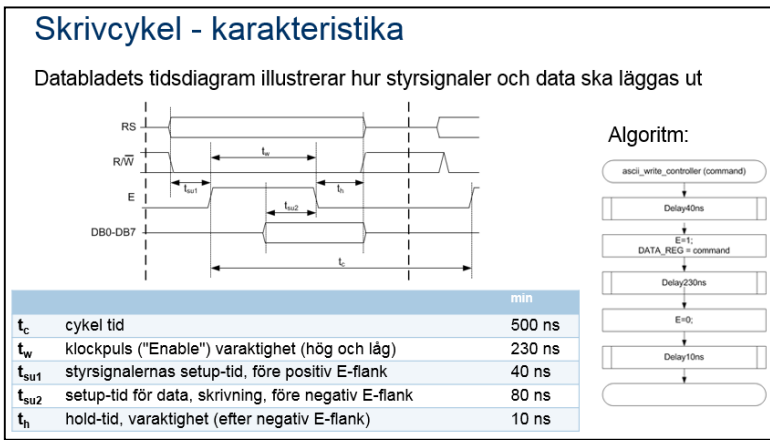
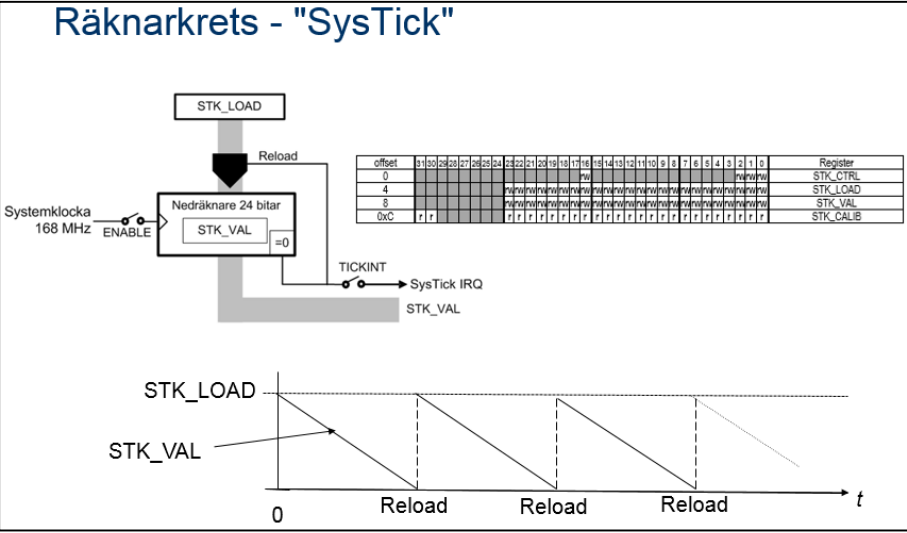
### Avbrottsprioritet

Flera avbrott, pending, active och prioritet



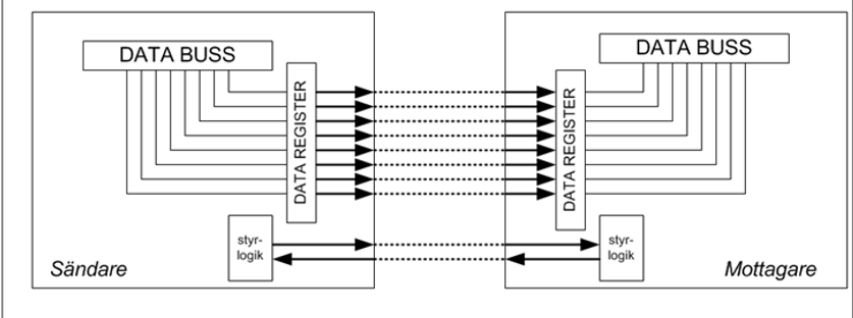
|   |   |   |   |          |   |   |   |   |   |   |   |           |
|---|---|---|---|----------|---|---|---|---|---|---|---|-----------|
| 0 | 9 | 8 | 7 | 6        | 5 | 4 | 3 | 2 | 1 | 0 |   |           |
|   |   |   |   |          |   |   |   |   |   |   | M | PRIMASK   |
|   |   |   |   |          |   |   |   |   |   |   |   |           |
| 0 | 9 | 8 | 7 | 6        | 5 | 4 | 3 | 2 | 1 | 0 |   |           |
|   |   |   |   |          |   |   |   |   |   |   | M | FAULTMASK |
|   |   |   |   |          |   |   |   |   |   |   |   |           |
| 0 | 9 | 8 | 7 | 6        | 5 | 4 | 3 | 2 | 1 | 0 |   |           |
|   |   |   |   | PRIORITY |   |   |   |   |   |   |   | BASEPRI   |

- *beskriva och använda kretsar för tidmätning.*



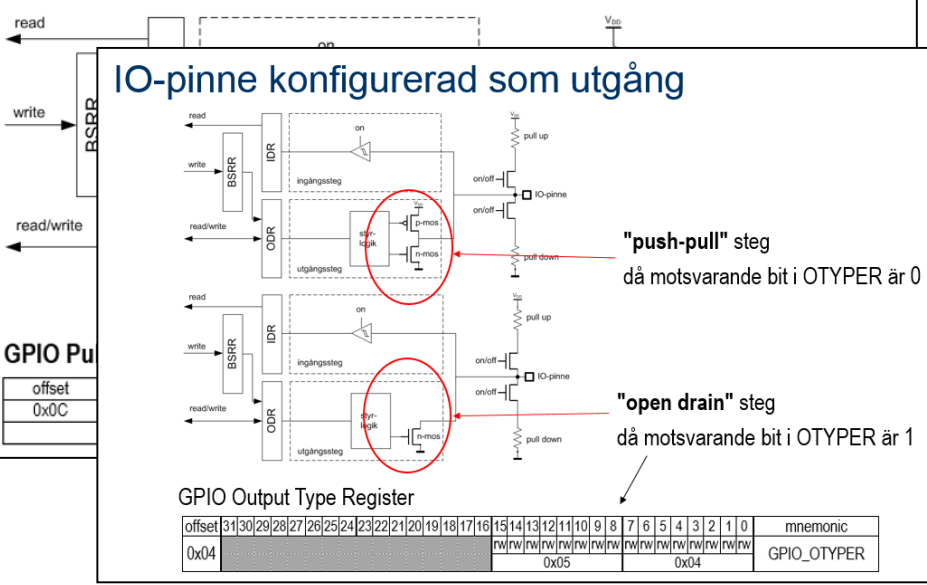
- *beskriva och använda kretsar för parallell respektive seriell överföring.*

Parallell överföring

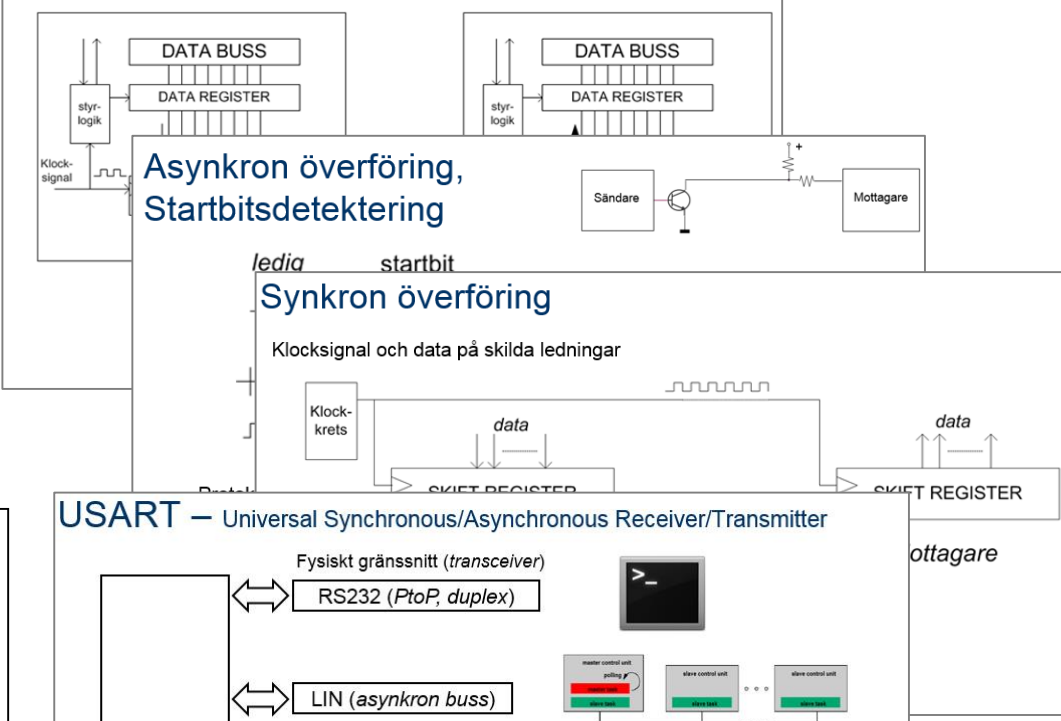


IO-pinne konfigurerad som ingång

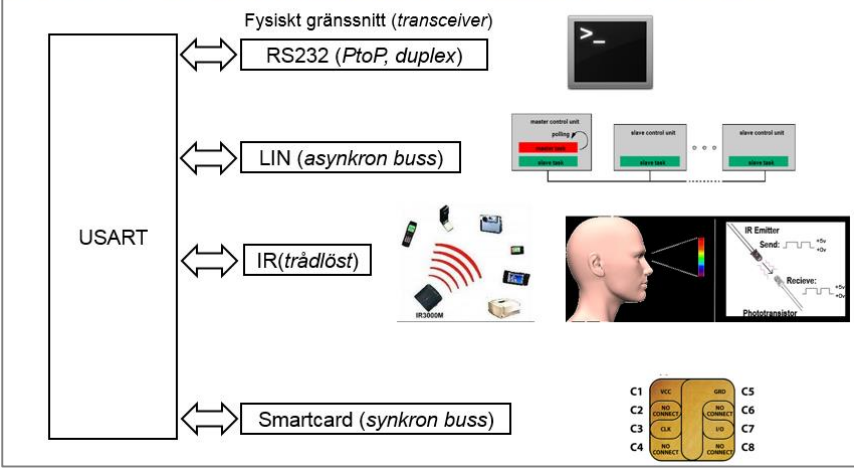
Vi kan programmera "pull-up" eller "pull-down"-funktion, inget externt motstånd behövs.



Seriell överföring



USART – Universal Synchronous/Asynchronous Receiver/Transmitter



# Av speciell vikt: "maskinorienterad programmering..."

- Läsa/skriva på fasta adresser (portar)
- Datatyper, storlek (8,16 eller 32 bitar...)
- Heltalstyper, med eller utan tecken, vad innebär typkonverteringarna?
- Bitoperationer &, |, ^ (AND, OR, XOR)
- Skiftoperationer <<, >> (vänster, höger)

```
asm volatile(  
    " LDR    R0,=0x00005555\n"  
    " LDR    R1,=0x40020C00\n"  
    " STR    R0,[R1]\n"  
    ) ;
```

```
void main(void)  
{  
    unsigned char c;  
    app_init();  
    while(1){  
        c = (unsigned char) *(( unsigned char *) 0x40020C11 );  
        c = (c >> 3) & 7;  
        * ( (unsigned char *) 0x40020C14) = c;  
    }  
}
```



# Kodningskonventioner

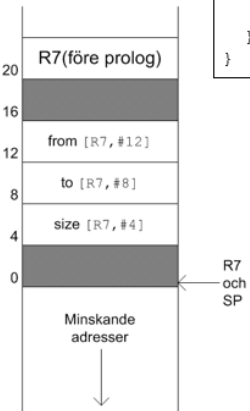
Program som kr ver k lltexter  
b de i 'C' och assemblerspr k...

## Kodoptimering – "f r hand" - exempel

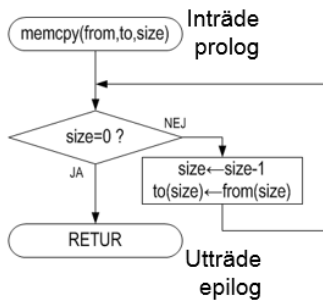
Anropssekvens: (formellt):

```
LDR R0,=from
LDR R1,=to
LDR R2,size
BL memcpy
```

Stackens utseende  
i "memcpy" efter  
prolog



```
void memcpy( unsigned char from[],
             unsigned char to[],
             unsigned int size )
{
    while (size > 0){
        size = size - 1;
        to[size] = from[size];
    }
}
```



## Registeranv ndning

I princip kan man anv nda de generella registren som man vill  
men det  r mycket b ttre att f lja ARM's rekommendationer:

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

## Kodf rb ttring, ingen on dig minnesanv ndning, dvs. registerallokering, "f rb ttrad f r hand"

Registerallokering:  
R0: from, ers tter [R7,#12]  
R1: to , ers tter [R7,#8]  
R2: size , ers tter [R7,#4]  
R3: tempor r  
R4: tempor r

Vi g r dessa substitutioner och  
kommenterar d refter ut de  
instruktioner som d  blir  
 verfl diga...

```
memcpy:
memcpy_prolog:
    PUSH {R7}
    SUB SP,SP,#20
    MOV R7,SP
    STR R0,[R7,#12]
    STR R1,[R7,#8]
    STR R2,[R7,#4]

memcpy_1:
    LDR R3,[R7,#4]
    CMP R3,#0
    BEQ memcpy_epilog

memcpy_2:
    LDR R3,[R7,#4]
    SUB R3,R3,#1
    STR R3,[R7,#4]
    LDR R2,[R7,#8]
    LDR R3,[R7,#4]
    ADD R3,R3,R2
    LDR R1,[R7,#12]
    LDR R2,[R7,#4]
    ADD R2,R2,R1
    LDRB R2,[R2]
    STRB R2,[R3]
    B memcpy_1

memcpy_epilog:
    ADD SP,SP,#20
    POP {R7}
    BX lr
```

```
memcpy:
memcpy_prolog:
    PUSH {R4,LR}
    SUB SP,SP,#20
    MOV R7,SP
    STR R0,[R7,#12]
    STR R1,[R7,#8]
    STR R2,[R7,#4]

memcpy_1:
    LDR R3,[R7,#4]
    CMP R2,#0
    BEQ memcpy_epilog

memcpy_2:
    LDR R3,[R7,#4]
    SUB R3,R3,#1
    STR R3,[R7,#4]
    LDR R2,[R7,#8]
    LDR R3,[R7,#4]
    MOV R3,R0
    ADD R3,R3,R2
    MOV R4,R1
    ADD R4,R4,R2
    ADD R3,R3,R2
    LDR R1,[R7,#12]
    LDR R2,[R7,#4]
    LDR R2,[R7,#4]
    ADD R2,R2,R1
    LDRB R3,[R3]
    STRB R3,[R4]
    B memcpy_1

memcpy_epilog:
    ADD SP,SP,#20
    POP {R4,PC}
    BX lr
```

## Pekare och deras användning, programmeringsuppgifter *eller* kortare frågor.

### Pekare

- Pekarens värde är en adress.
- Pekarens typ berättar hur man tolkar bitarna som finns på adressen.

```
// array av chars  
char str[] = "abc";
```

```
// p pekare till char  
char* p = str; // == &str[0] == &(str[0])
```

```
*p; // värdet p pekar på är 'a'
```

```
*p = 'b';  
MOV R0, #'b'  
LDR R1, p // r1 <- &str  
STRB R0, [R1] - skriv till *p
```

Vi hämtar värdet som ligger på adressen som ligger i p.

```
LDR R0, =str  
LDR R1, =p  
STR R0, [R1]  
  
LDR R0, p  
LDRB R0, [R0] - läs *p
```

7

### Absolutadressering

```
0x40011000 // ett hexadecimalt tal  
(unsigned char*) 0x40011000 // en unsigned char pekare som pekar på adress 0x40011004  
*((unsigned char*) 0x40011000) // dereferens av pekaren  
  
// läser från 0x40011000  
unsigned char value = *((unsigned char*) 0x40011000);  
  
// skriver till 0x40011004  
*((unsigned char*) 0x40011004) = value;
```

```
/*  
memcpy.c  
C-library function "memcpy"  
*/  
#include <string.h>  
void *memcpy(void *dst, void *src, size_t size)  
{  
    char *d, char *s, size_t n;  
    if (size <= 0)  
        return(dst);  
    s = (char *) src;  
    d = (char *) dst;  
    if (s <= d && s + (size - 1) >= d) {  
        /* Overlap, must copy right-to-left */  
        s += size - 1;  
        d += size - 1;  
        for (n = size; n > 0; n--)  
            *d-- = *s--;  
    } else  
        for (n = size; n > 0; n--)  
            *d++ = *s++;  
    return(dst);  
}
```

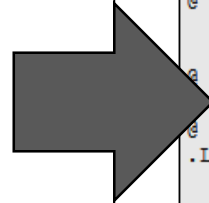
# Assemblerprogrammering...

Att översätta enklare C-kod till assemblerspråk, med kompilatorkonventioner.  
Eller från en annan specifikation, text eller flödesdiagram.

1.7.10. Funktionen `int g( int )` är definierad. Visa hur följande funktion kan kodas i assemblerspråk:

```
int f( int val )
{
    int i;
    int bits = 0;

    for( i=0 ; i< val; i++ )
    {
        bits = bits | g(i);
    }
    return bits;
}
```



```
@ Registerallokering:
@ R0, parameter och returvärde
@ R4 "i"
@ R5 "bits"
@ R6 "val" spill från R0
f:
PUSH                {R4,R5,R6,LR}
@ 'prolog'
    MOV             R6,R0    @ spillregister R6
    MOV             R5,#0    @ bits = 0;
@ 'uttryck 1'
    MOV             R4,#0    @ i = 0;
@ 'For_continue' - 'uttryck 2'
.L1:
    CMP             R4,R6    @ i - val
    BGE             .L2      @ i < val, komplementvillkor
@ 'satser'
    MOV             R0,R4
    BL              g        @ g(i);
    ORR             R5,R5,R0 @ bits = bits | g(i);
@ 'uttryck 3'
    ADD             R4,R4,#1  @ i++;
    B               .L1
@ 'For_Break'
.L2:
    MOV             R0,R5    @ "bits" -> R0 (returvärde)
    POP             {R4,R5,R6,PC}
```

# Utformning av tentamen

Tentamen består av uppgifter hämtade i första hand från följande problemområden:

- Översättning av C till assemblerspråk
  - hur tillämpas kompilatorkonventioner (parametrar och returvärden) ?
  - hur kodas evaluering av uttryck i assemblerspråk ?
  - hur kodas enkla funktioner i assemblerspråk och hur kan man styra kodgenerering hos en C-kompilator ? (ej EDA488)
- Maskinnära programmering i C
  - hur används typsystemet för att skapa deklARATIONER av portar (sammansatta typer och bitfält) ?
  - vad menas med *pekare* och hur kan dom användas ?
  - hur använder man och skapar nya programbibliotek (ej EDA488)
- Konfigurering och användning av systemenheter och periferikretsar
  - hur konfigureras GPIO för digital in- och utmatning ?
  - hur används systemenheten SYSTICK (med eller utan avbrott) för synkronisering i realtid ?
  - hur konfigureras SYSCFG, EXTI och NVIC för användning med externa enheter ?

Typexempel hittar du i exempelsamlingen såväl som tidigare tentamina (se kursens hemsida).