

# Undantagshantering och interna avbrott

ARM Cortex-M4 "exceptions", programmering av undantagshantering

## Ur innehållet:

- Faults

- Software traps

- Avbrott från interna enheter, "Systick"

## Läsanvisningar:

- Arbetsbok kap 6

- STM32F4xx Cortex M4 programming manual

  - Kap 2.3 "Exception model"

  - SCB, sidor 206-227

# Exception Handling

“Undantagshantering” är det sammanfattande begreppet för när normal sekventiell exekvering inte kan, eller ska, fortsätta.

Detta är föranlett av någon “exceptionell” händelse i systemet.

Ett inledande exempel på undantagshantering:

Vi vet att ARM-processorns load/store- instruktioner optimerats genom att inte kunna referera word (4 bytes) på en udda address.

Men vi kan enkelt skriva ett program som gör det, vad händer då?



```
@ exception_unhandled.asm
```

```
start:
```

```
    LDR    R0,=0x20001001
```

```
    LDR    R0, [R0]
```

```
    NOP
```

```
void main(void)
```

```
{
```

```
    int *ip, i;
```

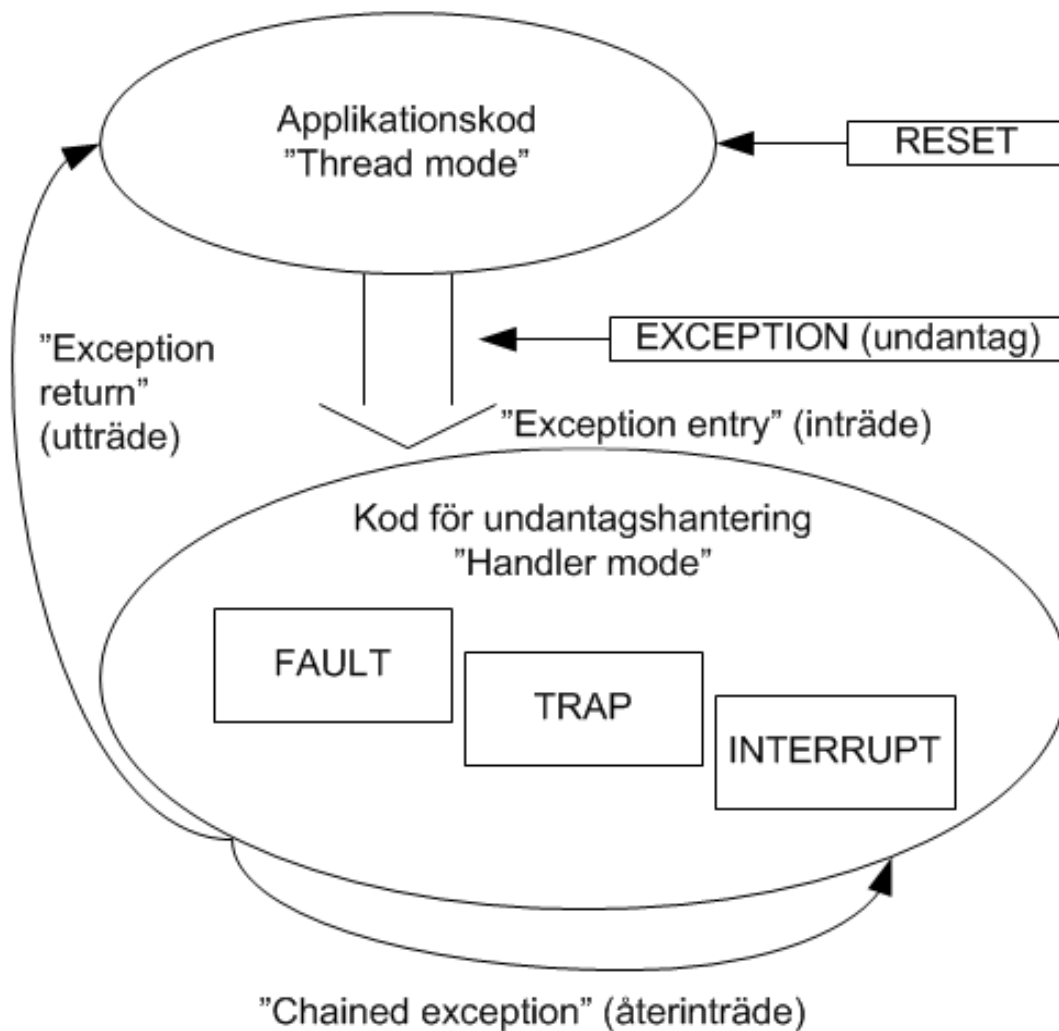
```
    ip = (int *) 0x20001001;
```

```
    i = *ip;
```

```
}
```

This is an address to  
a short NOT to an int!

# Exception handling - "exception"

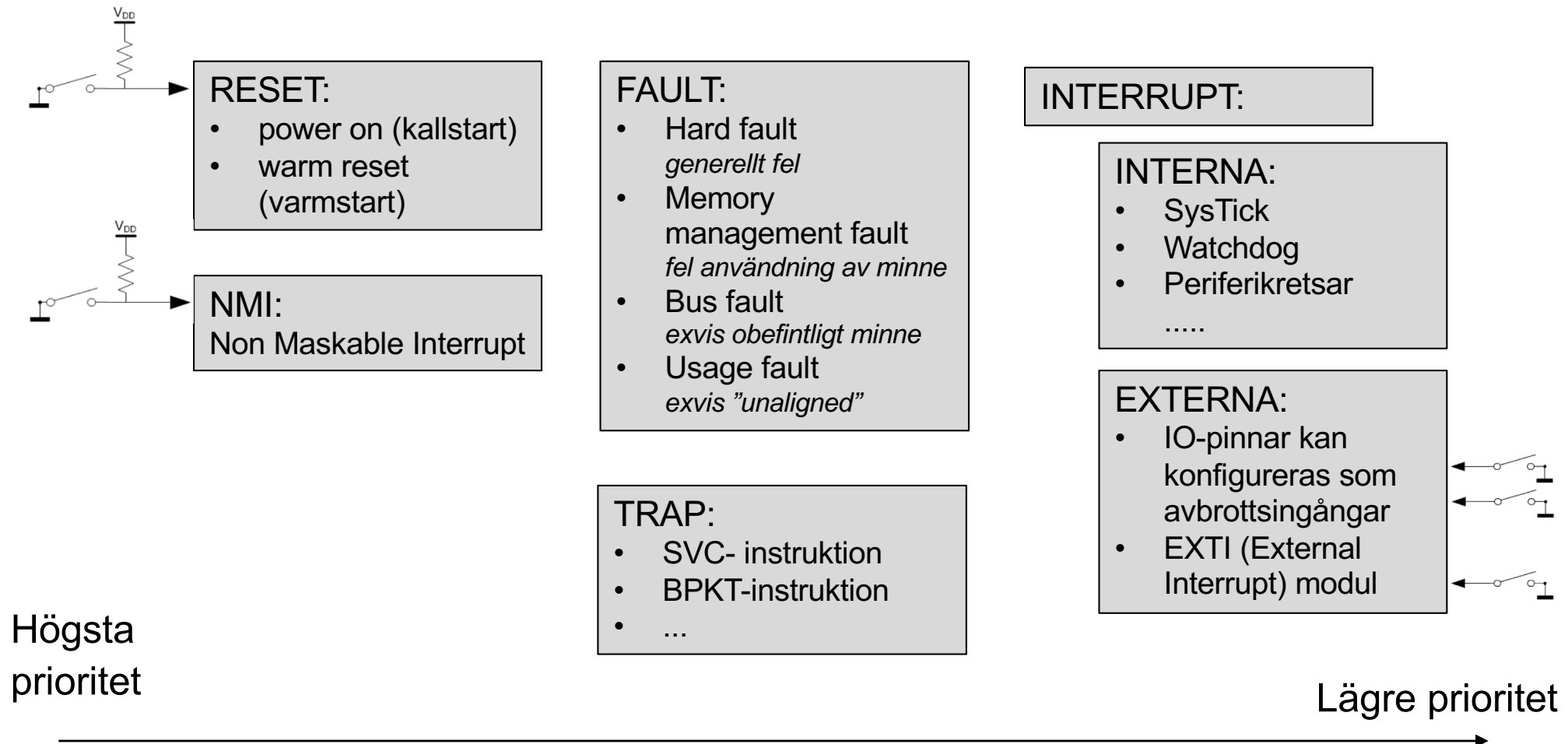


Med "undantag" menar vi här en rad olika typer av händelser:

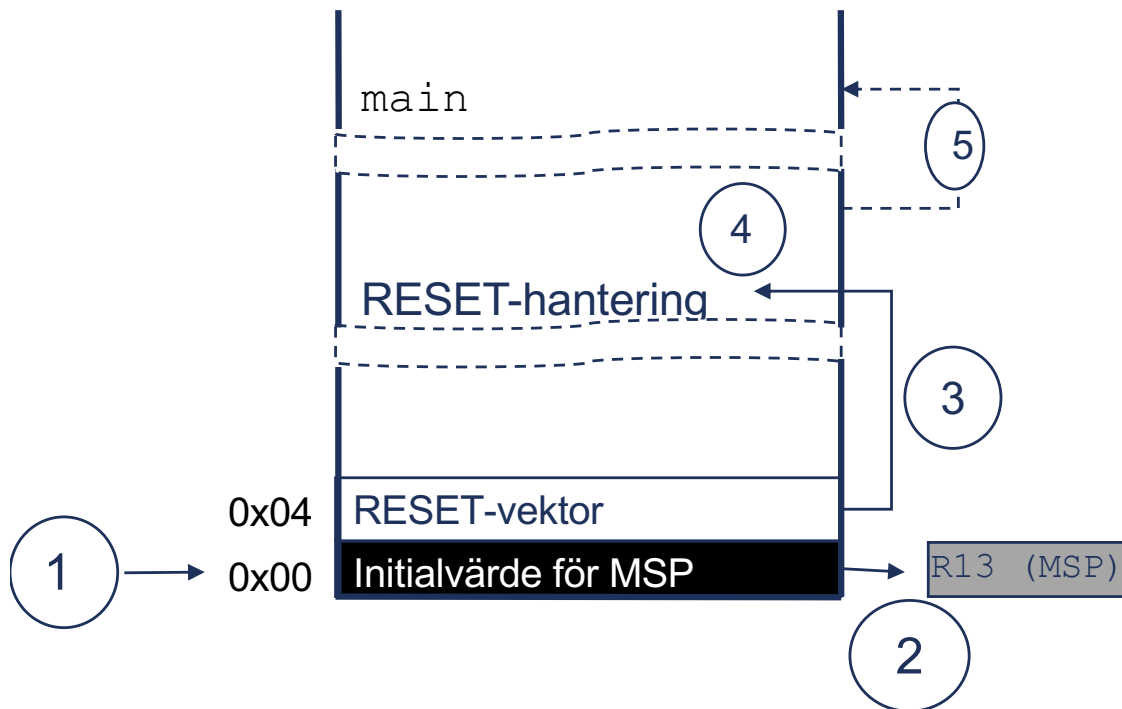
- **RESET (återstart):**
  - *power on* (kallstart)
  - *warm reset* (varmstart)
- **FAULT:**
  - Exekveringsfel – kan inte fortsätta
- **TRAP:**
  - Programmerat avbrott, initierat av maskininstruktion
- **INTERRUPT:**
  - Hårdvarusignalerat avbrott

# Exception type

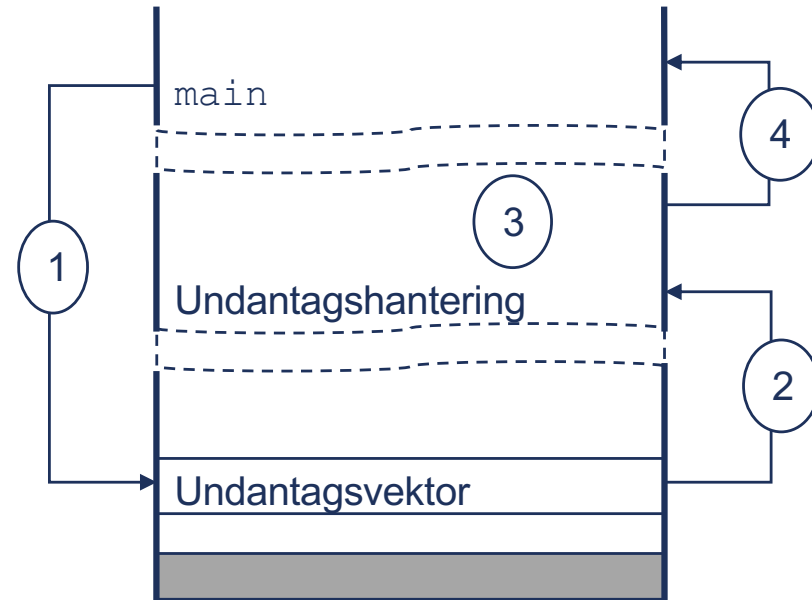
De olika undantagstyperna har en naturlig, fallande prioritet



# Exception execution



1. Återstart (Reset aktiverad)
2. Ladda MSP (Main Stack Pointer) från adress 0x00
3. Ladda RESET-vector från adress 0x04
4. RESET-hantering exekveras i "Thread mode"
5. Systemets huvudprogram startas



1. Något undantag inträffar
  - Pågående instruktion utförs klart
  - Vektortabellen adresseras
2. Den specifika undantagsvektorn hämtas
3. Undantagshantering i "Handler Mode"
4. Återgång efter undantagshantering

# Vector table

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               |
|     | -1 | fixed    | HardFault     | All class of fault                                                                       | 0x0000 000C               |
|     | 0  | settable | MemManage     | Memory management                                                                        | 0x0000 0010               |
|     | 1  | settable | BusFault      | Pre-fetch fault, memory access fault                                                     | 0x0000 0014               |
|     | 2  | settable | UsageFault    | Undefined instruction or illegal state                                                   | 0x0000 0018               |
|     | .  | .        | .             | Reserved                                                                                 | 0x0000 001C - 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               |
| 0   | 7  | settable | WWDG          | Window Watchdog interrupt                                                                | 0x0000 0040               |
| 1   | 8  | settable | PVD           | PVD through EXTI line detection interrupt                                                | 0x0000 0044               |
| 2   | 9  | settable | TAMP_STAMP    | Tamper and TimeStamp interrupts through the EXTI line                                    | 0x0000 0048               |
| 3   | 10 | settable | RTC_WKUP      | RTC Wakeup interrupt through the EXTI line                                               | 0x0000 004C               |
| 4   | 11 | settable | FLASH         | Flash global interrupt                                                                   | 0x0000 0050               |
| 5   | 12 | settable | RCC           | RCC global interrupt                                                                     | 0x0000 0054               |
| 6   | 13 | settable | EXTI0         | EXTI Line0 interrupt                                                                     | 0x0000 0058               |
| 7   | 14 | settable | EXTI1         | EXTI Line1 interrupt                                                                     | 0x0000 005C               |
| 8   | 15 | settable | EXTI2         | EXTI Line2 interrupt                                                                     | 0x0000 0060               |
| 9   | 16 | settable | EXTI3         | EXTI Line3 interrupt                                                                     | 0x0000 0064               |
| 10  | 17 | settable | EXTI4         | EXTI Line4 interrupt                                                                     | 0x0000 0068               |
| ... |    |          |               |                                                                                          |                           |
| ... |    |          |               |                                                                                          |                           |
| ... |    |          |               |                                                                                          |                           |
| 81  | 88 |          | FPU           | FPU global interrupt                                                                     | 0x0000 0184               |

# Exception handling - details

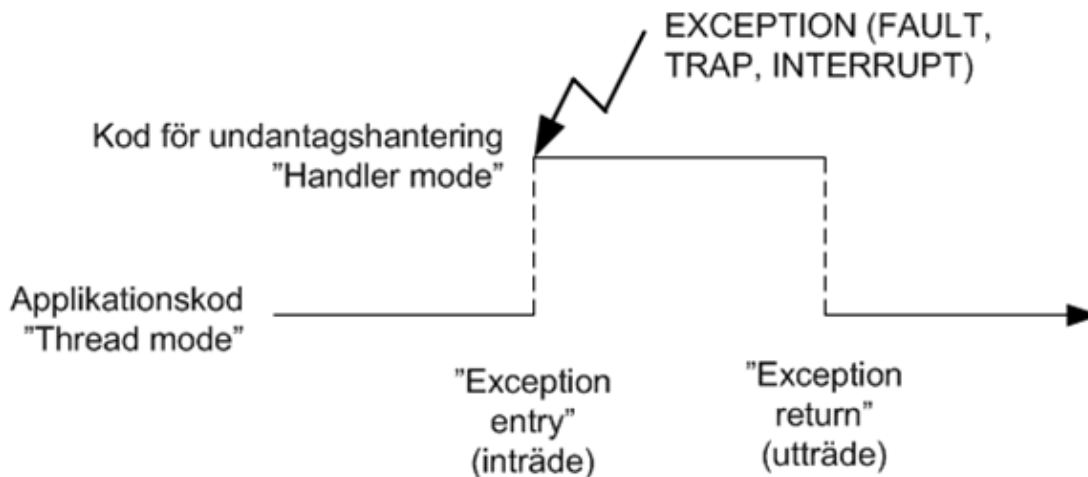
"Inträdet" hanteras automatiskt av processor.  
 "Utträdet" initieras och handhas av programvaran (undantagsrutinen)

## Utträde:

### Återställ avbruten processorstatus ("restore context")

Återställ PC från LR. Det speciella EXC\_RETURN värde som då hamnar i PC avkodas.

Registerinnehåll återställs då från stacken varvid PC får rätt återhoppadress.



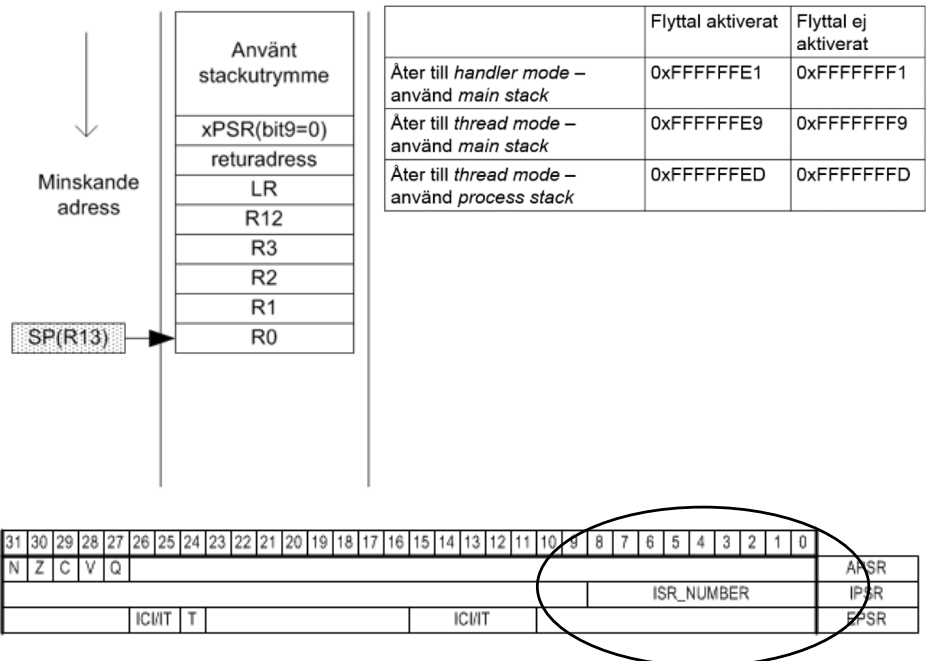
## Inträde:

**Spara aktuell processorstatus ("save context"),** 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



# Relocation of the vector table

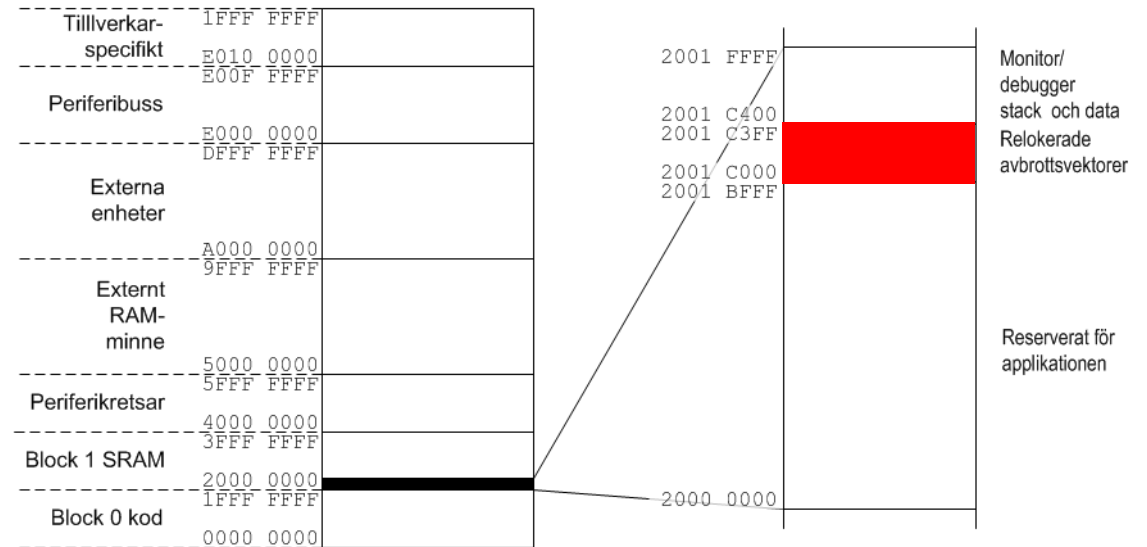
Vektortabellen startar på address 0 i minnet (Alltid RO-minne) ....

... men kan relokeras så att vektorerna hamnar i RWM, exempelvis:

```
#define SCB_VTOR ((unsigned long *)0xE00ED08)
```

```
*SCB_VTOR = 0x2001C000;
```

**System Control Block (SCB) 0xE00ED00**  
**Vector Table Offset Register (VTOR) 0x8**



**EXEMPEL: Initiera avbrottsvektor "usage fault"**

|   |          |                   |                                        |
|---|----------|-------------------|----------------------------------------|
| 0 | settable | Memory management | 0x0000 0010                            |
| 1 | settable | BusFault          | Pre-fetch fault, memory access fault   |
| 2 | settable | UsageFault        | Undefined instruction or illegal state |
| . | .        | .                 | Reserved                               |
| 3 | settable | SVCall            | System service call via SWI            |

```
void usage_fault_handler( void );
```

```
...
```

```
*((void (**)(void) ) 0x2001C018 ) = usage_fault_handler;
```

# Return from interrupt

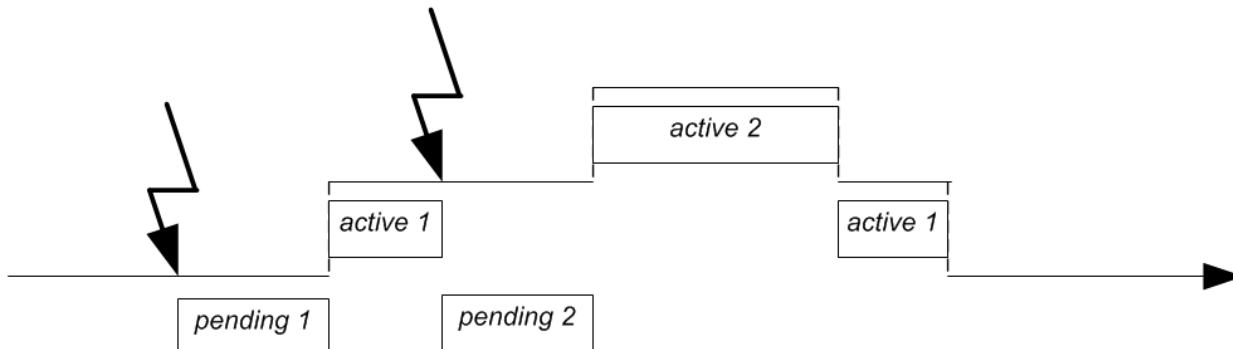
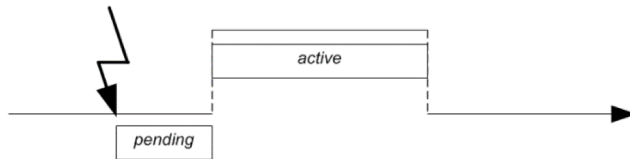
Värdet i LR (EXC\_RETURN) anger detaljerna för återställning av stack och PC efter avbrott

|                                                            | Flyttal aktiverat | Flyttal ej aktiverat |
|------------------------------------------------------------|-------------------|----------------------|
| Åter till <i>handler mode</i> – använd <i>main stack</i>   | 0xFFFFFFE1        | 0xFFFFFFFF1          |
| Åter till <i>thread mode</i> – använd <i>main stack</i>    | 0xFFFFFFE9        | 0xFFFFFFFF9          |
| Åter till <i>thread mode</i> – använd <i>process stack</i> | 0xFFFFFED         | 0xFFFFFED            |

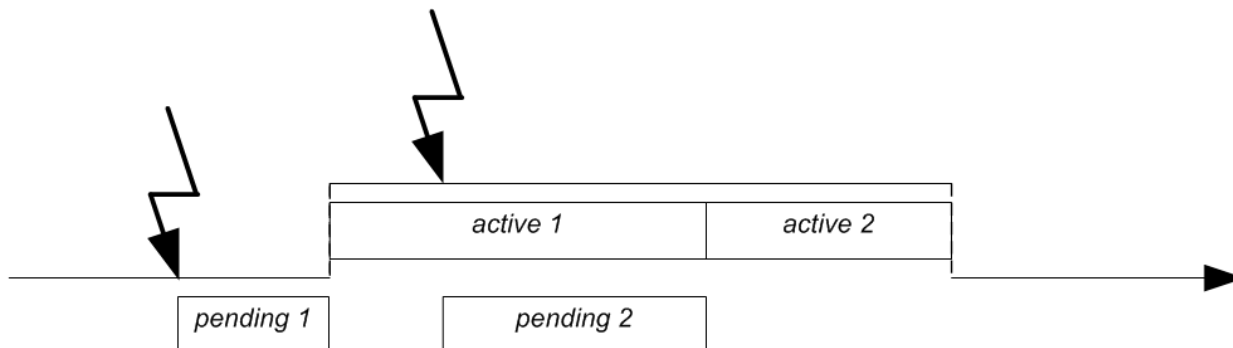
- ☐ Kan återvända från avbrott med följande instruktioner då PC laddas med “det magiska värdet” 0xFFFF\_FFFX
  - LDR PC, .....
  - LDM/POP            då PC ingår i registerlistan
  - BX LR              mest vanligt, typiskt retur från en subrutin
- ☐ Om inga ytterligare avbrott avvaktar:
  - återställs stack och tillstånd baserat på EXC\_RETURN
- ☐ Om avbrott avvaktar:
  - återställning av stack/tillstånd fördröjs (“tail-chaining”)
  - avvaktande avbrott betjänas

# Interrupt priority

Flera avbrott, *pending*, *active* och *prioriteter*



Avbrott 2 har högre prioritet,  
ytterligare “context switch”



Avbrott 2 har lägre prioritet,  
“tail-chained” interrupt

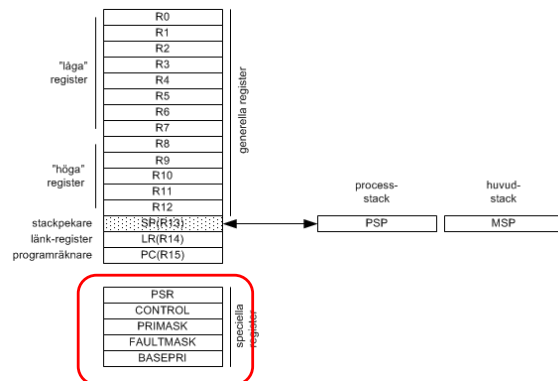
# Special registers

Speciella register används för att ange:

- Om flyttalsprocessor ska aktiveras
- Vilken stack som används
- Vilken exekverings-mode som används
- Om avbrott ska betjänas

EXEMPEL:

```
void __setCONTROL( unsigned int val)
{
    __asm(
        " MSR    CONTROL, R0\n"
    );
}
```



|    |    |    |    |    |        |    |    |    |    |    |    |    |        |    |    |            |    |    |            |      |    |   |   |   |   |   |      |      |   |   |   |  |
|----|----|----|----|----|--------|----|----|----|----|----|----|----|--------|----|----|------------|----|----|------------|------|----|---|---|---|---|---|------|------|---|---|---|--|
| 31 | 30 | 29 | 28 | 27 | 26     | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18     | 17 | 16 | 15         | 14 | 13 | 12         | 11   | 10 | 9 | 8 | 7 | 6 | 5 | 4    | 3    | 2 | 1 | 0 |  |
| N  | Z  | C  | V  | Q  |        |    |    |    |    |    |    |    |        |    |    |            |    |    | ISR_NUMBER |      |    |   |   |   |   |   | APSR |      |   |   |   |  |
|    |    |    |    |    |        |    |    |    |    |    |    |    |        |    |    | ISR_NUMBER |    |    |            |      |    |   |   |   |   |   |      | IPSR |   |   |   |  |
|    |    |    |    |    | IC/IIT |    | T  |    |    |    |    |    | IC/IIT |    |    |            |    |    |            | EPSR |    |   |   |   |   |   |      |      |   |   |   |  |

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |       |   |   |   |         |   |   |   |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|---------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7     | 6 | 5 | 4 | 3       | 2 | 1 | 0 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   | F S P |   |   |   | CONTROL |   |   |   |

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |         |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |   |         |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   | M | PRIMASK |

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |   |           |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|-----------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |   |           |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   | M | FAULTMASK |

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |          |   |   |   |   |   |   |   |         |  |  |  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|---------|--|--|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7        | 6 | 5 | 4 | 3 | 2 | 1 | 0 |         |  |  |  |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   | PRIORITY |   |   |   |   |   |   |   | BASEPRI |  |  |  |

# Mask exceptions

I undantagsrutinen kan man tillfälligt höja den egna prioriteten till högsta nivå genom att sätta prioritetsmasken i PRIMASK till 1.

```
void disable_interrupt( void )
{
    __asm(
        " CPSID I\n"
    );
}
```

Undantagsrutinen måste då också återställa prioritetsmasken till 0 innan återgång. I annat fall kommer alla framtida avbrott att blockeras.

# The different states of the processor

*Handler (undantagshantering)/Thread mode (normal exekvering)*, sköts automatiskt av processorn.

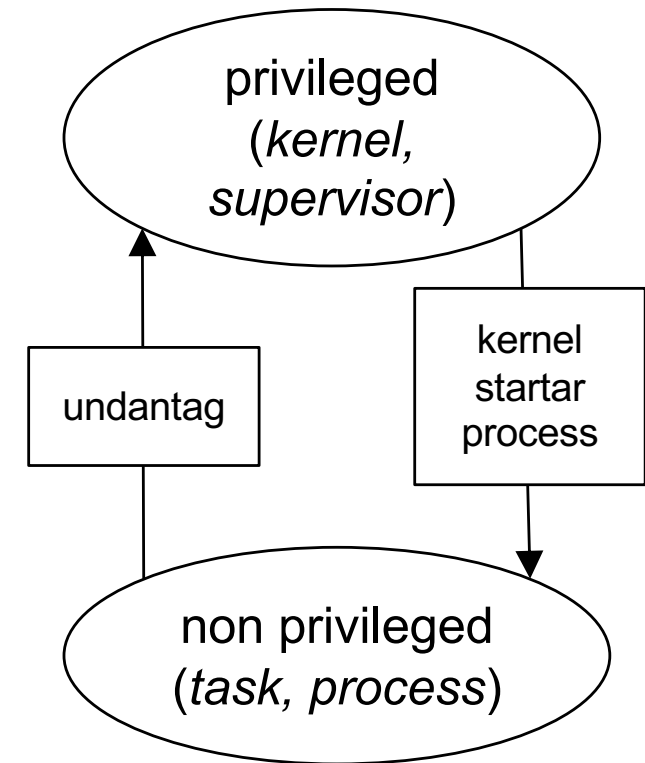
För att underlätta konstruktion av operativsystem finns också: *Privileged/Non-privileged state*, kan programmeras.

I *Non-privileged state* fungerar *priviligierade* instruktioner som NOP.

*Main stack/Process stack* – kan programmeras, processorn använder olika fysiska register (MSP eller PSP) som stackpekare.

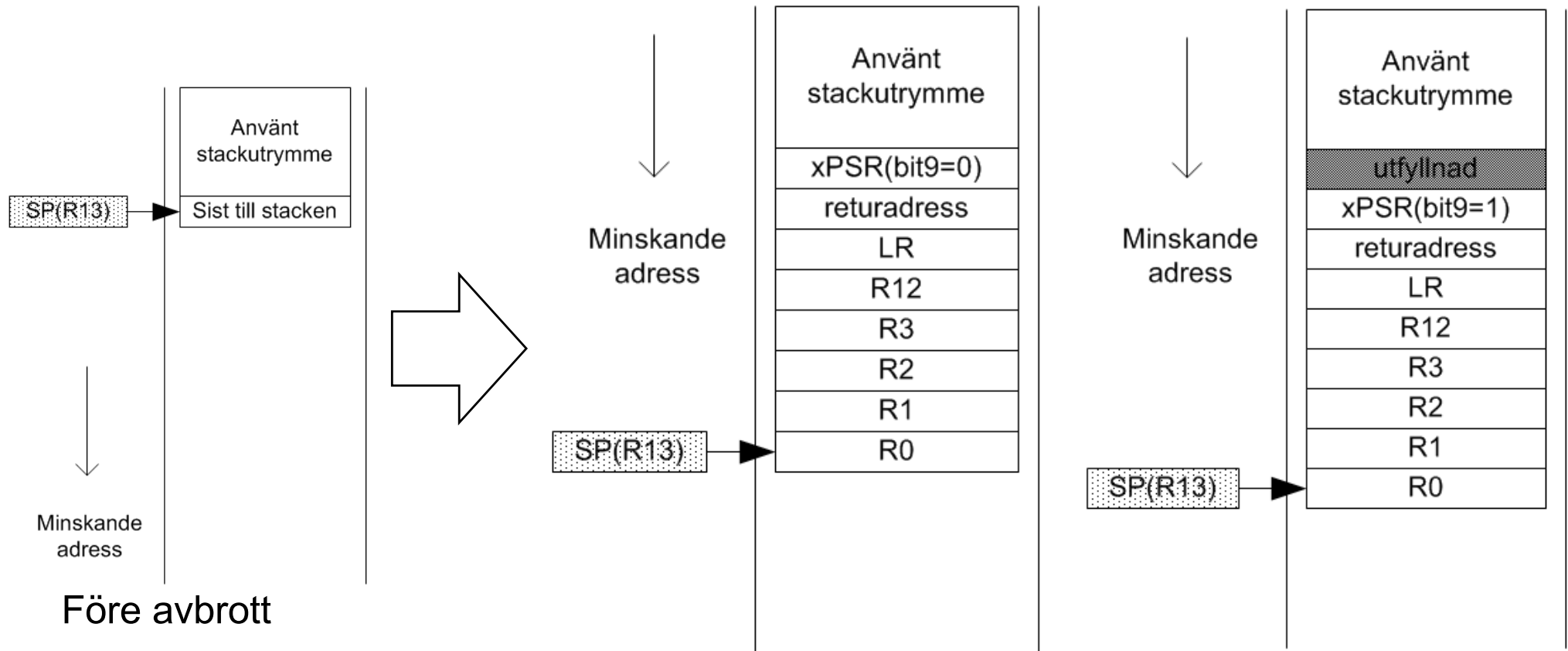
Operativsystemets programvara (*kernel*) exekveras med alla resurser tillgängliga (priviligierat).

Applikationsprogrammet exekveras i en begänsad miljö (icke privilegierat), övervakad av *kernel*.



Ej aktiverad flyttalsenhet

# The stack in the interrupt function



Processorn fyller automatiskt ut med 4 bytes om detta skulle krävas för att SP ska innehålla adress på adress jämnt delbar med 8. Detta indikeras med att bit9 i PSR sätts till 1.

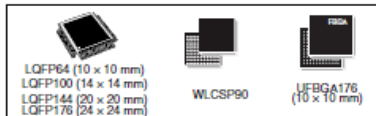
# Internal interrupts



## STM32F405xx STM32F407xx

ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera

Datasheet - production data



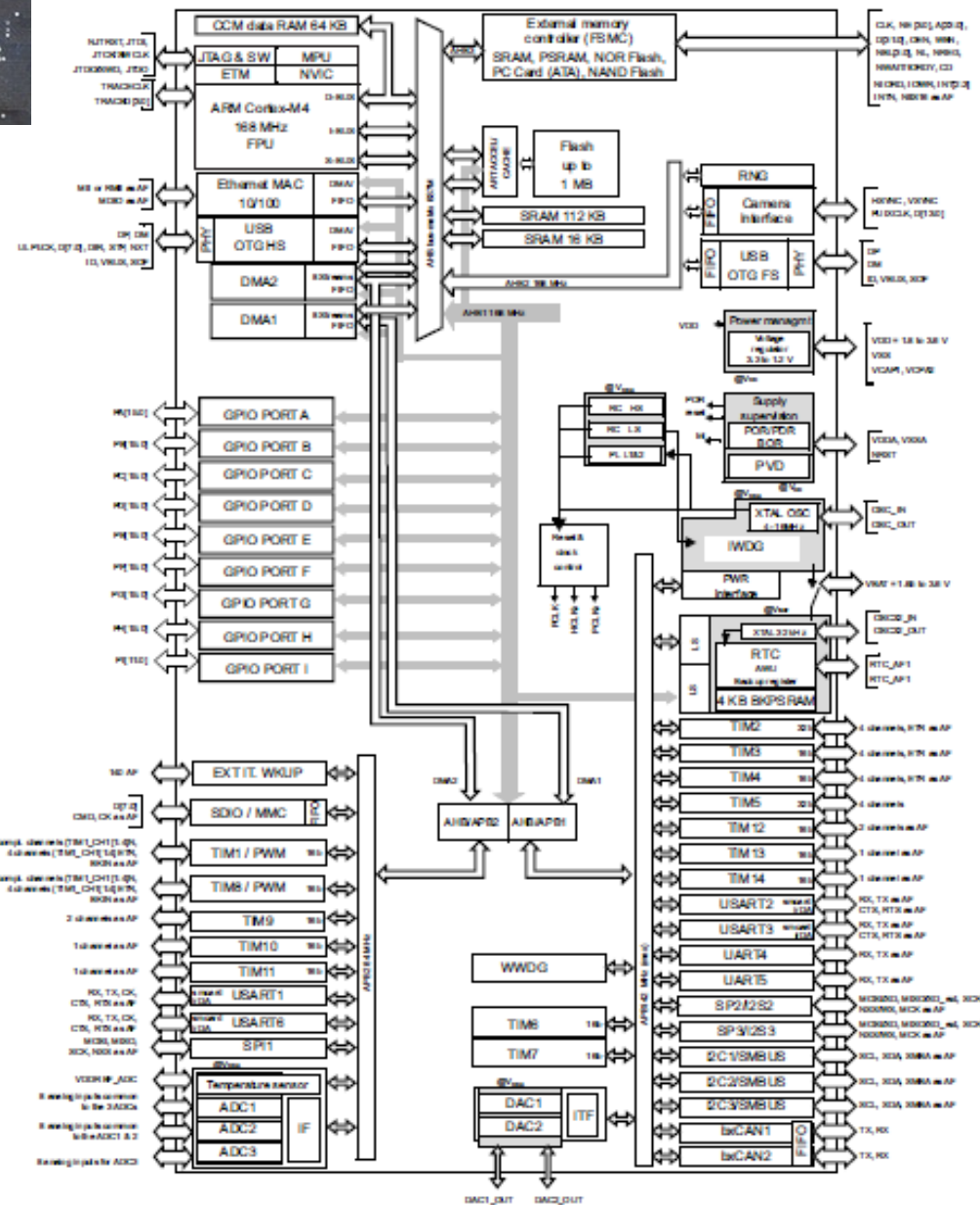
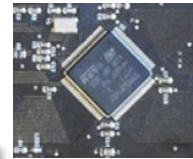
### Features

- Core: ARM 32-bit Cortex™-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 168 MHz, memory protection unit, 210 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
  - Up to 1 Mbyte of Flash memory
  - Up to 192+4 Kbytes of SRAM including 64-Kbyte of CCM (core coupled memory) data RAM
  - Flexible static memory controller supporting Compact Flash, SRAM, PSRAM, NOR and NAND memories
- LCD parallel interface, 8080/6800 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 26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - 32 kHz oscillator for RTC with calibration
  - Internal 32 kHz RC with calibration
- Low power
  - Sleep, Stop and Standby modes
  - V<sub>BAT</sub> supply for RTC, 20x32 bit backup registers + optional 4 KB backup SRAM
- 3x12-bit, 2.4 MSPS A/D converters: up to 24 channels and 7.2 MSPS in triple interleaved mode
- 2x12-bit D/A converters
- General-purpose DMA: 16-stream DMA controller with FIFOs and burst support
- Up to 17 timers: up to twelve 16-bit and two 32-bit timers up to 168 MHz, each with up to 4

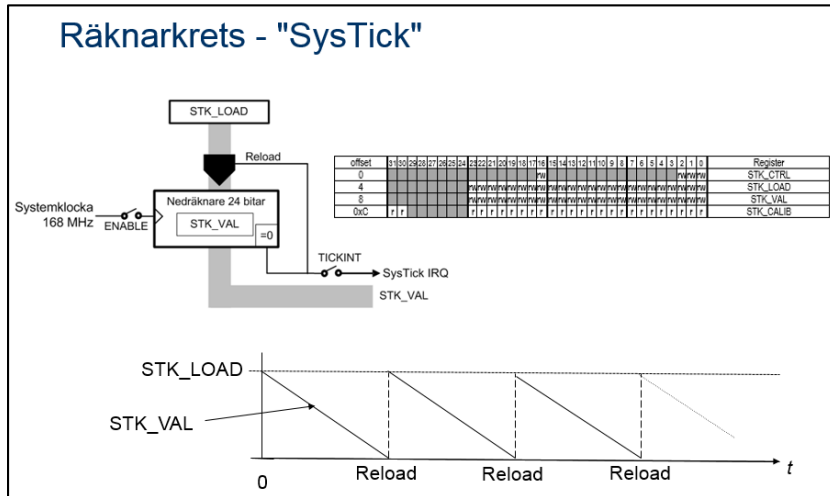
- IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
  - Cortex-M4 Embedded Trace Macrocell™
- Up to 140 I/O ports with interrupt capability
  - Up to 136 fast I/Os up to 84 MHz
  - Up to 138 5 V-tolerant I/Os
- Up to 15 communication interfaces
  - Up to 3 x I<sup>2</sup>C interfaces (SMBus/PMBus)
  - Up to 4 USARTs/2 UARTs (10.5 Mbit/s, ISO 7816 interface, LIN, IrDA, modem control)
  - Up to 3 SPIs (42 Mbit/s), 2 with muxed full-duplex I<sup>2</sup>S to achieve audio class accuracy via internal audio PLL or external clock
  - 2 x CAN interfaces (2.0B Active)
  - SDIO 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 ULP
  - 10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII
- 8- to 14-bit parallel camera interface up to 54 Mbytes/s
- True random number generator
- CRC calculation unit
- 96-bit unique ID
- RTC: subsecond accuracy, hardware calendar

Table 1. Device summary

| Reference   | Part number                                                                  |
|-------------|------------------------------------------------------------------------------|
| STM32F405xx | STM32F405RG, STM32F405VG, STM32F405ZG, STM32F405QG, STM32F405QE              |
| STM32F407xx | STM32F407VG, STM32F407IG, STM32F407ZG, STM32F407VE, STM32F407ZE, STM32F407IE |



# "SysTick" with interrupt...



Algoritm:

|                        |                           |
|------------------------|---------------------------|
| STK_CTRL = 0           | Återställ SysTick         |
| STK_LOAD =             | CountValue                |
| STK_VAL = 0            | Nollställ räknarregistret |
| STK_CTRL = 5           | Starta om räknaren        |
| Vänta till COUNTFLAG=1 |                           |
| STK_CTRL = 0           | Återställ SysTick         |

"Blockerande" – ty programmet kan inte göra något annat än vänta....

STK\_CTRL Status och styrregister

| offset | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2  | 1  | 0  | Register |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|----|----|----|----------|
| 0      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | rw |    |    |    |    |    |    |   |   |   |   |   |   |   | rw | rw | rw | STK_CTRL |

Bit 16: (COUNTFLAG):

Biten är 1 om räknaren räknat ned till 0. Biten nollställs då registret läses.

Bit 2: (CLKSOURCE) biten är 1 efter RESET:

0: Systemklocka/8

1: Systemklocka

Bit 1: (TICKINT): Aktivera avbrott

0: Inget avbrott genereras.

1: Då räknaren slagit om till 0 genererar SysTick avbrott.

Anm: Man kan programvarumässigt undersöka om räknaren räknat ned till 0 genom att kontrollera biten COUNTFLAG, det är alltså inte nödvändigt att använda avbrottsmekanismen.

Bit 0: (ENABLE) Aktivera räknare

Då ENABLE ettställs laddas värdet från STK\_LOAD till STK\_VAL och räknaren börjar räkna ned. Då räknaren nått 0, ettställs COUNTFLAG och proceduren upprepas.

0: Räknare deaktiverad

1: Räknare aktiverad

Algoritm:

|              |                           |
|--------------|---------------------------|
| STK_CTRL = 0 | Återställ SysTick         |
| STK_LOAD =   | CountValue                |
| STK_VAL = 0  | Nollställ räknarregistret |
| STK_CTRL = 7 | Starta om räknaren        |

Avbrott genereras då COUNTFLAG=1

"Icke-blockerande" – ty programmet kan fortsätta med att exekvera "parallell" kod

## EXAMPLE: Interrupt from "SysTick"

Skapa en fördröjningsfunktion med en global variabel i form av ett "meddelande".

Visa hur ett testprogram kan använda funktionen för att åstadkomma fördröjning i en bunden programslinga *utan att blockeras*.

Testprogrammet ska tända en diodramp under den tid fördröjningen varar.

*Vi löser på tavlan...*

```
void init_app( void )
{
    /* Initiera port D ... */
    *((void (**)(void) ) 0x2001C03C ) = systick_irq_handler;
}

#define DELAY_COUNT 10000

void main(void)
{
    init_app();
    *GPIO_ODR_LOW = 0;
    delay( DELAY_COUNT );
    *GPIO_ODR_LOW = 0xFF;
    while(1)
    {
        if( systick_flag )
            break;
        /* "Parallell kod" ... */
    }
    *GPIO_ODR_LOW = 0;
}
```