

# Undantagshantering och interna avbrott

## Ur innehållet:

- Faults
- Software traps
- Undantagsprioriteter
- Avbrott från interna enheter, "Systick"

## Läsanvisningar:

- Arbetsbok kap 6.1-6.4

## Målsättningar:

- Förklara hur undantagshantering går till
- Implementera en icke-blockerande fördröjning med SysTick

# Undantagshantering

“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

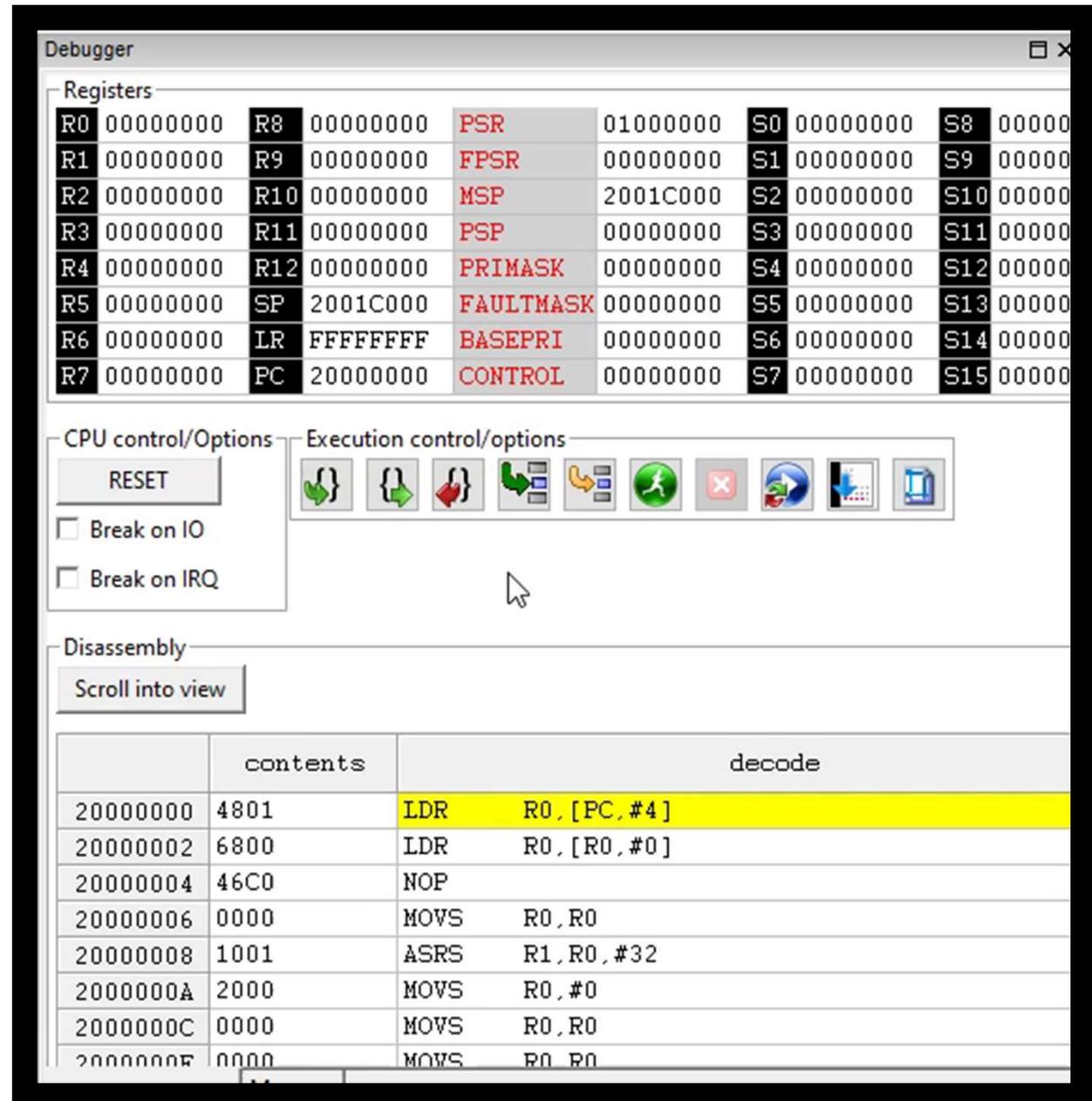
start:
    LDR    R0, =0x20001001
    LDR    R0, [R0]
    NOP
```

```
void main(void)
{
    int *ip, i;
    ip = (int *) 0x20001001;
    i = *ip;
}
```

*Vi demonstrerar med simulator*

```

@
start:
    LDR    R0, =0x20001001
    LDR    R0, [R0]
    NOP
    
```



**Debugger**

**Registers**

R0	00000000	R8	00000000	PSR	01000000	S0	00000000	S8	00000000
R1	00000000	R9	00000000	FPSR	00000000	S1	00000000	S9	00000000
R2	00000000	R10	00000000	MSP	2001C000	S2	00000000	S10	00000000
R3	00000000	R11	00000000	PSP	00000000	S3	00000000	S11	00000000
R4	00000000	R12	00000000	PRIMASK	00000000	S4	00000000	S12	00000000
R5	00000000	SP	2001C000	FAULTMASK	00000000	S5	00000000	S13	00000000
R6	00000000	LR	FFFFFFFF	BASEPRI	00000000	S6	00000000	S14	00000000
R7	00000000	PC	20000000	CONTROL	00000000	S7	00000000	S15	00000000

**CPU control/Options**

RESET

Break on IO

Break on IRQ

**Execution control/options**









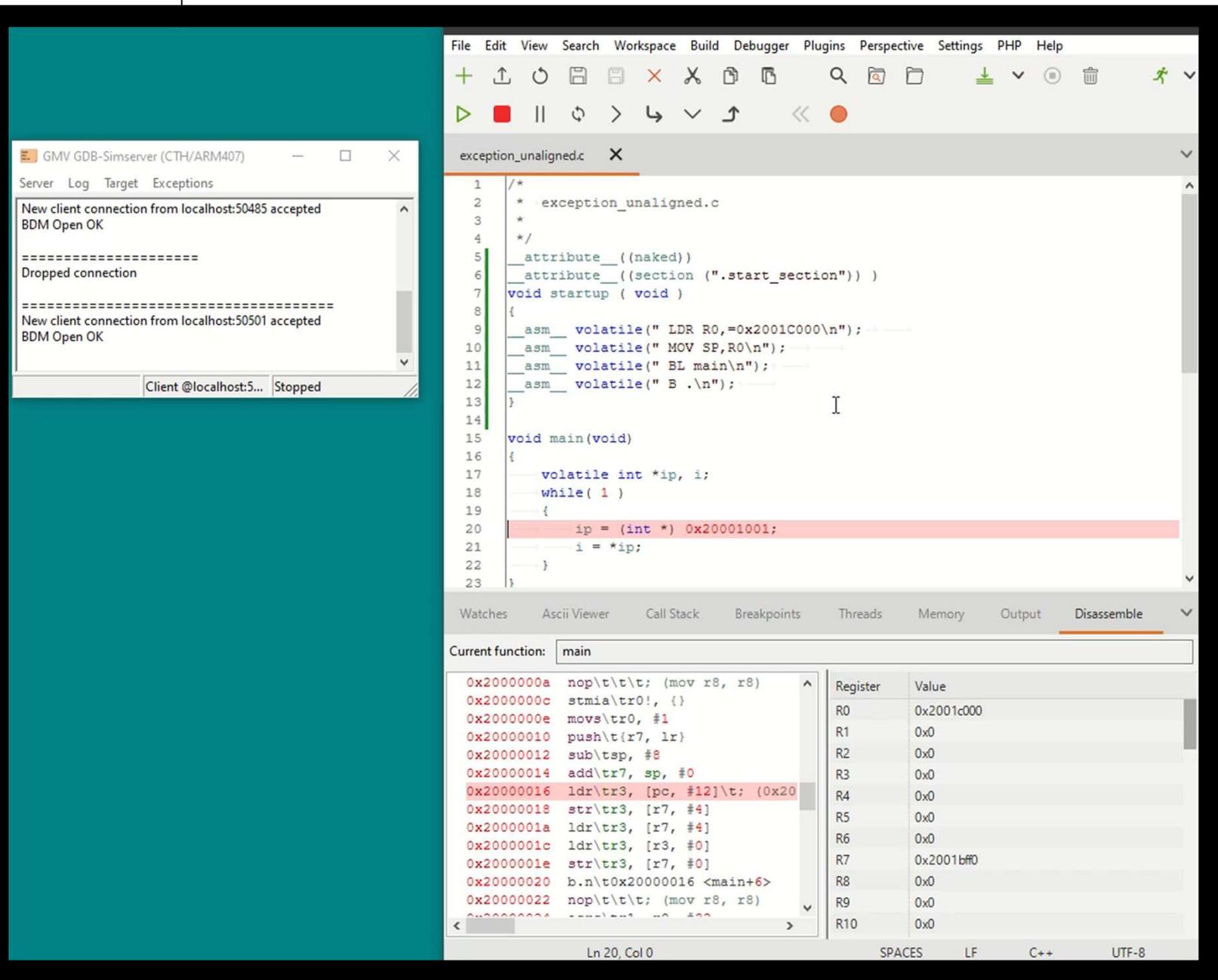



**Disassembly**

Scroll into view

	contents	decode
20000000	4801	LDR R0, [PC, #4]
20000002	6800	LDR R0, [R0, #0]
20000004	46C0	NOP
20000006	0000	MOVS R0, R0
20000008	1001	ASRS R1, R0, #32
2000000A	2000	MOVS R0, #0
2000000C	0000	MOVS R0, R0
2000000E	0000	MOVS R0, R0

```
void main(void)
{
    int *ip, i;
    ip = (int *) 0x20001001;
    i = *ip;
}
```



The screenshot shows a debugger window titled "GMV GDB-Simserver (CTH/ARM407)". The "Server Log" pane displays the following messages:

```

New client connection from localhost:50485 accepted
BDM Open OK

=====
Dropped connection

=====
New client connection from localhost:50501 accepted
BDM Open OK
    
```

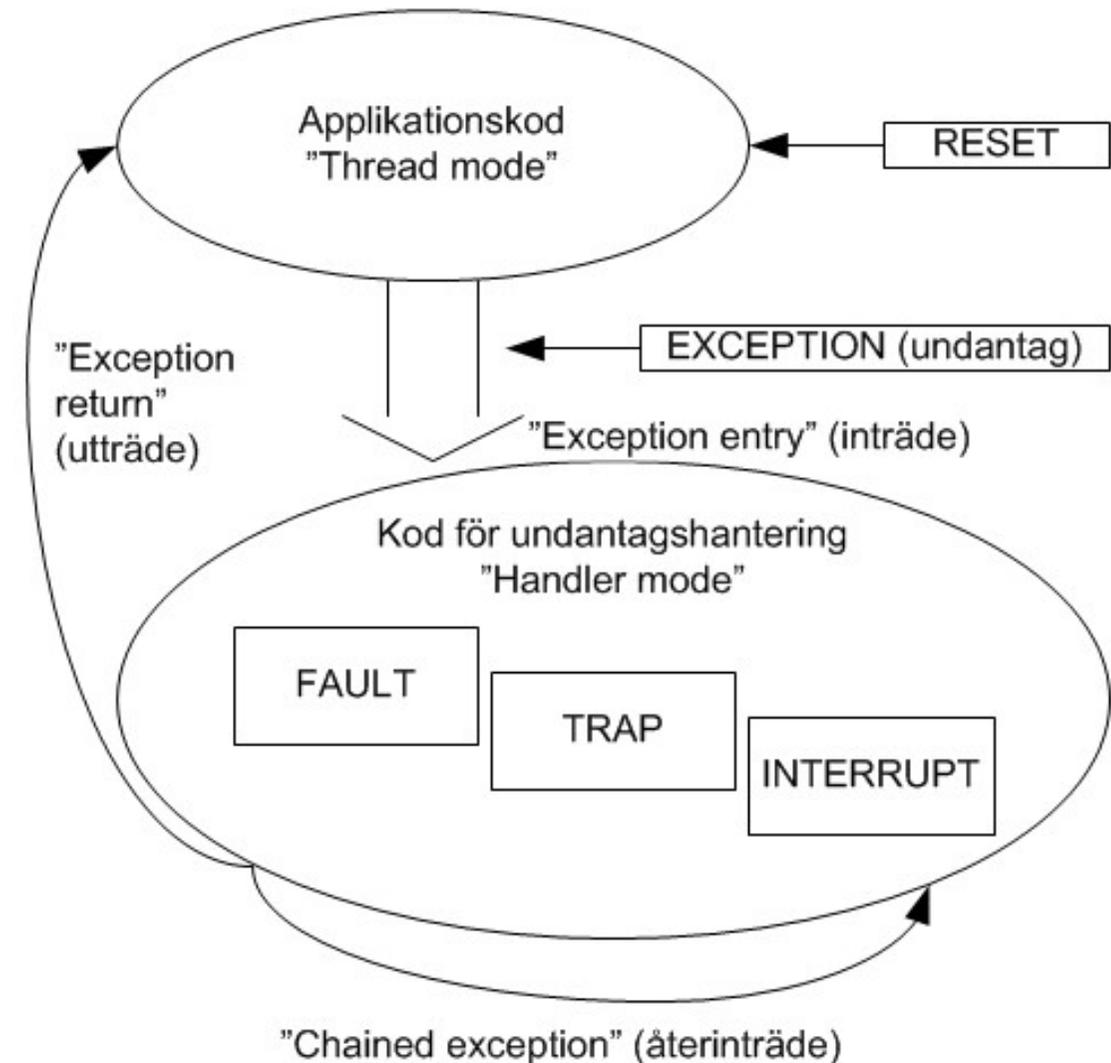
The main editor shows the source code for "exception\_unaligned.c". The function "main" is highlighted, and the line "ip = (int \*) 0x20001001;" is selected. Below the source code, the "Disassemble" pane shows the corresponding assembly instructions:

Address	Disassembly	Register	Value
0x2000000a	nop\t\t\t; (mov r8, r8)	R0	0x2001c000
0x2000000c	stmia\tr0!, {}	R1	0x0
0x2000000e	movs\tr0, #1	R2	0x0
0x20000010	push\tr7, lr	R3	0x0
0x20000012	sub\tsp, #8	R4	0x0
0x20000014	add\tr7, sp, #0	R5	0x0
0x20000016	ldr\tr3, [pc, #12]\t; (0x20001001)	R6	0x0
0x20000018	str\tr3, [r7, #4]	R7	0x2001bff0
0x2000001a	ldr\tr3, [r7, #4]	R8	0x0
0x2000001c	ldr\tr3, [r3, #0]	R9	0x0
0x2000001e	str\tr3, [r7, #0]	R10	0x0
0x20000020	b.n\t0x20000016 <main+6>		
0x20000022	nop\t\t\t; (mov r8, r8)		

# Undantagshantering - "exception"

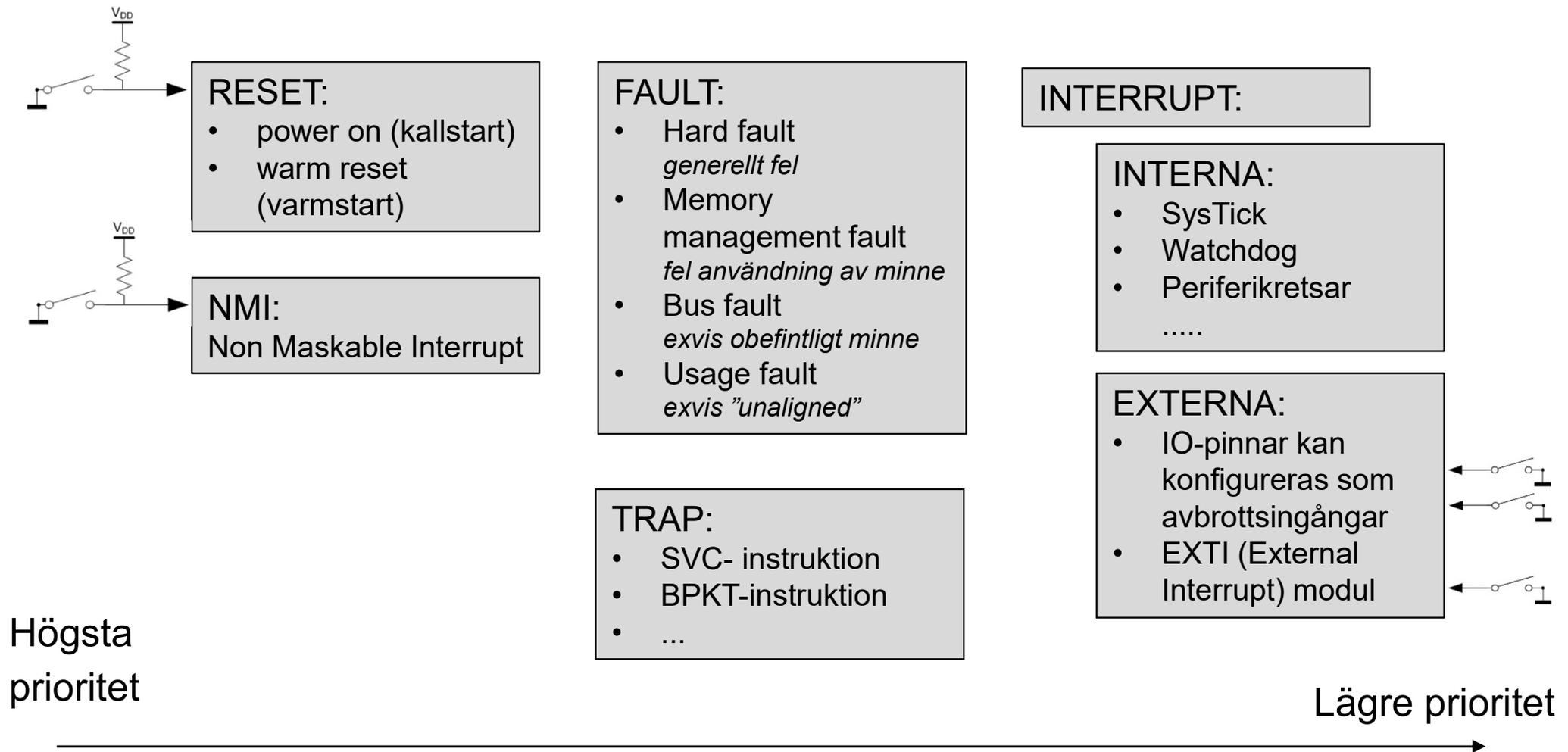
Med "undantag" menar vi olika typer av händelser:

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

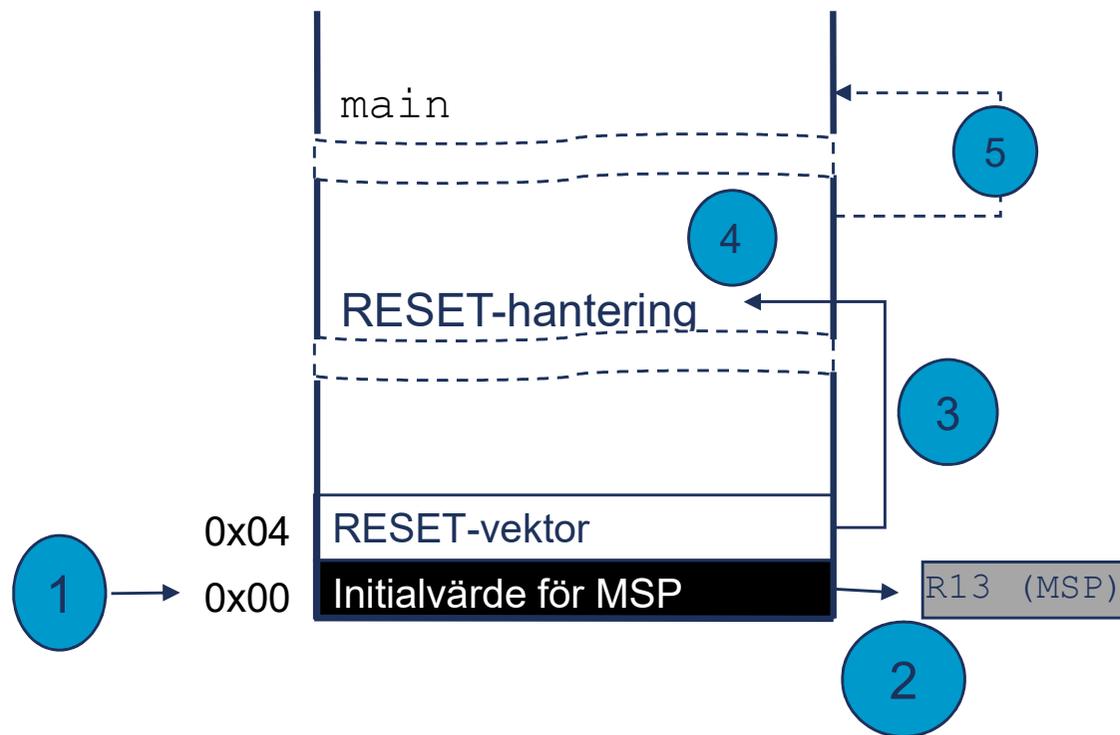


# Undantagstyper

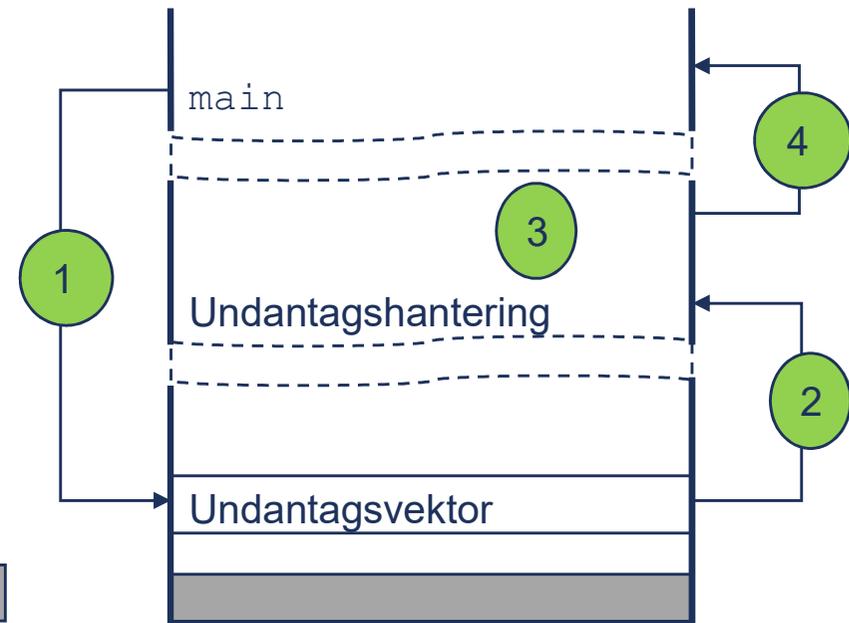
De olika undantagstyperna har en naturlig, fallande prioritet



# Exekvering vid undantag



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 "Thread Mode"

# Vektortabellen

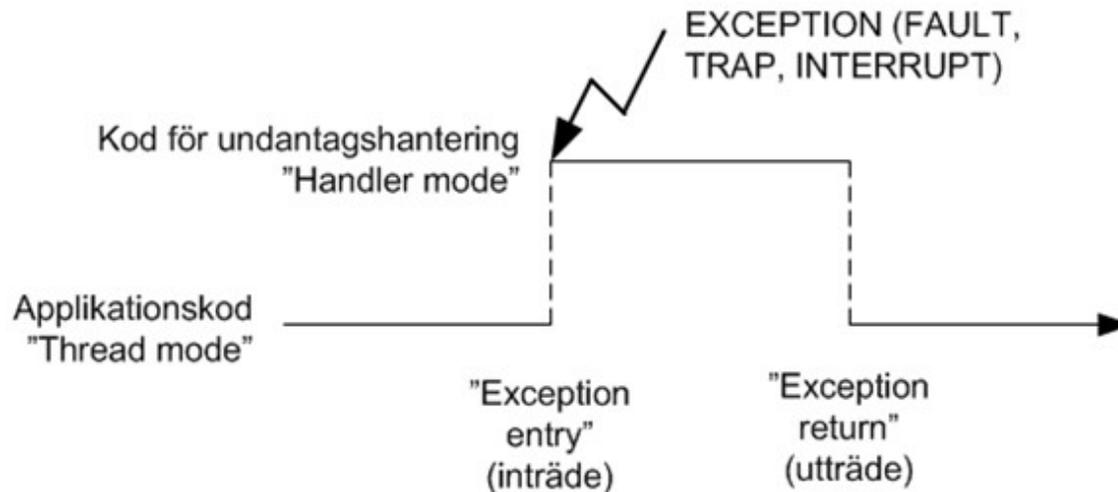
Anger position för de olika undantagsvektorer.

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

Resten av tabellen är specifik för den aktuella microcontrollern.

IRQ num	priority	name	description	vector offset
-			Reserved for initial stack pointer	0x0000 0000
-	fixed, -3	Reset	Reset	0x0000 0004
-14	fixed, -2	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-13	fixed, -1	HardFault	All class of fault	0x0000 000C
-12	settable	MemManage	Memory management	0x0000 0010
-11	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-10	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
			Reserved	0x0000 001C - 0x0000 002B
-5	settable	SVCall	System service call via SWI instruction	0x0000 002C
-4	settable	Debug Monitor	Debug Monitor	0x0000 0030
			Reserved	0x0000 0034
-2	settable	PendSV	Pendable request for system service	0x0000 0038
-1	settable	SysTick	System tick timer	0x0000 003C
0	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	settable	PVD	PVD through EXTI line detection interrupt	0x0000 0044
2	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	0x0000 004C
4	settable	FLASH	Flash global interrupt	0x0000 0050
5	settable	RCC	RCC global interrupt	0x0000 0054
6	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068
11	settable	DMA1_Stream0	DMA1_Stream0 global interrupt	0x0000 006C
19	settable	CRYP	CRYP Crypto global interrupt	0x0000 017C
80	settable	HASH_RNG	Hash and Rng global interrupt	0x0000 0180
81	settable	FPU	FPU global interrupt	0x0000 0184

# Undantagshantering - detaljerna



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

## Utträde:

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

Återställ PC. Det speciella EXC\_RETURN värdet som då hamnar i PC avkodas.  
 Registerinnehåll återställs då från stacken varvid PC får rätt återhopsadress.

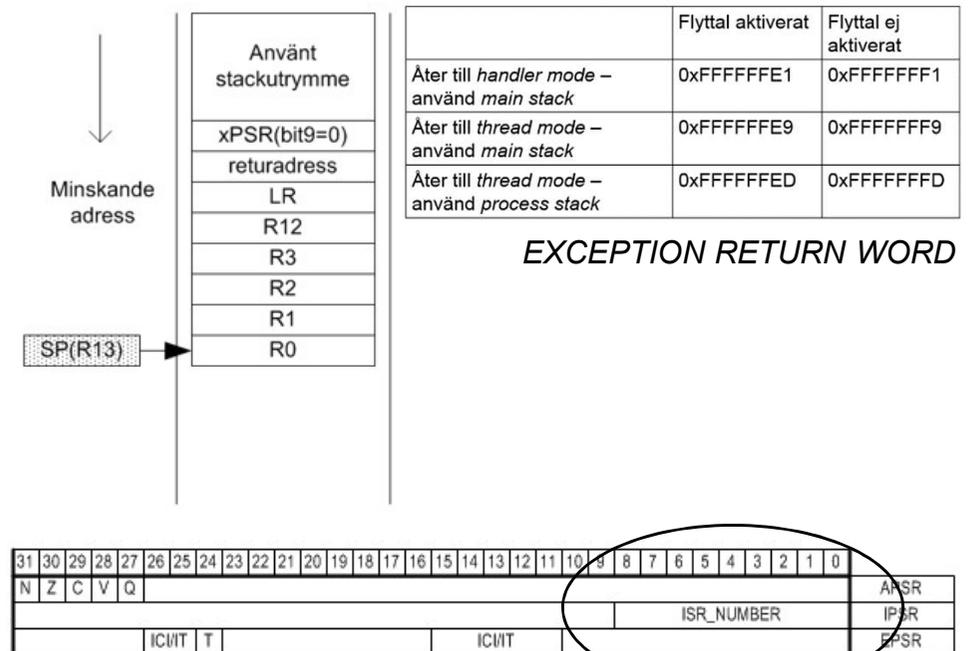
## Inträde:

Spara aktuell processorstatus ("save context"), dvs.

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

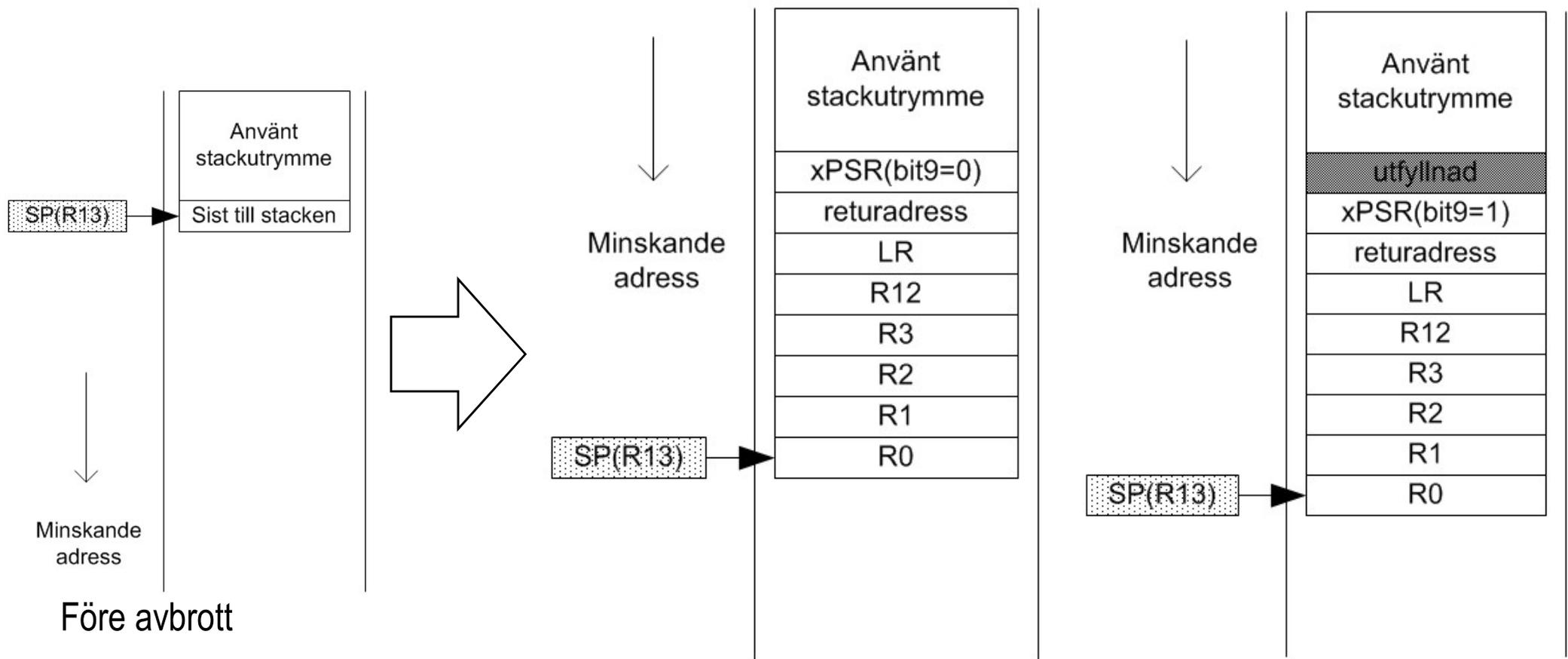
## Ändra processorstatus för undantagshantering

- Sätt undantagsnummer i PSR
- Placera undantagsvektor i PC



# Stacken i avbrottsfunktionen

Ej aktiverad flyttalsenhet



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.

# Återgång från avbrott

Värdet som laddas i PC anger detaljerna för återställning efter avbrott

## EXEC RETURN WORD

	Flyttal aktiverat	Flyttal ej aktiverat
Åter till <i>handler mode</i> – använd <i>main stack</i>	0xFFFFFFFFE1	0xFFFFFFFFF1
Åter till <i>thread mode</i> – använd <i>main stack</i>	0xFFFFFFFFE9	0xFFFFFFFFF9
Åter till <i>thread mode</i> – använd <i>process stack</i>	0xFFFFFFFFED	0xFFFFFFFFFD

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

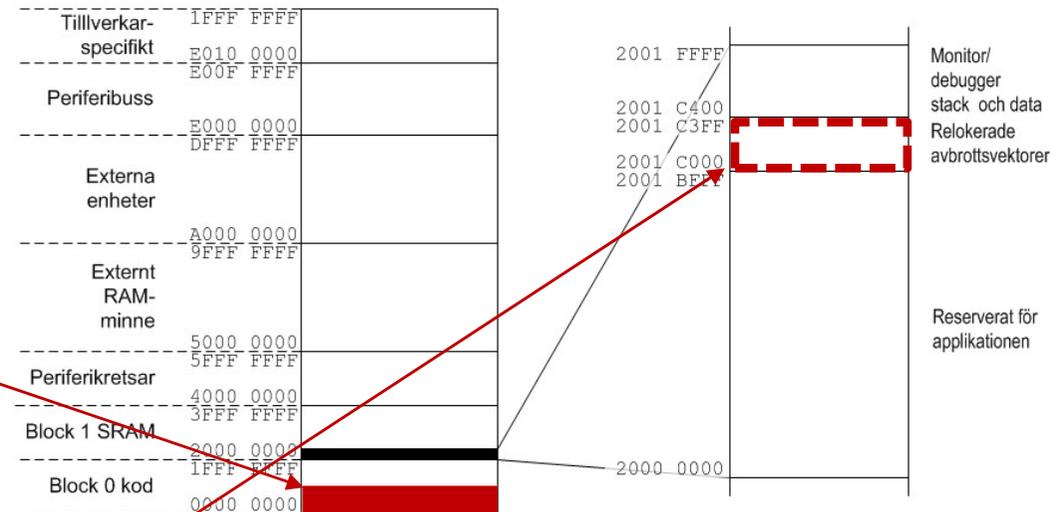
Om avbrott avvaktar:

- återställning av stack/tillstånd fördröjs (“tail-chaining”)
- avvaktande avbrott betjänas

(EXC\_RETURN)  
avgör hur stack och  
processortillstånd  
återställs efter  
avbrott

# Relokering av vektortabellen

Vektortabellen startar på address 0 i minnet (Alltid *Read Only*-minne) ....



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

```
#define SCB_VTOR ((volatile unsigned long *)0xE000ED08)
*SCB_VTOR = 0x2001C000;
```

## Exempel:

Initiera avbrottsvektor  
"usage fault"

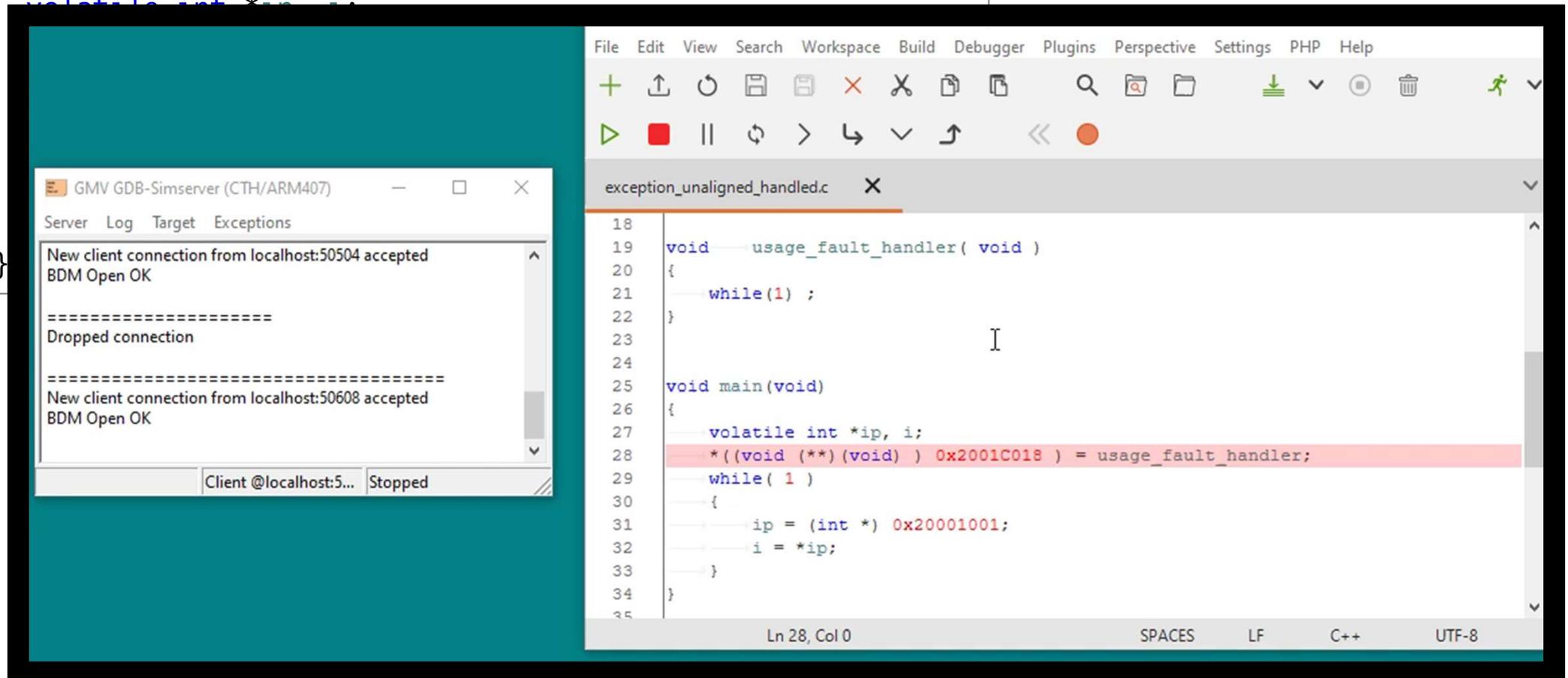
```
void usage_fault_handler( void );
...
*((void (**)(void) ) 0x2001C018 ) = usage_fault_handler;
```

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
3	settable	SVCall	System service call via SWI	0x0000 002C

```
void usage_fault_handler( void )
{
    while(1) ;
}
```

```
void main(void)
{
```

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



The screenshot shows a debugger interface with two main windows:

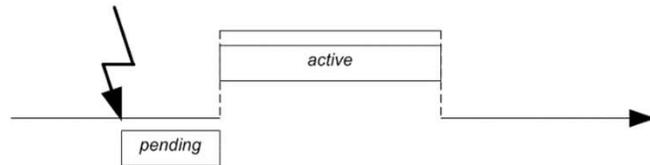
- GDB Console (Left):** Titled "GMV GDB-Simserver (CTH/ARM407)". It displays a log of server activity:
  - Server Log Target Exceptions
  - New client connection from localhost:50504 accepted
  - BDM Open OK
  - =====
  - Dropped connection
  - =====
  - New client connection from localhost:50608 accepted
  - BDM Open OK
  - Client @localhost:5... Stopped
- Source Code Editor (Right):** Titled "exception\_unaligned\_handled.c". It shows the following code:
 

```
18
19 void usage_fault_handler( void )
20 {
21     while(1) ;
22 }
23
24
25 void main(void)
26 {
27     volatile int *ip, i;
28     *((void (**)(void) ) 0x2001C018 ) = usage_fault_handler;
29     while( 1 )
30     {
31         ip = (int *) 0x20001001;
32         i = *ip;
33     }
34 }
35
```

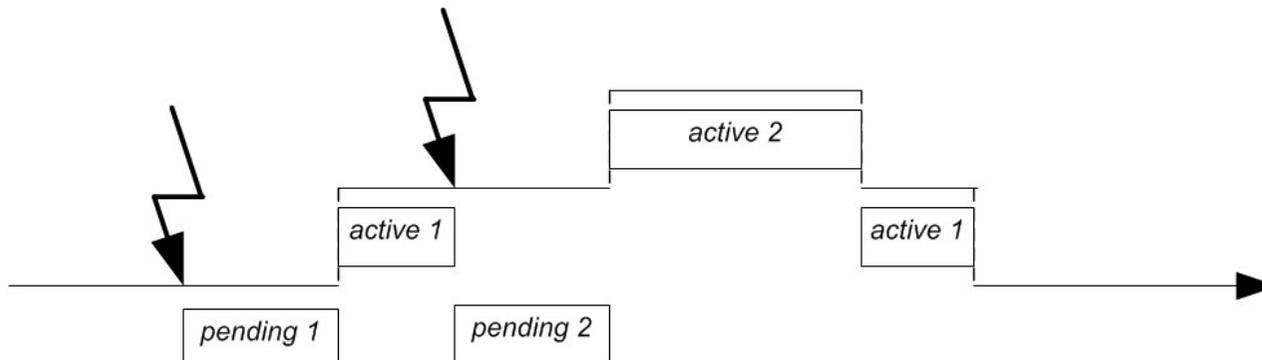
 The line 28 is highlighted in red. The status bar at the bottom indicates "Ln 28, Col 0", "SPACES", "LF", "C++", and "UTF-8".

# Avbrottsprioritet

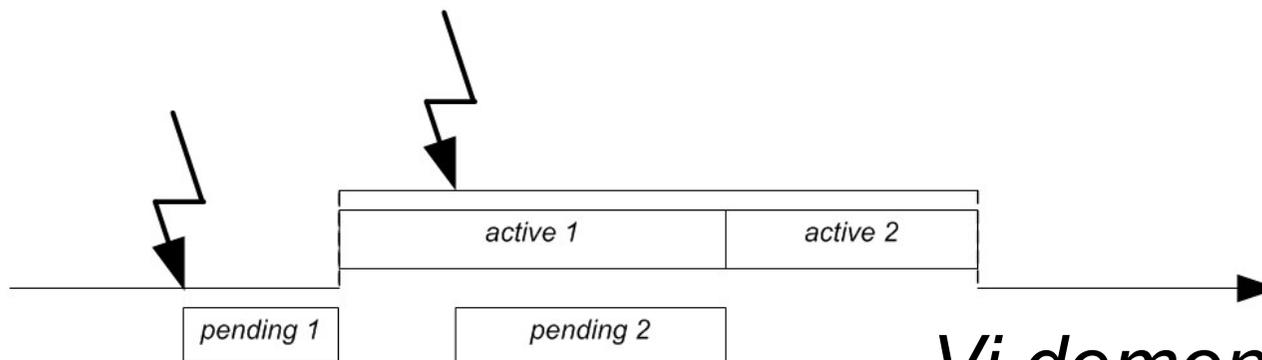
Flera samtidiga avbrott:  
*pending, active och prioriteter*



Ett avbrott uppträder och betjänas



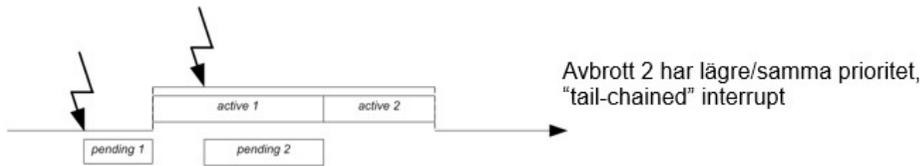
Avbrott 2 har högre prioritet, ytterligare “context switch”



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

*Vi demonstrerar med simulator*

# Avbrottsprioritet



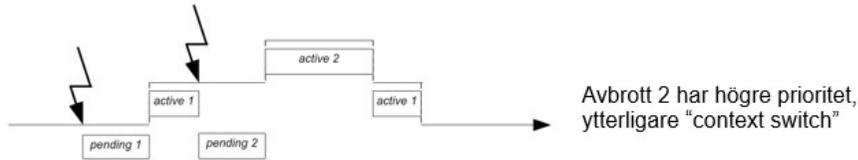
Avbrott 2 har lägre/samma prioritet, "tail-chained" interrupt

```
void svc_irq_handler( void )
{
    out7seg( 1 );
    SET_SYSTICK_IRQ_PENDING;
    out7seg( 3 );
}
void systick_irq_handler( void )
{
    out7seg( 2 );
}
void main(void)
{
    . . .
    SET_SVC_PRIORITY(0);
    SET_SYSTICK_PRIORITY(0);
    while( 1 )
    {
        out7seg( 0 );
        SET_SVC_IRQ_PENDING;
        out7seg( 4 );
    }
}
```

*Här ska sifferföljden: 0,1,3,2,4 skrivs till 7-segmentsdisplayen*

```
File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help
+ ↑ ↺ ↻ ⌂ ✕ ✂ 📄 📁 🔍 🗑️ 🏠 🚶 ⌂
▶ ■ || ⏪ > ↶ ↷ ⏩ ⏹
exception_unaligned_handled.c priorities.c X
55 void svc_irq_handler( void )
56 {
57     out7seg( 1 );
58     SET_SYSTICK_IRQ_PENDING; /* fake SYSTICK interrupt */
59     out7seg( 3 );
60 }
61 }
62
63 void systick_irq_handler( void )
64 {
65     out7seg( 2 );
66 }
67 }
68
69 // #define USE_PRIORITIES
70 void main(void)
71 {
72     *GPIO_D_MODER = 0x00005555;
73     *GPIO_D_OTYPER = 0x00000000;
74 #ifdef USE_PRIORITIES
75     SET_SVC_PRIORITY(0xF);
76     SET_SYSTICK_PRIORITY(3);
77 #else
78     SET_SVC_PRIORITY(0);
79     SET_SYSTICK_PRIORITY(0);
80 #endif
81
82     *((void (**)(void)) 0x2001C038) = svc_irq_handler;
83     *((void (**)(void)) 0x2001C03C) = systick_irq_handler;
84
85     while( 1 )
86     {
87         out7seg( 0 );
88         SET_SVC_IRQ_PENDING;
89         out7seg( 4 );
90     }
91 }
92
Re: Ln 72, Col 0 SPACES LF C++ UTF-8
```

# Avbrottsprioritet



*Här ska sifferföljden: 0,1,2,3,4 skrivas till 7-segmentsdisplayen*

The screenshot shows an IDE window with the following code in `priorities.c`:

```

58 SET_SYSTICK_IRQ_PENDING; /* fake SYSTICK interrupt */
59 out7seg( 3 );
60
61 }
62
63 void systick_irq_handler( void )
64 {
65     out7seg( 2 );
66 }
67
68
69 #define USE_PRIORITIES
70 void main(void)
71 {
72     *GPIO_D_MODER = 0x00005555;
73     *GPIO_D_OTYPER= 0x00000000;
74 #ifdef USE_PRIORITIES
75     SET_SVC_PRIORITY(0xF);
76     SET_SYSTICK_PRIORITY(3);
77 #else
78     SET_SVC_PRIORITY(0);
79     SET_SYSTICK_PRIORITY(0);
80 #endif
81
82     *((void (**)(void) ) 0x2001C038 ) = svc_irq_handler;
83     *((void (**)(void) ) 0x2001C03C ) = systick_irq_handler;
84
85     while( 1 )
86     {
87         out7seg( 0 );
88         SET_SVC_IRQ_PENDING;
89         out7seg( 4 );
90     }
91 }
92
93
    
```

On the right side of the IDE, there is a window showing a 7-segment display module labeled "PD0-7". The display is currently showing the digit "8". The module includes a 74HC595 shift register and a 7-segment display.

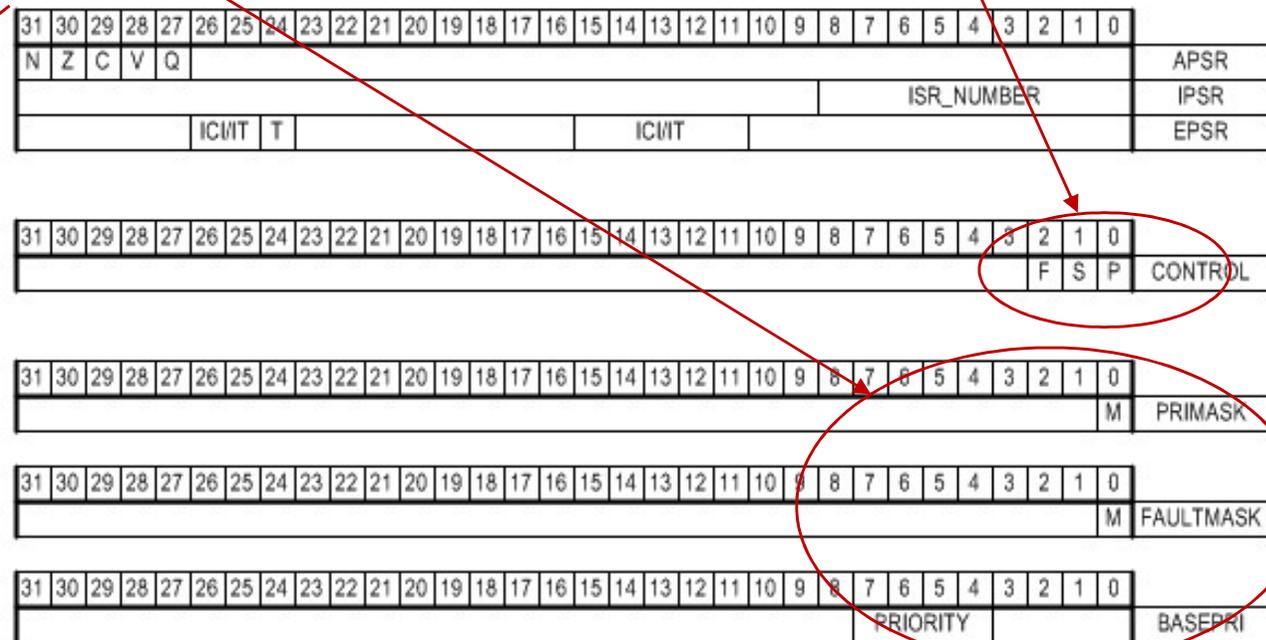
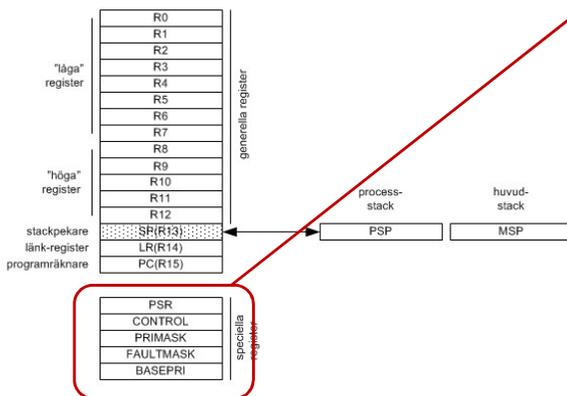
# Speciella register

används för att ange:

- Om flyttalsprocessor ska aktiveras
- Vilken stack som används
- Vilket exekveringstillstånd som används
- Om undantag/avbrott ska betjänas

**Exempel:**

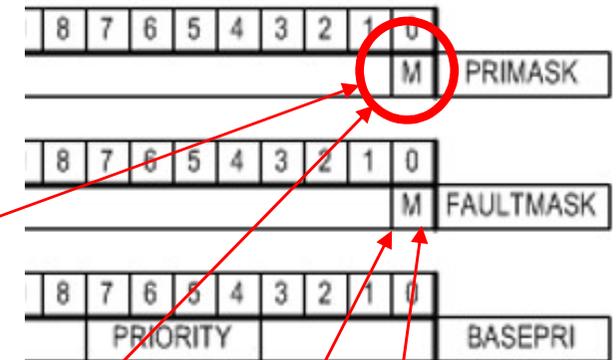
```
void __setCONTROL( unsigned int val)
{
    __asm(
        " MSR CONTROL, R0\n"
    );
}
```



# Maskera undantag

Man kan undvika problematiken med avbrottsprioriteter:

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



<pre>void disable_interrupt( void ) {     __asm(         " CPSID I\n"     ); }</pre>	<pre>void enable_interrupt( void ) {     __asm(         " CPSIE I\n"     ); }</pre>
--	---

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

```
void irq_handler( void )
{
    disable_interrupt();
    ....
    enable_interrupt();
}
```

```
void disable_fault( void )
{
    __asm(
        " CPSID F\n"
    );
}
```

```
void enable_fault( void )
{
    __asm(
        " CPSIE F\n"
    );
}
```

# Processorns olika tillstånd

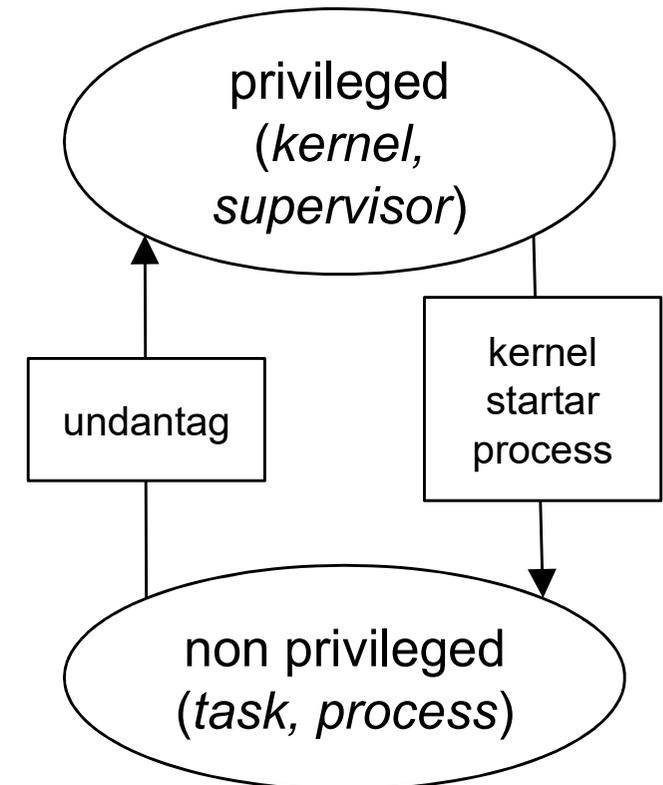
- **Handler** (undantagshantering)/**Thread mode** (normal exekvering), sköts automatiskt av processorn.

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.

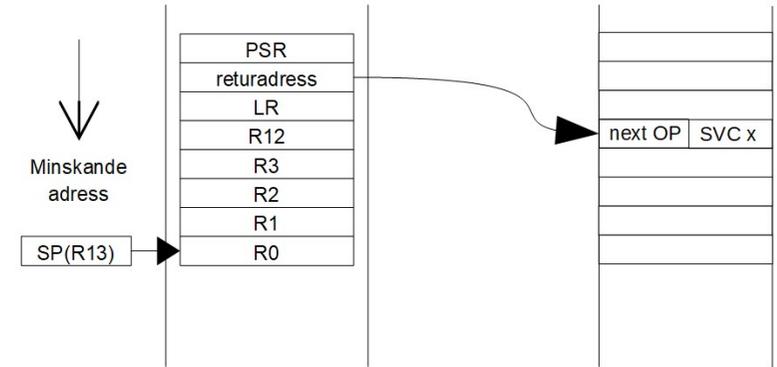
Operativsystemets programvara (*kernel*) exekveras med alla resurser tillgängliga (priviligierat).

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



# SVCall (SuperVisor Call)

		Reserved	0x0000 001C -1
-5	settable	SVCall	System service call via SWI instruction
-4	settable	Debuu Monitor	Debuu Monitor
			0x0000 002C
			0x0000 0030



En software trap instruktion för att utföra inbyggda funktioner.

- Ger en kontrollerad växling mellan *non-privileged* och *privileged* mode.
- Kompakt sätt att utföra funktion

@ SVC, SuperVisorCall  
operationskoden är 0xDFxx, där xx utgör konstanten

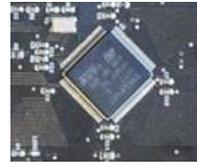
```
__attribute__((naked))
void graphic_initialize(void)
{
    __asm volatile(" .HWORD 0xDFF0\n");
    __asm volatile (" BX LR\n");
}
```

```
static __attribute__((naked))
void svcHandler( void )
{
    __asm volatile(" MOV R0,SP\n");
    __asm volatile(" B  os_call\n");
    __asm volatile(" .ALIGN 2\n");
}
```

```
void _graphic_initialize( void );
void _graphic_clear_screen( int x, int y);
void _graphic_pixel_set( int x, int y);
void _graphic_pixel_clear( int x, int y);
```

```
void os_call( unsigned int * call_args)
{
    unsigned int oscall = ((char *) call_args[6]) [-2];
    switch( oscall )
    {
        case 0xF0: _graphic_initialize(); break;
        case 0xF1: _graphic_clear_screen(); break;
        case 0xF2: _graphic_pixel_set(call_args[0],call_args[1] )break;
        case 0xF3: _graphic_pixel_clear(call_args[0],call_args[1] )break;
    }
}
```

# Interna avbrott



**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

LQFP64 (10 x 10 mm)  
LQFP100 (14 x 14 mm)  
LQFP144 (20 x 20 mm)  
LQFP176 (24 x 24 mm)

WLCSP90

LFBGA176 (10 x 10 mm)

**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 Controller
- Up to 140 I/O ports
  - Up to 136 fast I/Os
  - Up to 138 5 V I/Os
- Up to 15 comm. interfaces
  - Up to 3 x I<sup>2</sup>C
  - Up to 4 USARTs (7816 interface)
  - Up to 3 SPIs (full-duplex I<sup>2</sup>S accuracy via internal clock)
  - 2 x CAN interfaces
  - 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**

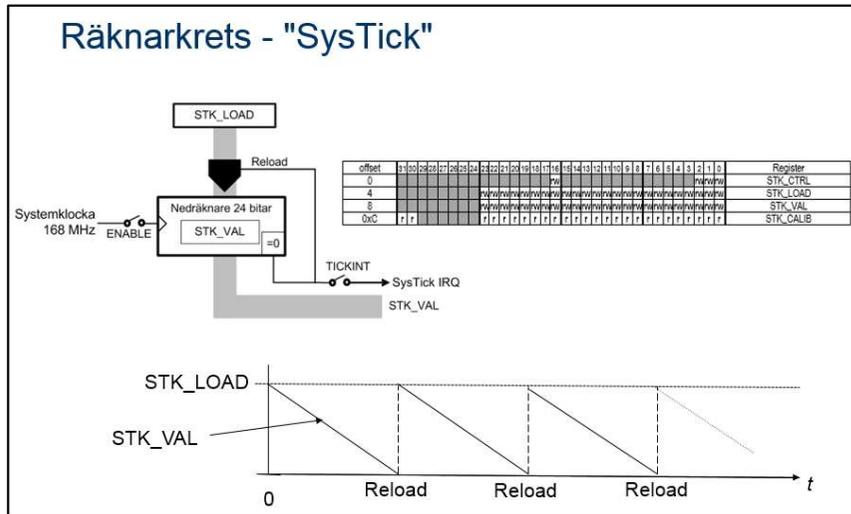
Reference	Part number
STM32F405xx	STM32F405RG, STM32F405VG, STM32F405ZG, STM32F405OG, STM32F405CE
	STM32F407VG, STM32F407IG, STM32F407ZG, STM32F407VE, STM32F407ZE, STM32F407IE
	Temperature sensor
	ADC1, ADC2, ADC3

Externa avbrott –  
via portpinnar från  
kretsar utanför  
en-chipsdatorn.

Interna avbrott –  
från andra kretsar i  
en-chipsdatorn.



# "SysTick" med avbrott...



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 tills fördröjningen, dvs. COUNTFLAG blir 1, är klar....

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, då fördröjningen är klar, dvs. COUNTFLAG är 1, genereras ett avbrott.

# Icke-blockerande fördröjning med SysTick

Skapa en fördröjningsfunktion `delay(unsigned int u)` som skapar `u` mikrosekunders fördröjning.

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.

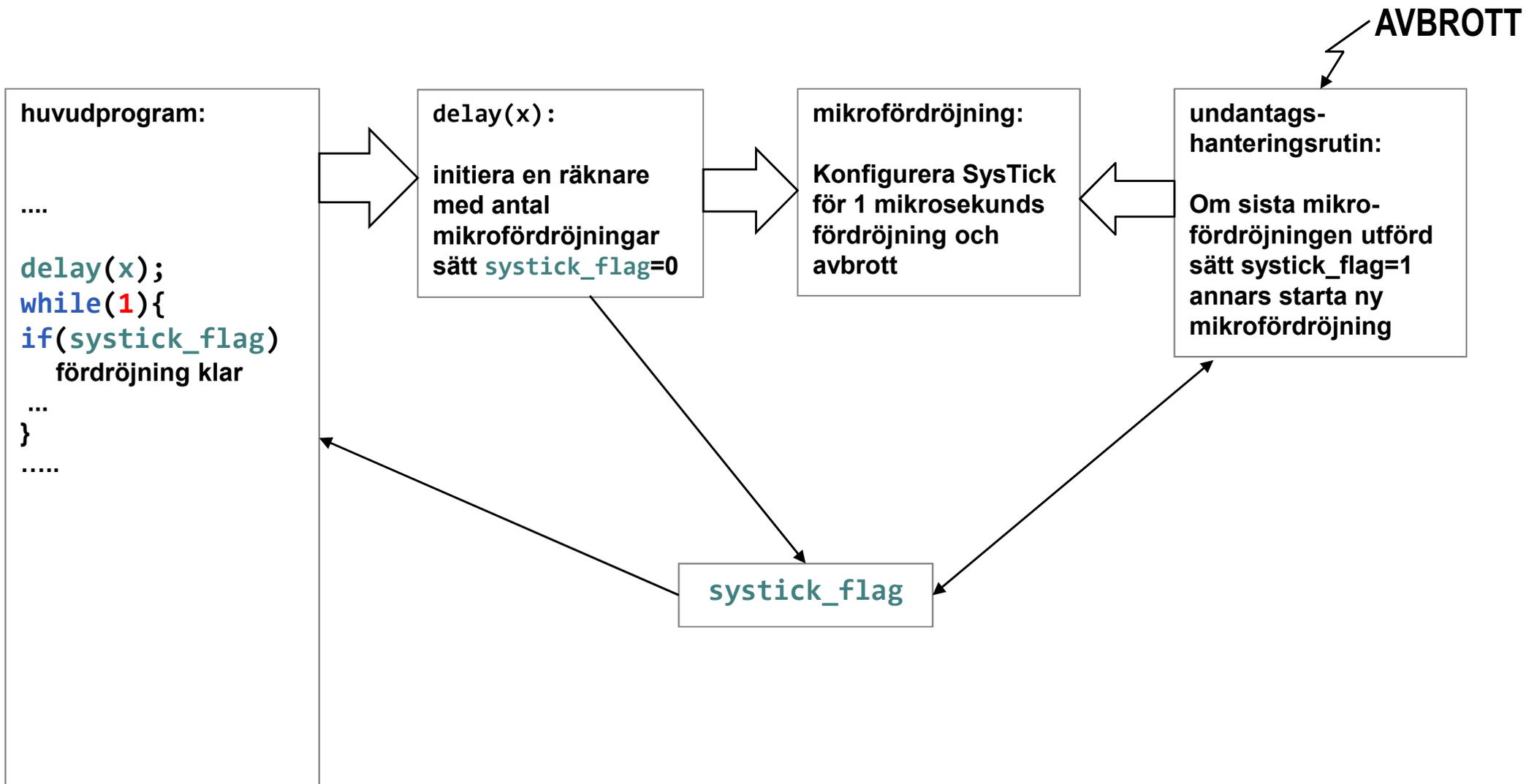
En annan diodramp ska visa en kontinuerligt uppräknad variabel

Testprogrammet:

```
#define DELAY_COUNT 1000
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_HIGH = c;
        c++;
    }
    *GPIO_ODR_LOW = 0;
}
```

# Icke-blockerande fördröjning

Skiss av programdelar



# Icke-blockerande fördröjning

Implementering

mikrofördröjning:

Konfigurera SysTick  
för 1 mikrosekunds  
fördröjning och avbrott

undantags-  
hanteringsrutin:

Om sista mikro-  
fördröjningen utförd  
sätt `systick_flag=1`  
annars starta ny  
mikrofördröjning

`delay(x)`:

initiera en räknare  
med antal  
mikrofördröjningar  
sätt `systick_flag=0`

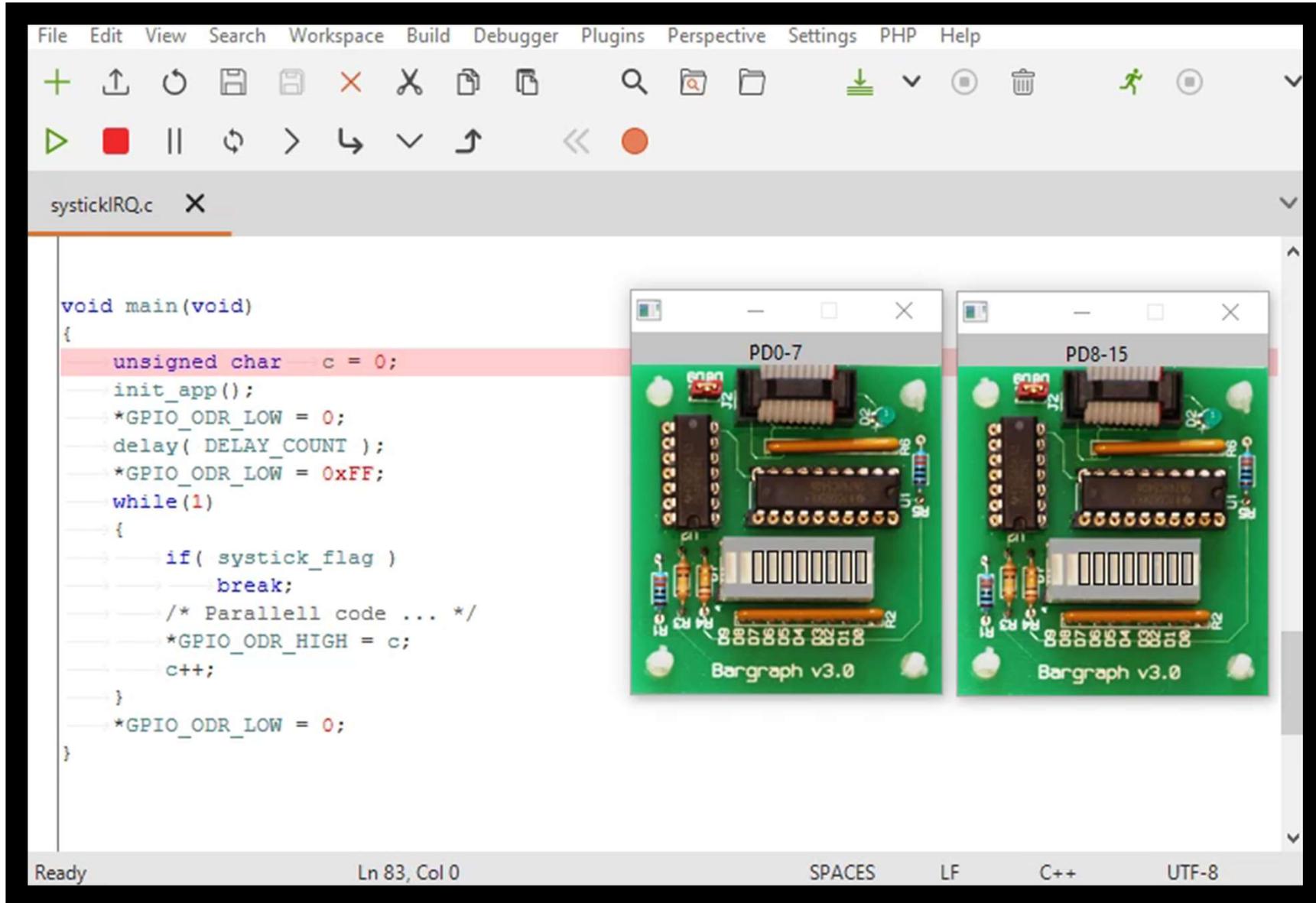
```
void delay_1mikro( void )
{
    *STK_CTRL = 0;
    *STK_LOAD = ( 168 - 1 );
    *STK_VAL = 0;
    *STK_CTRL = 7;
}

static volatile int  systick_flag;
static volatile int  delay_count;

void systick_irq_handler( void )
{
    *STK_CTRL = 0;
    delay_count -- ;
    if( delay_count > 0 ) delay_1mikro();
    else systick_flag = 1;
}

void delay( unsigned int count )
{
    if( count == 0 ) return;
    delay_count = count;
    systick_flag = 0;
    delay_1mikro();
}
```

# Icke-blockerande fördröjning med SysTick



The screenshot shows an IDE window titled 'systickIRQ.c' with the following C++ code:

```

void main(void)
{
    unsigned char c = 0;
    init_app();
    *GPIO_ODR_LOW = 0;
    delay( DELAY_COUNT );
    *GPIO_ODR_LOW = 0xFF;
    while(1)
    {
        if( systick_flag )
            break;
        /* Parallell code ... */
        *GPIO_ODR_HIGH = c;
        c++;
    }
    *GPIO_ODR_LOW = 0;
}
    
```

Two hardware diagrams of the 'Bargraph v3.0' board are overlaid on the code. The left diagram is labeled 'PD0-7' and the right diagram is labeled 'PD8-15'. Both diagrams show a green PCB with a central 8-bit shift register (74HC595), a 7-segment display, and various passive components like resistors and capacitors. The board is populated with components and has a USB connector at the bottom.

The IDE status bar at the bottom indicates: Ready, Ln 83, Col 0, SPACES, LF, C++, UTF-8.