

Undantagshantering och interna avbrott

Ur innehållet:

- Faults
- Software traps
- Undantagsprioriteter
- Avbrott från interna systemenheter, ("Systick")

Läsanvisningar:

- Arbetsbok kap 6.1, 6.2, 6.4, 6.5

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?

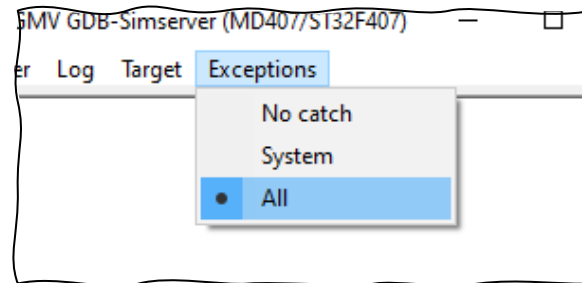
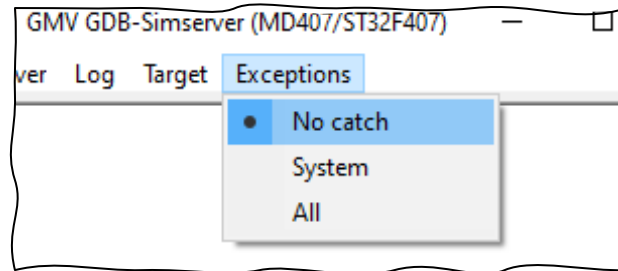
```
@ exception_unhandled.asm

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
```



Eterm8 [MD407/ST32F407]

File Edit View Tools Help

Navigator

- Files
 - Assembler source
 - Loadfile (.s19)
 - Listfile (.lst)
 - Other files

Start Page exception_unhandled.asm

@ exception_unhandled.asm

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

Debugger

Registers

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	SP	LR	PC	PSR	FPSR	MSP	PSP	PRIMASK	FAULTMASK	BASEPRI	CONTROL
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	2001C000	FFFFFFF	20000000	01000000	00000000	2001C000	00000000	00000000	00000000	00000000	00000000

CPU control/Options

RESET

☐ Break on IO

☐ Break on IRQ

Execution control/options

Disassembly

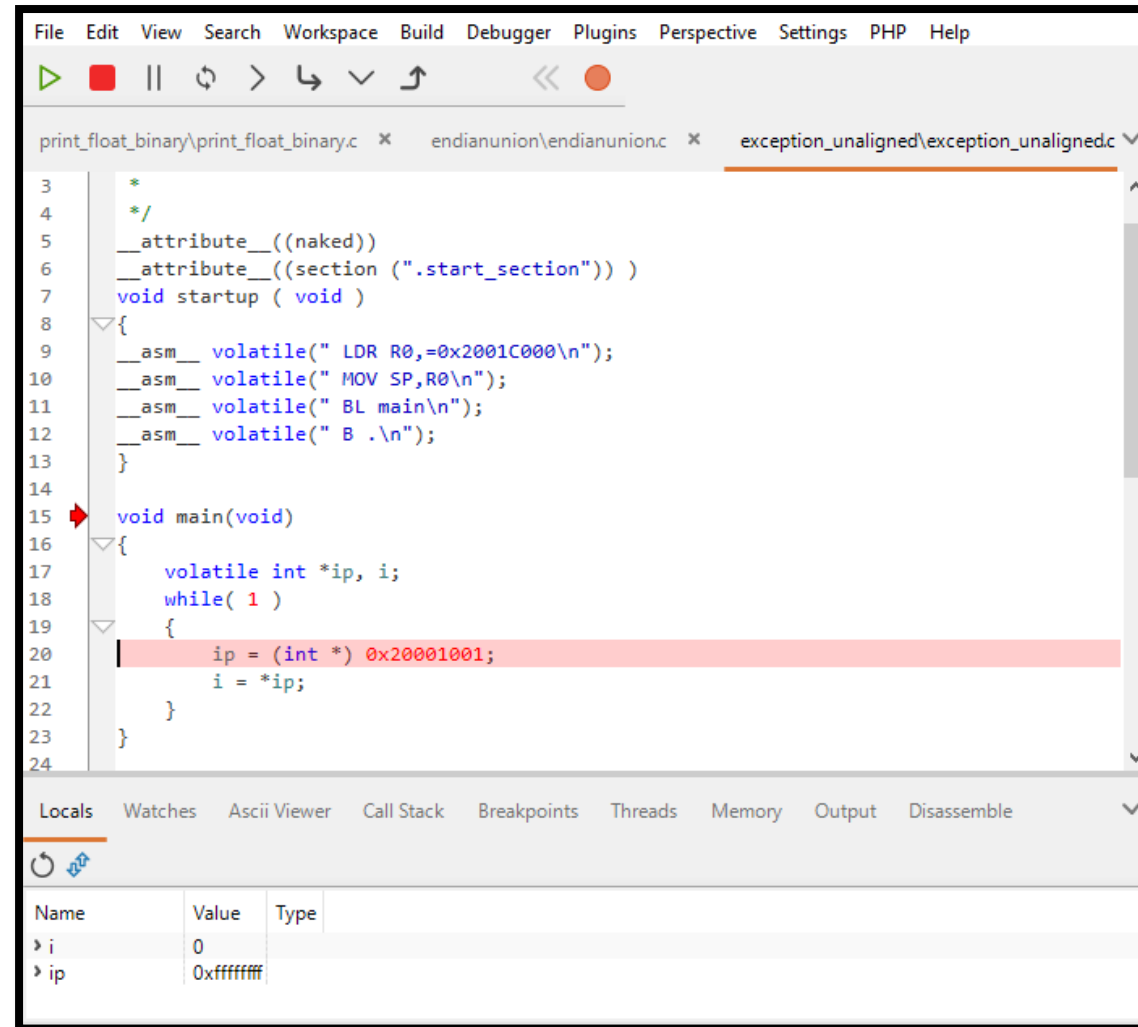
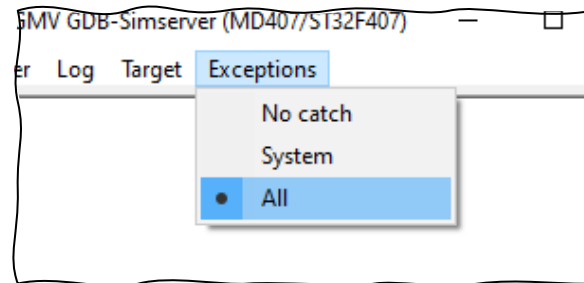
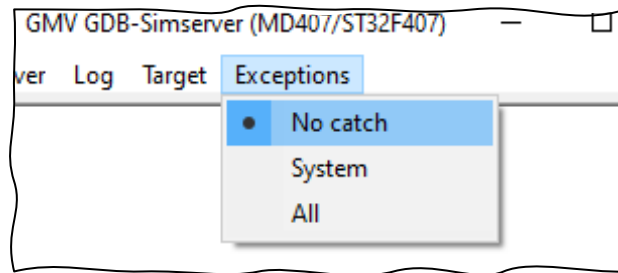
Scroll into view

	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

Output Console

Connected to server

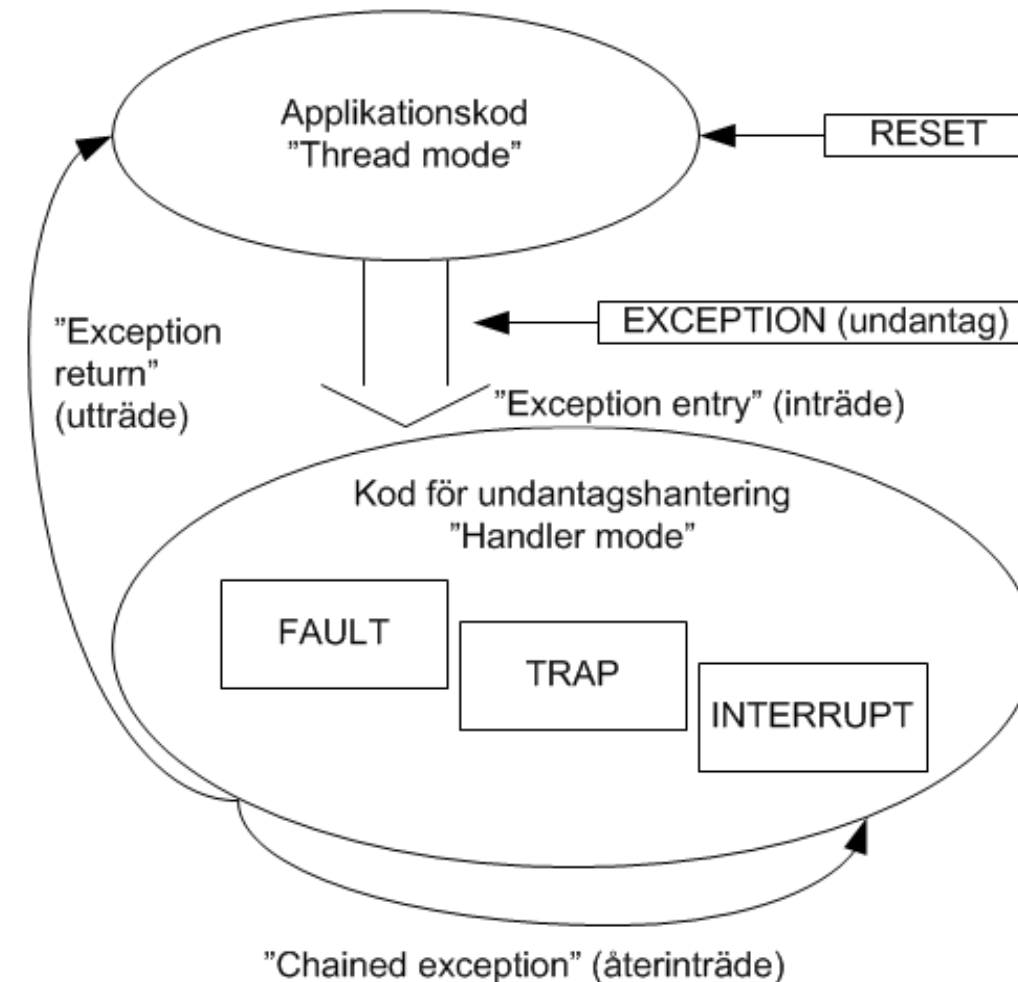
```
void main(void)
{
    int *ip, i;
    ip = (int *) 0x20001001;
    i = *ip;
}
```



Undantagshantering - "exception"

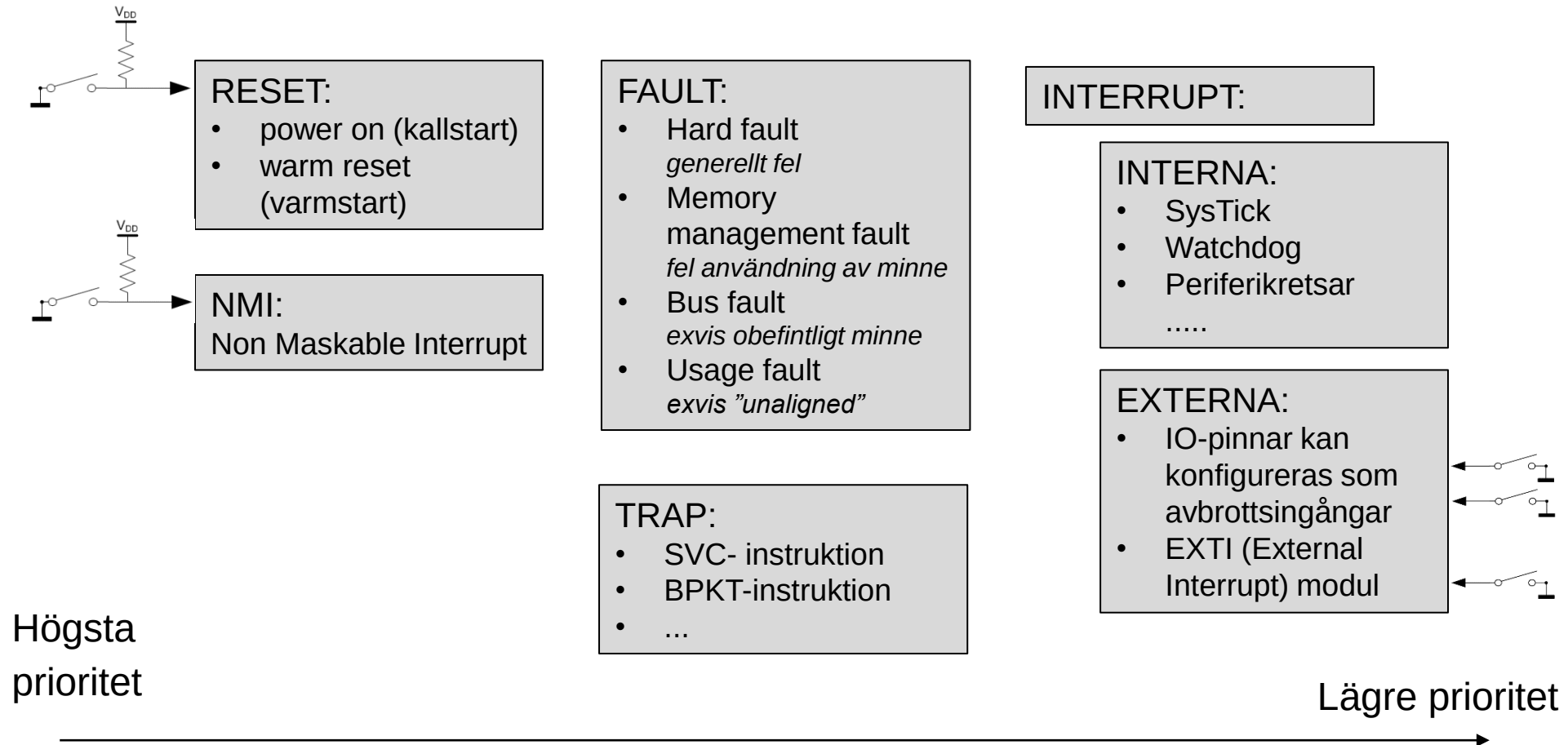
Med "undantag" menar vi olika typer av händelser:

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

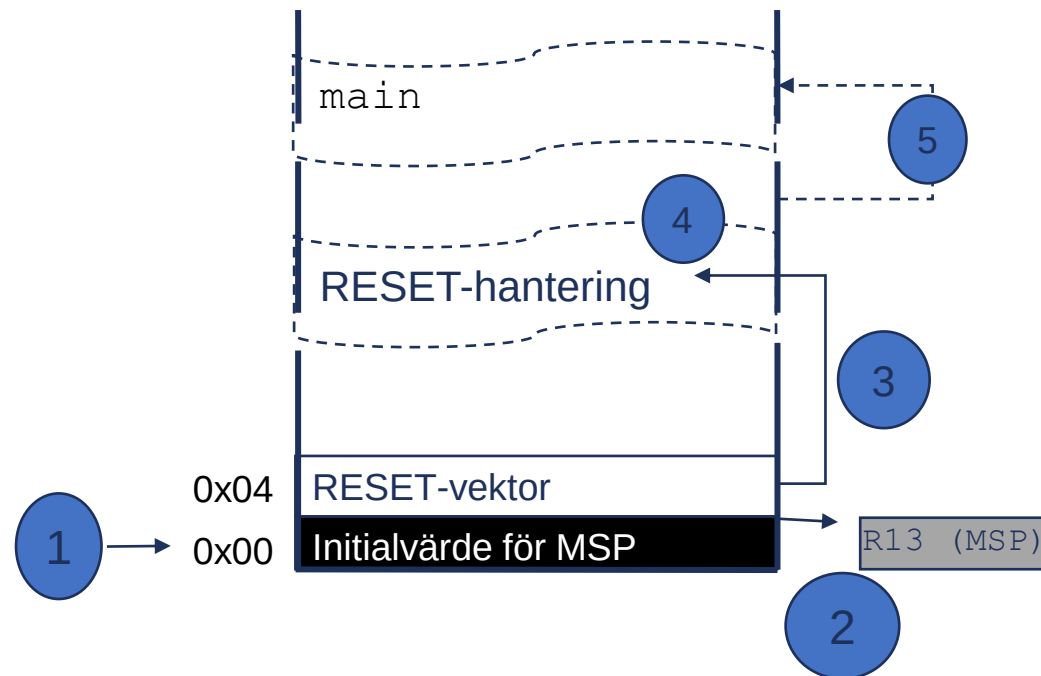


Undantagstyper

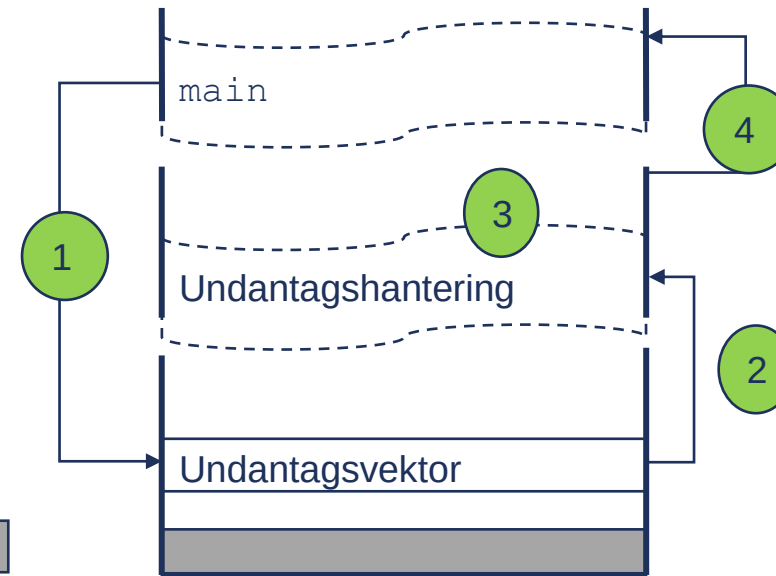
De olika undantagstyperna har en naturlig, fallande prioritet



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.

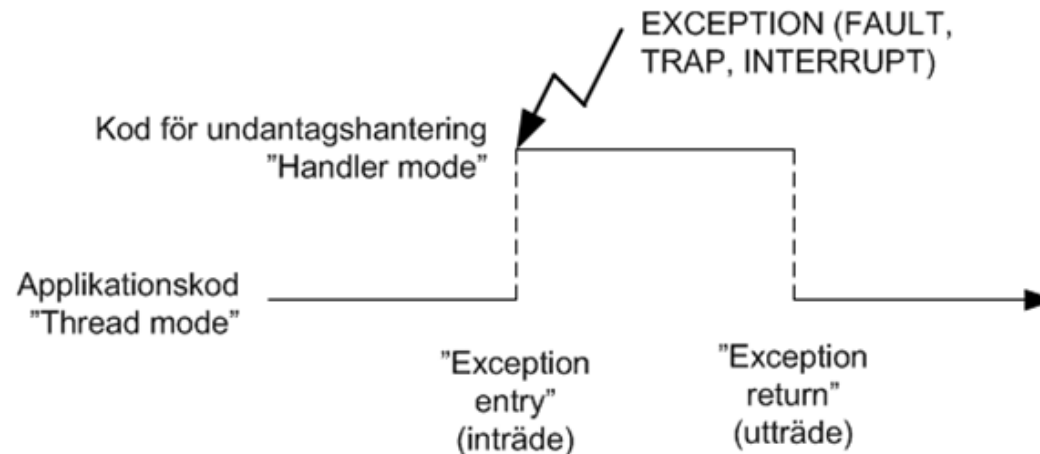
De 16 första positionerna är samma för alla Cortex-M4.

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)



Inträde:

Spara aktuell processorstatus (“save context”), dvs.

- Spara registerinnehåll på stacken
- Sätt nytt speciellt innehåll i LR (EXC_RETURN) baserat på processorstatus

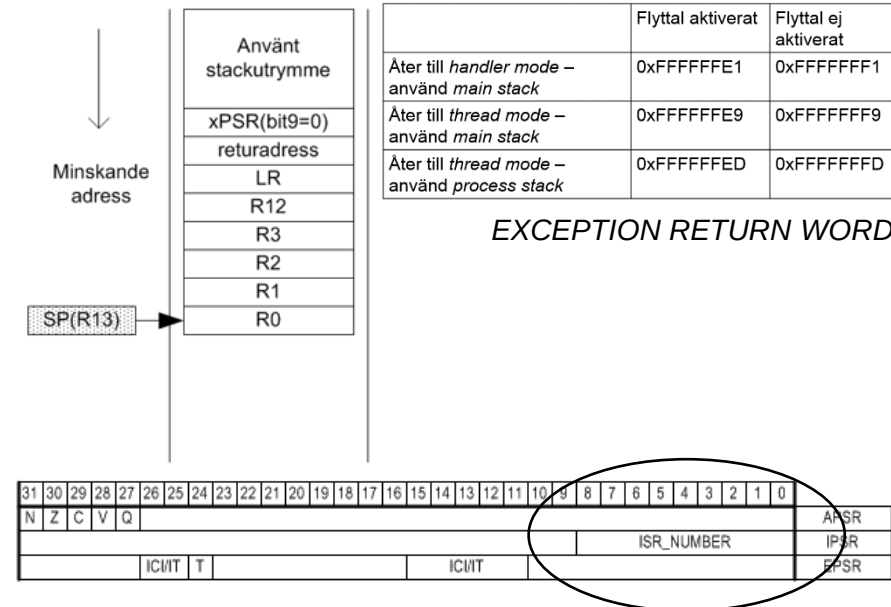
Ändra processorstatus för undantagshantering

- Sätt undantagsnummer i PSR
- Placera undantagsvektor i PC

Utträde:

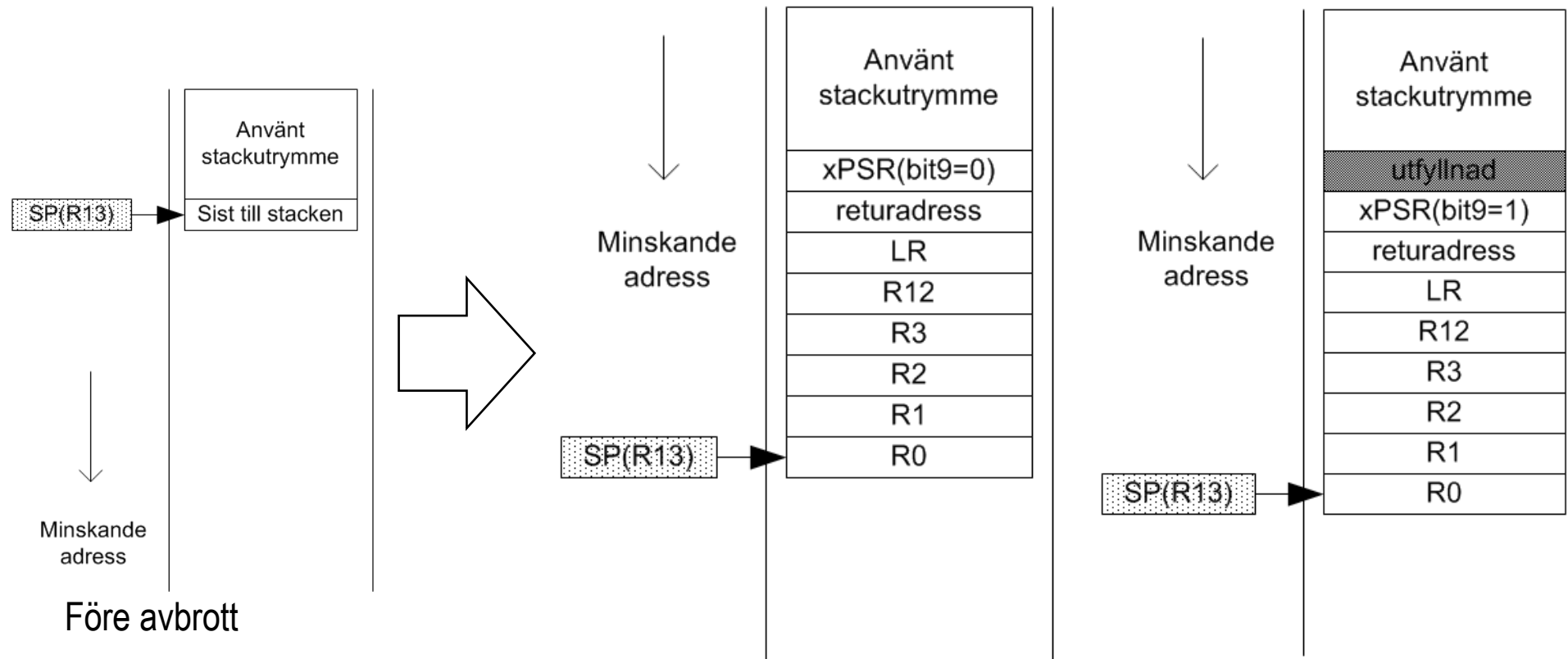
Återställ avbruten processorstatus (“restore context”)

Återställ PC. Det speciella EXC_RETURN värdet som då hamnar i PC avkodas.
 Registerinnehåll återställs då från stacken varvid PC får rätt återhoppadress.



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>	0xFFFFFEE1	0xFFFFFFF1
Åter till <i>thread mode</i> – använd <i>main stack</i>	0xFFFFFEE9	0xFFFFFFF9
Åter till <i>thread mode</i> – använd <i>process stack</i>	0xFFFFFEEB	0xFFFFFFF3

- ☐ 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

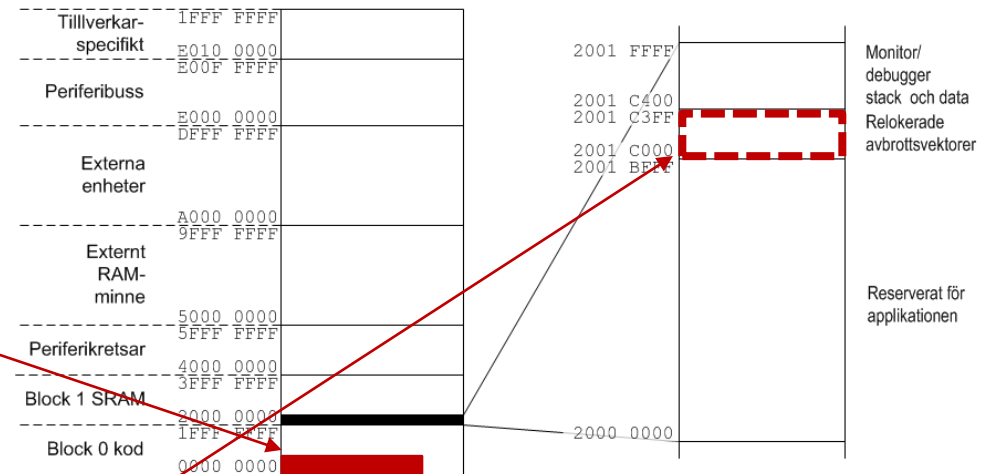
- ☐ Om avbrott avvaktar:

- 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;
```

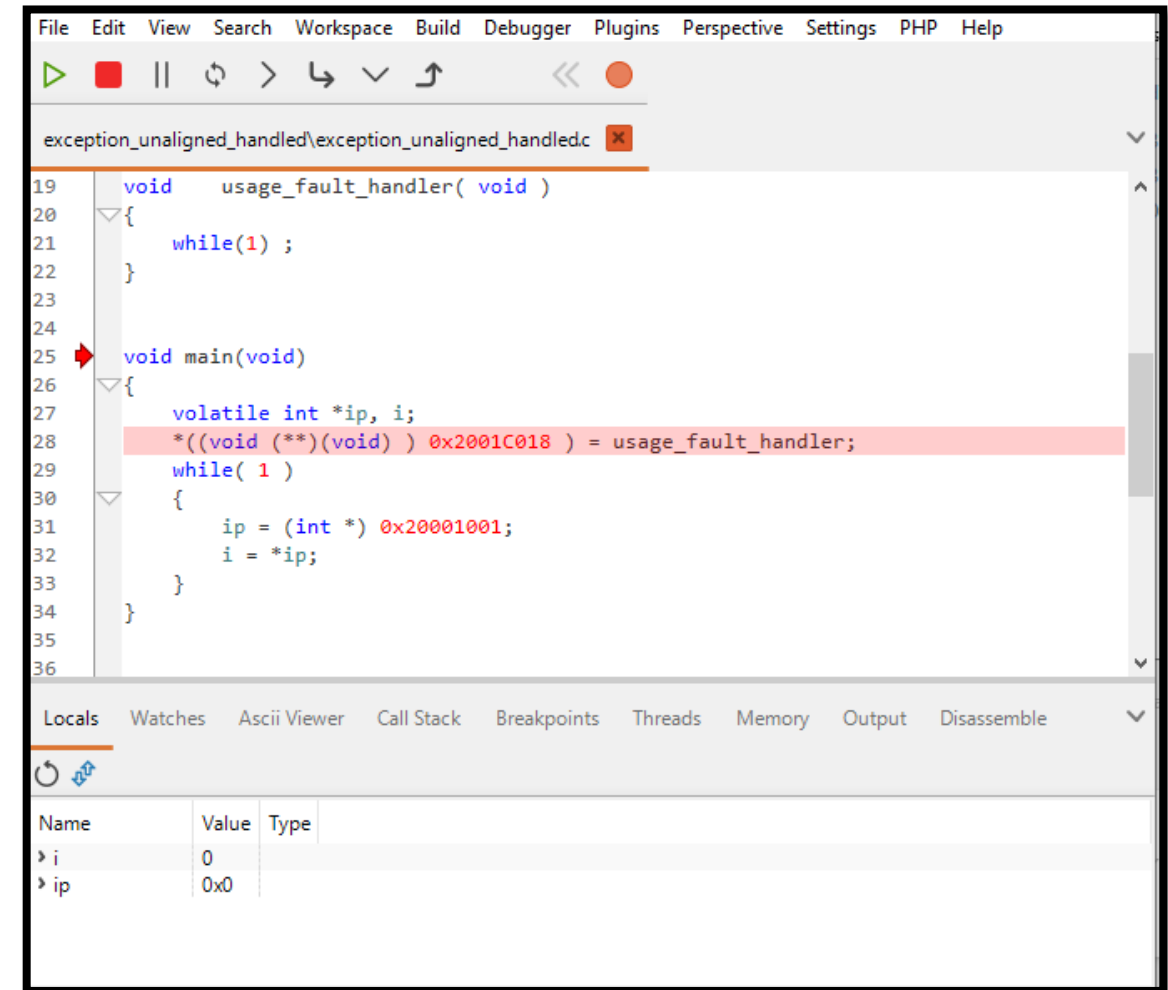
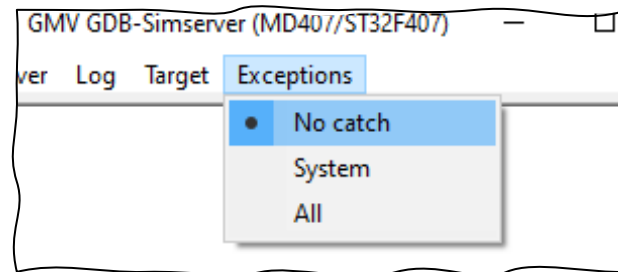
Exempel:
Initiera avbrottsvektor
"usage fault"

0	settable	Memory management	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	0x0000 002C

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

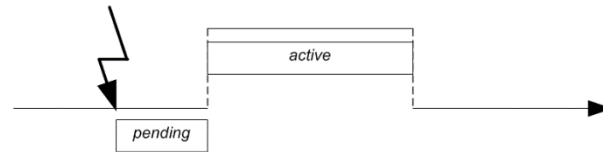
```
void usage_fault_handler( void )
{
    while(1) ;
}

void main(void)
{
    volatile int *ip, i;
    *((void (**)(void) ) 0x2001C018 ) = usage_fault_handler;
    while( 1 )
    {
        ip = (int *) 0x20001001;
        i = *ip;
    }
}
```

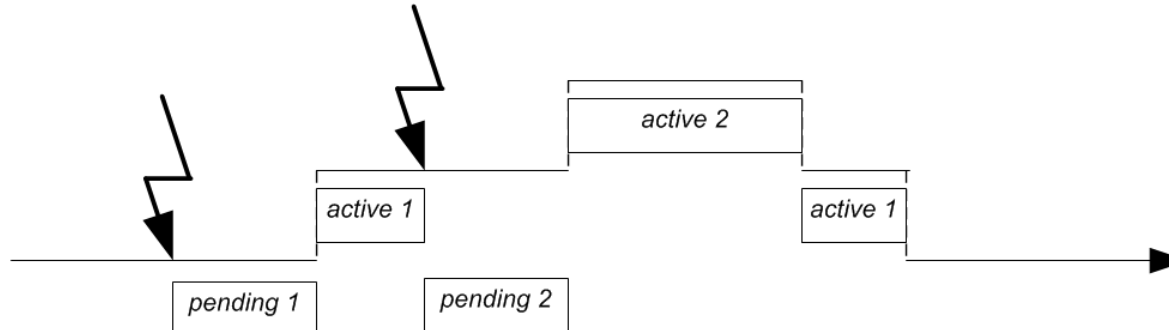


Avbrottsprioritet

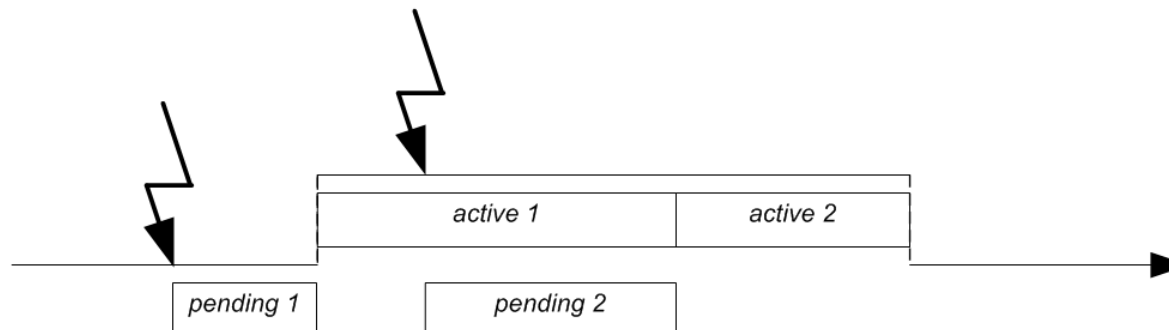
Flera samtidiga avbrott:
pending, *active* och *prioriteter*



Ett avbrott uppträder och betjänas



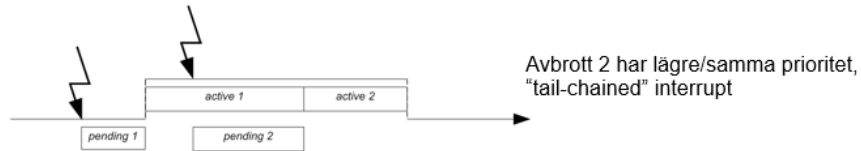
Avbrott 2 har högre prioritet,
ytterligare “context switch”



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

Vi demonstrerar med simulator

Avbrottsprioritet



```
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 );
    }
}
```

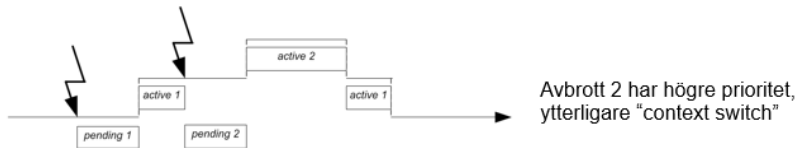
```
File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help
priorities\priorities.c
// #define USE_PRIORITIES
void main(void)
{
    *GPIO_D_MODER = 0x00005555;
    *GPIO_D_OTYPER = 0x00000000;
    #ifdef USE_PRIORITIES
        SET_SVC_PRIORITY(0xF);
        SET_SYSTICK_PRIORITY(3);
    #else
        SET_SVC_PRIORITY(0);
        SET_SYSTICK_PRIORITY(0);
    #endif

    *((void (*)(void)) 0x2001C038) = svc_irq_handler;
    *((void (*)(void)) 0x2001C03C) = systick_irq_handler;

    while( 1 )
    {
        out7seg( 0 );
        SET_SVC_IRQ_PENDING;
        out7seg( 4 );
    }
}
```

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

Avbrottsprioritet



```
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(0xF);
    SET_SYSTICK_PRIORITY(3);
    while( 1 )
    {
        out7seg( 0 );
        SET_SVC_IRQ_PENDING;
        out7seg( 4 );
    }
}
```

```
File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help
priorities\priorities.c
#define USE_PRIORITIES
void main(void)
{
    *GPIO_D_MODER = 0x00005555;
    *GPIO_D_OTYPER = 0x00000000;
    #ifdef USE_PRIORITIES
        SET_SVC_PRIORITY(0xF);
        SET_SYSTICK_PRIORITY(3);
    #else
        SET_SVC_PRIORITY(0);
        SET_SYSTICK_PRIORITY(0);
    #endif

    *((void (*)(void)) 0x2001C038) = svc_irq_handler;
    *((void (*)(void)) 0x2001C03C) = systick_irq_handler;

    while( 1 )
    {
        out7seg( 0 );
        SET_SVC_IRQ_PENDING;
        out7seg( 4 );
    }
}
```

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

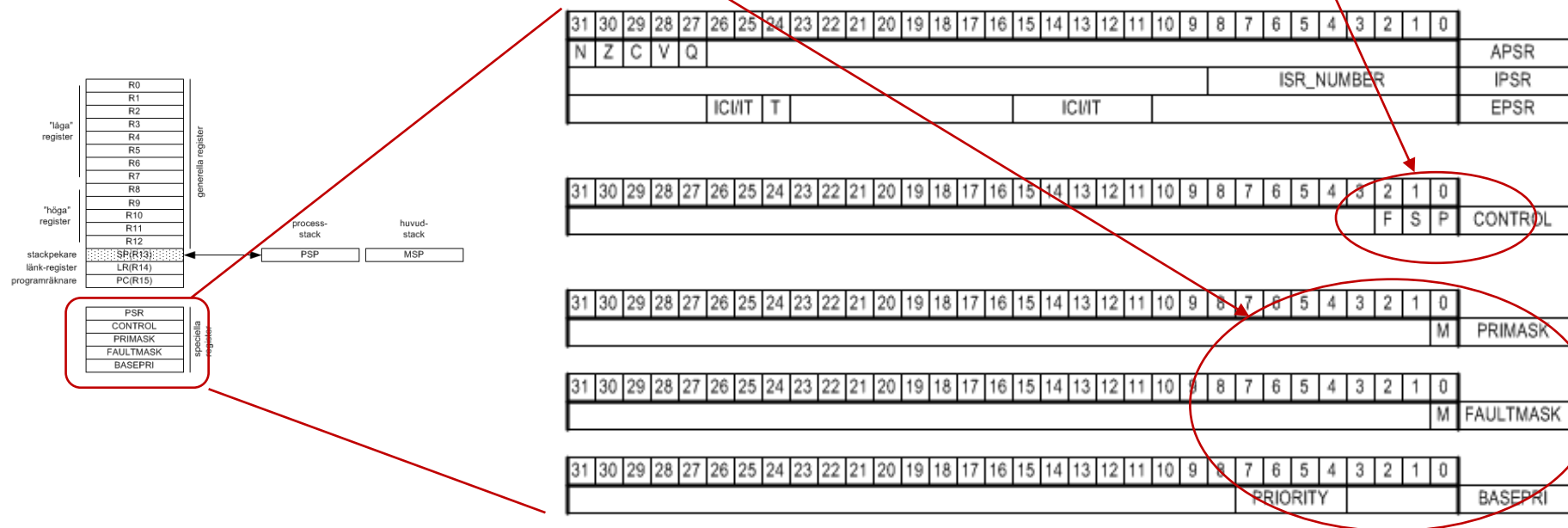
Speciella register

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__ volatile (" 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.



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

```
void enable_interrupt( void )
{
    __asm__ volatile(" CPSIE I\n");
}
```

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__ volatile (" CPSID F\n");
}
```

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

Processorns olika tillstånd

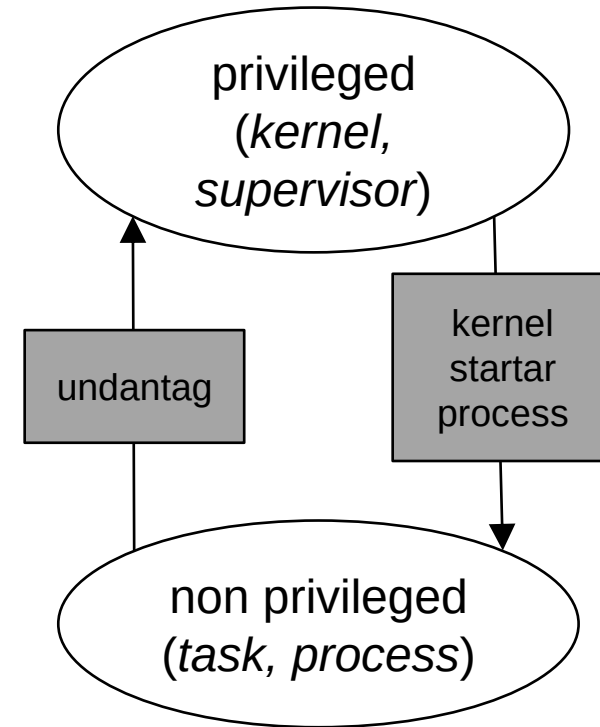
- **Handler** (undantagshantering)/**Thread mode** (normal exekvering), sköts automatiskt av processorn.

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 begränsad miljö (icke privilegierat), övervakad av *kernel*.

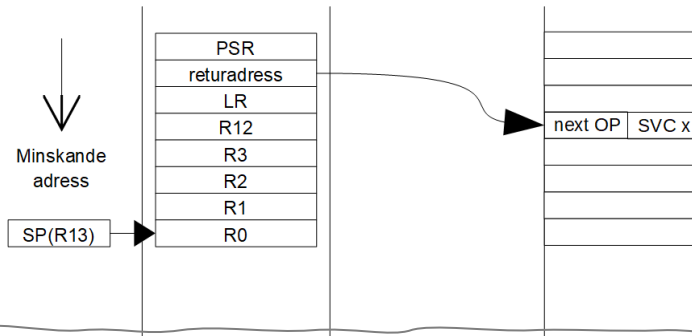


SVCall (SuperVisor Call)

			Reserved	0x0000 001C - 1
-5	settable	SVCall	System service call via SWI instruction	0x0000 002C
-4	settable	Debug Monitor	Debug Monitor	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 × 10 mm)
LQFP100 (14 × 14 mm)
LQFP144 (20 × 20 mm)
LQFP176 (24 × 24 mm)
WLCSP90
LQFP176 (10 × 10 mm)

Features

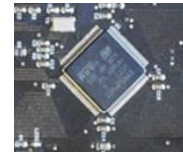
- Core: ARM 32-bit Cortex™-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 168 MHz, memory protection unit, 210 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
 - Up to 1 Mbyte of Flash memory
 - 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_{BAT} supply for RTC, 20×32 bit backup registers + optional 4 KB backup SRAM
- 3×12-bit, 2.4 MSPS A/D converters: up to 24 channels and 7.2 MSPS in triple interleaved mode
- 2×12-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

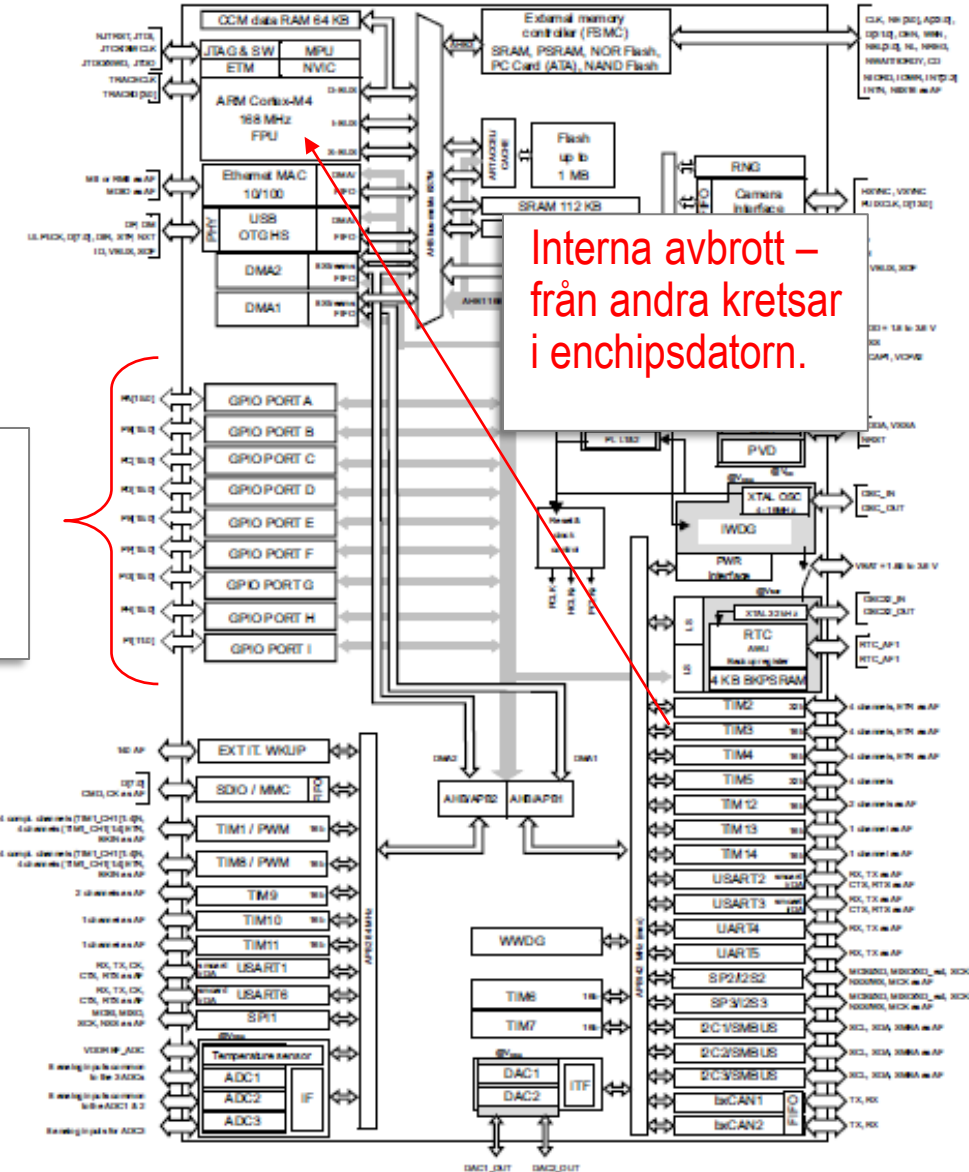
- Debug mode
 - Serial wire debug (SWD) & JTAG interfaces
 - Cortex-M4 Embedded Trace Macrocell™
- Up to 140 I/O ports with interrupt capability
 - Up to 136 fast I/Os up to 84 MHz
 - Up to 138 5 V-tolerant I/Os
- Up to 15 communication interfaces
 - Up to 3 × I²C interfaces (SMBus/PMBus)
 - Up to 4 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 I²S to achieve audio class accuracy via internal audio PLL or external clock
 - 2 × 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 ULPI
 - 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, STM32F405VQ, STM32F405ZG, STM32F405QG, STM32F405OE
STM32F407xx	STM32F407VG, STM32F407VQ, STM32F407ZG, STM32F407VE, STM32F407ZE, STM32F407IE

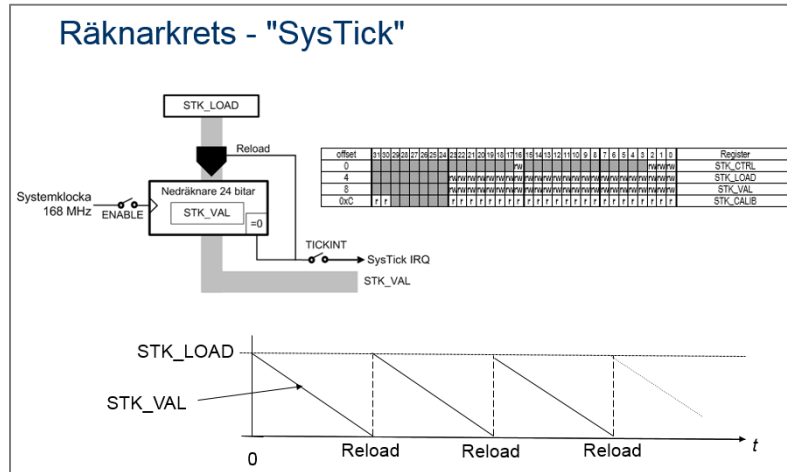


Externa avbrott – via portpinnar från kretsar utanför enchipsdatorn.



MS19020V3

"SysTick" med avbrott...



Algorit:

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

Algorit:

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.

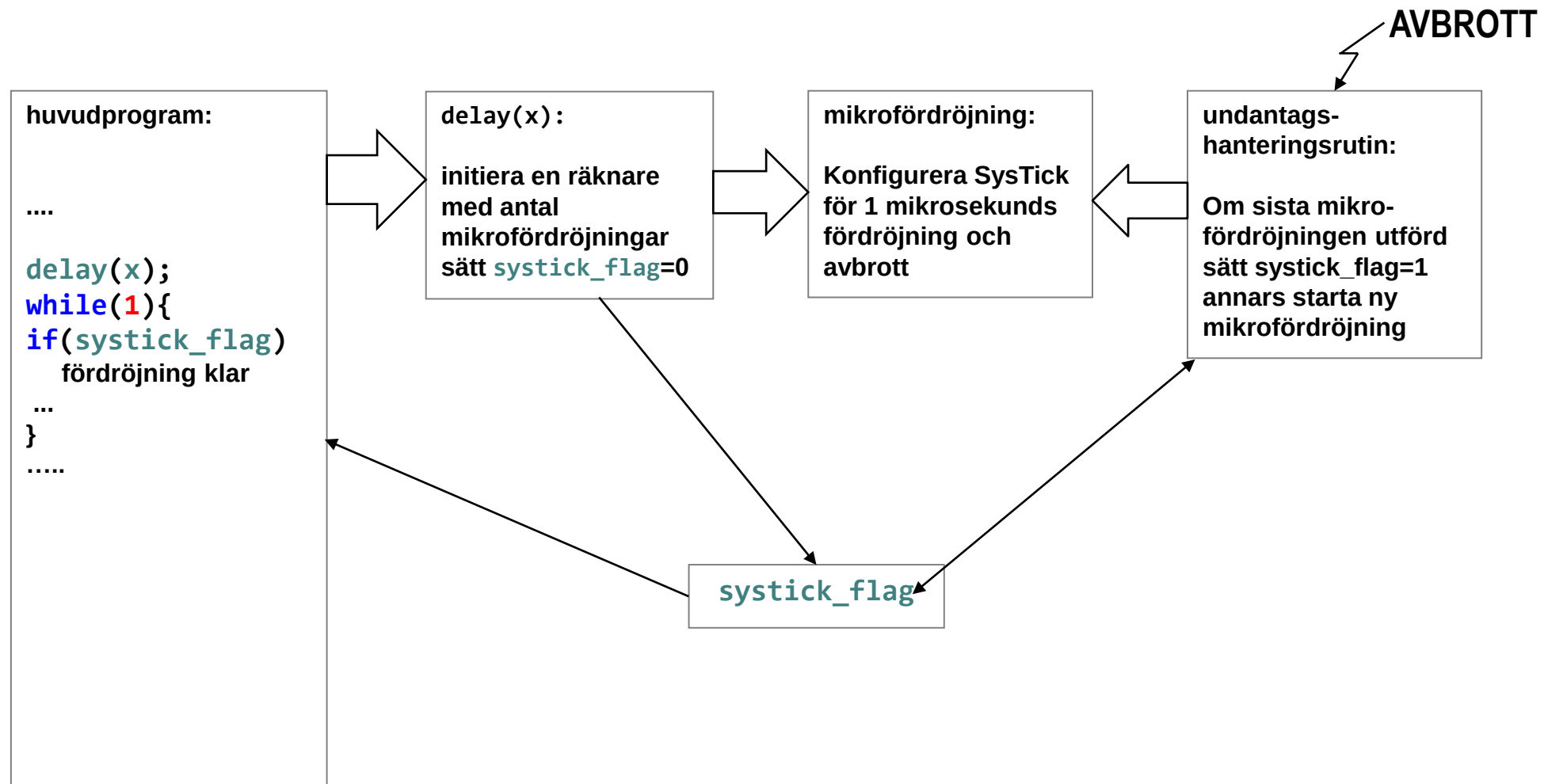
En annan diodramp ska visa en
kontinuerligt uppräknad variabel

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

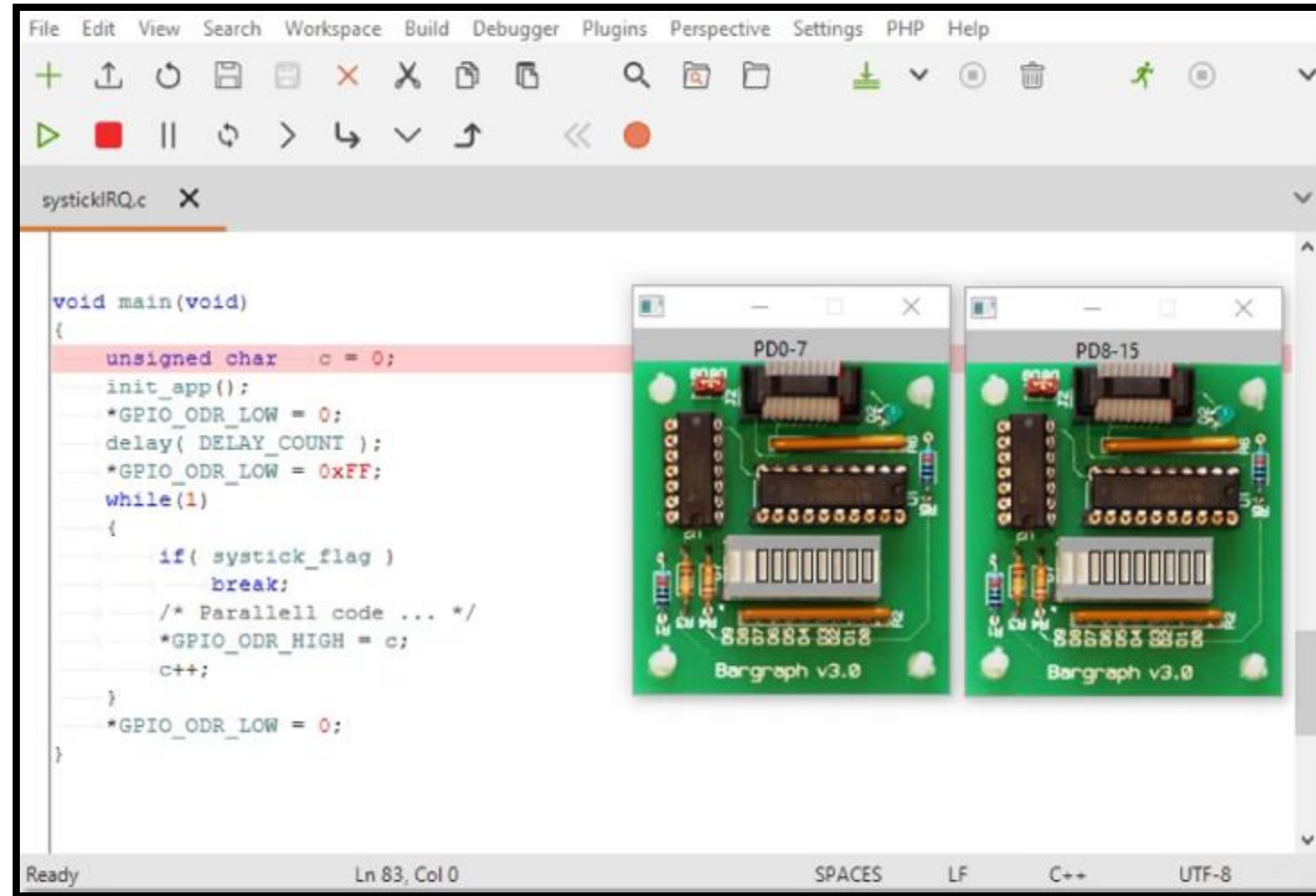
```
void delay_1mikro( void )
{
    *STK_CTRL = 0;
    *STK_LOAD = ( 168 - 1 );
    *STK_VAL = 0;
    *STK_CTRL = 7;
}

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'. The code is as follows:

```
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 circuit diagrams are overlaid on the code. The left diagram, labeled 'PD0-7', shows a green PCB with a microcontroller, a 10-pin header, and a 7-pin header. The right diagram, labeled 'PD8-15', shows a similar PCB with a 15-pin header. Both diagrams are labeled 'Banggraph v3.0' at the bottom.