Interrupts II

ECE 3710

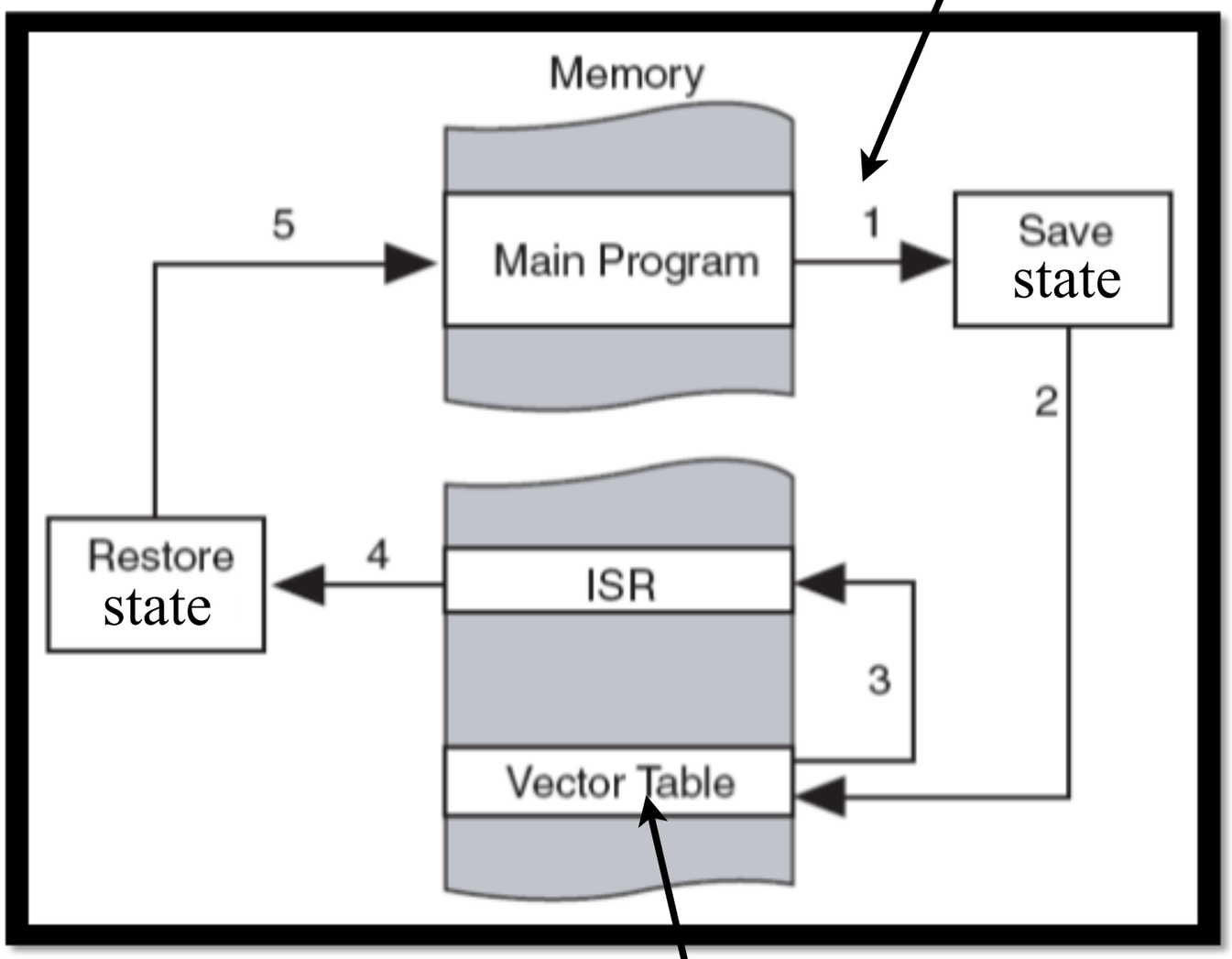
In Vegas, I got into a long argument with the man at the roulette wheel over what I considered to be an odd number.

- Steven Wright

when an interrupt occurs

(uC is doing something)

interrupt here



1. push state onto stack
2. uC looks up address of routine associated with that interrupt
3. PC set to that routine
4. routine finishes: original state restored

each interrupt has own entry

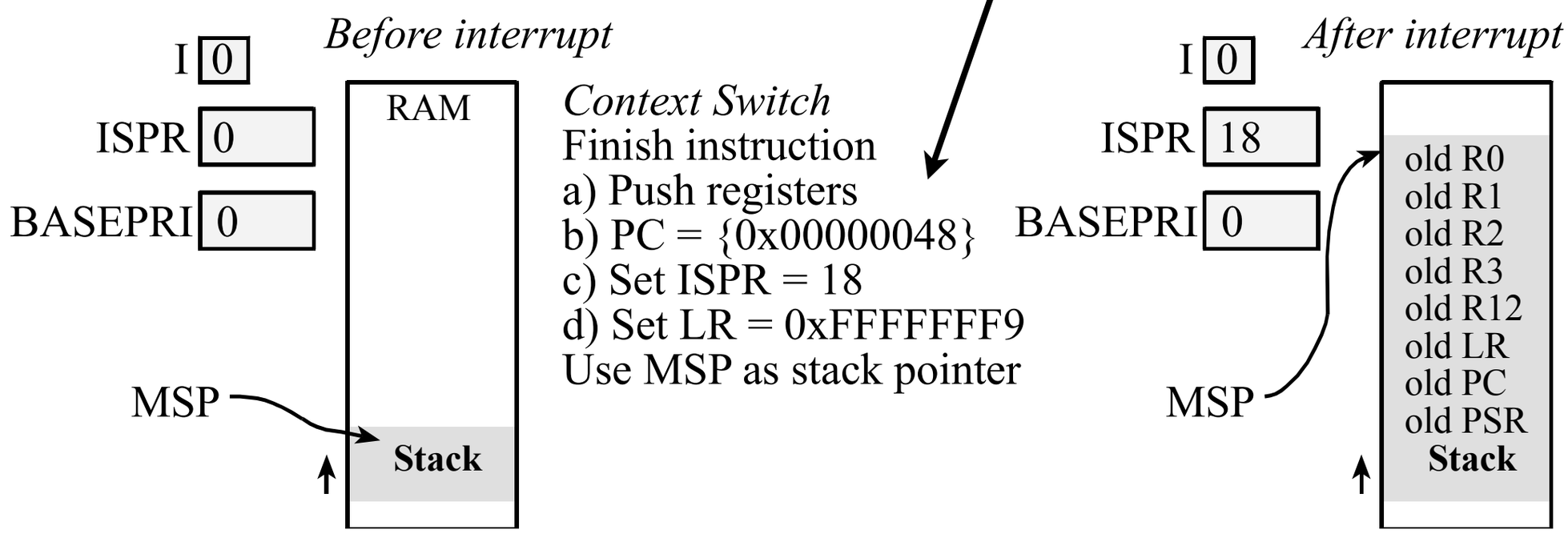
1--4 happens automatically

(less work for us)

context (state) switch

for interrupt by GPIOC:

GPIOC ISR addr located here



end of interrupt service routine (ISR): `bx lr`

set to 0xF...F9 (by uC)

causes registers to be popped off

err..location of interrupts?

Vector address	Number	IRQ	ISR name in **Startup.s**
0x00000038	14	-2	**`PendSV_Handler`**
0x0000003C	15	-1	**`SysTick_Handler`**
0x00000040	16	0	**`GPIOPortA_Handler`**
0x00000044	17	1	**`GPIOPortB_Handler`**
0x00000048	18	2	**`GPIOPortC_Handler`**
0x0000004C	19	3	**`GPIOPortD_Handler`**
0x00000050	20	4	**`GPIOPortE_Handler`**

these are hard coded

(i.e. memory(0x3C) should always have addr of SysTick ISR)

location of interrupts

first few words of Cortex M3 program:

top of stack

Address	Exception Number	Value (Word Size)
0x00000000	–	MSP initial value
0x00000004	1	Reset vector (program counter initial value)
0x00000008	2	NMI handler starting address
0x0000000C	3	Hard fault handler starting address
…	…	Other handler starting address

where your program starts

sample program:

```
0x000 20000408 ;top of stack
0x004 00000101
...
0x100 F000B823  B.W      Start (0x0000014A)
...
0x0000014A <CODE HERE>
```

location of interrupts

Startup.s:

```
        AREA    RESET, CODE, READONLY
        THUMB

        EXPORT  __Vectors
__Vectors
        DCD     StackMem + Stack        ; Top of Stack
        DCD     Reset_Handler           ; Reset Handler
        DCD     NMI_Handler             ; NMI Handler
        DCD     HardFault_Handler       ; Hard Fault Handler
        DCD     MemManage_Handler       ; MPU Fault Handler
        DCD     BusFault_Handler        ; Bus Fault Handler
        DCD     UsageFault_Handler      ; Usage Fault Handler
        DCD     0                       ; Reserved
        DCD     0                       ; Reserved
        DCD     0                       ; Reserved
        DCD     SVC_Handler             ; SVCall Handler
        DCD     DebugMon_Handler        ; Debug Monitor Handler
        DCD     0                       ; Reserved
        DCD     PendSV_Handler          ; PendSV Handler
        DCD     SysTick_Handler         ; SysTick Handler
        DCD     GPIOPortA_Handler       ; GPIO Port A
        DCD     GPIOPortB_Handler       ; GPIO Port B
        DCD     GPIOPortC_Handler       ; GPIO Port C
        DCD     GPIOPortD_Handler       ; GPIO Port D
        DCD     GPIOPortE_Handler       ; GPIO Port E
```

reserves one word for each

0x00

0x50

create function with name and assembler/compiler puts its addr here

what can trigger interrupt?

we'll consider:

1. timer expiration (SysTick &GPTM)
2. peripherals (UART)
3. external events (GPIO)

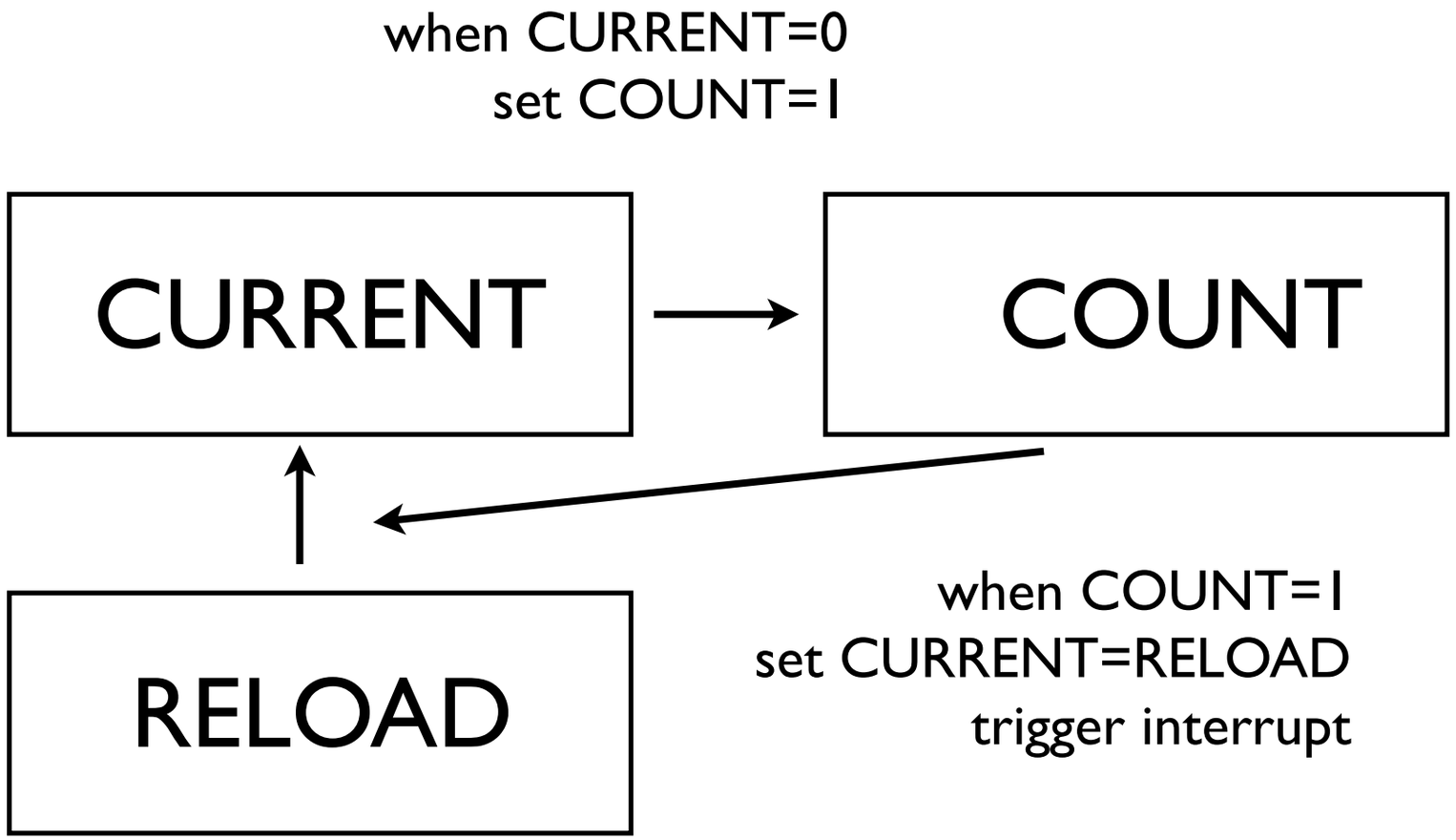
to enable interrupt for event, set appropriate bit to 1

(good lucking finding bit)

1. timer interrupts (SysTick)

SysTick:

1. start at RELOAD
(CURRENT=RELOAD)
2. count down to zero
(CURRENT--)
3. set COUNT=1
4. trigger interrupt
(15 put into IPSR)
5. goto '1.'

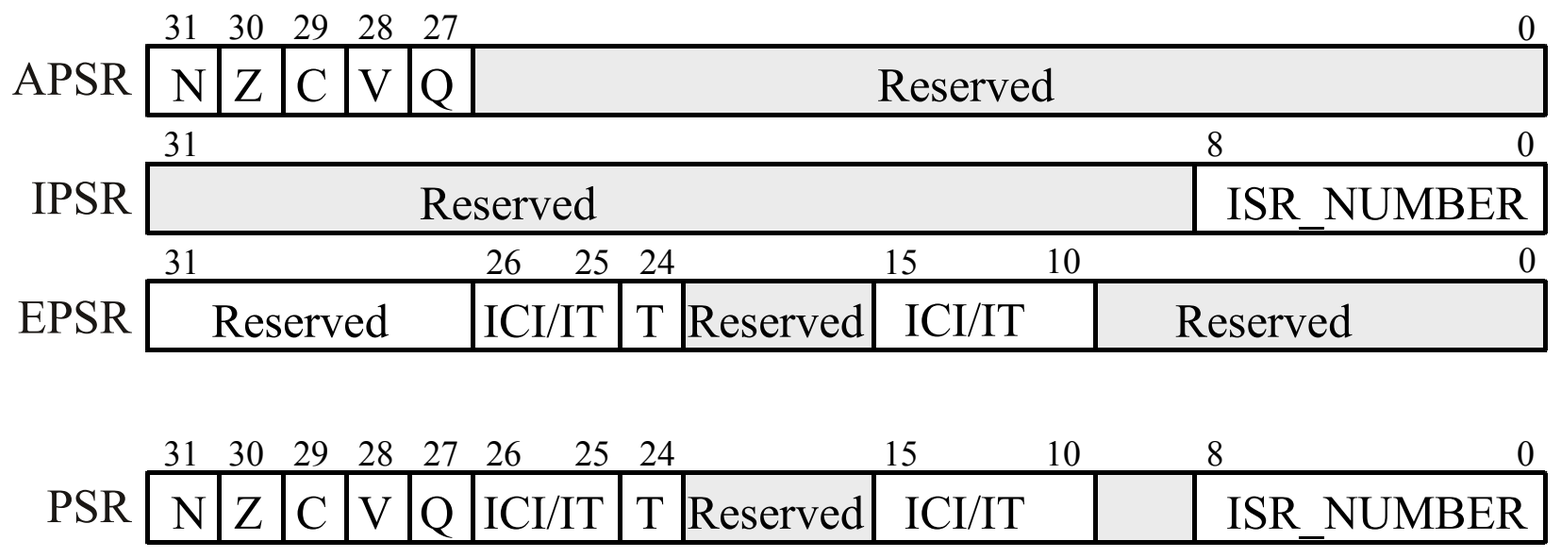


SysTick is special

note: COUNT needn't be reset:
1. timer keeps counting
2. further interrupts

1. timer interrupts (SysTick)

xPSR:



interrupt number being serviced

15 for SysTick

to use SysTick interrupt

=1

Address	31-24	23-17	16	15-3	2	1	0	Name
$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

ex: 50% duty cycle

(`SysTick` as src, `PD.0` as output)

```
;cortex m3 core peripherals (systick and nvic, e.g.)
M3CP EQU 0xE000E000
;bit band addr for port d, etc
PD0_DATA_B EQU 0x420E7F80
PD0_DIR_B EQU 0x420E8000
PD0_EN_B EQU 0x420EA380
CLK_PD_B EQU 0x43FC210C
```

word in R/W memory

```
  AREA  DATA, ALIGN=2
CNT SPACE   4  ; records number of SysTick interrupts
  ALIGN        ; make sure the end of this section is
aligned


  AREA    |.text|, CODE, READONLY, ALIGN=2
  THUMB
  EXPORT  SysTick_Handler
  EXPORT  Start
```

need to explicitly define ISR for SysTick interrupt

ex: 50% duty cycle

(SysTick as src, PD.0 as output)

```
Start
  ;port d init
  ...

  ;SysTick Init
  ldr R1,=M3CP
  ;1. stop timer
  mov R0,#0
  str R0,[R1,#0x10]
  ;2. set init value (RELOAD)
  ldr R0,=0xFF
  str R0,[R1,#0x14]
  ;3. clear current timer value (any write clears CURRENT)
  mov R0,#0
  str R0,[R1,#0x18]
  ;4. set options and start counting
  ; clk_src=system clock
  ; inten=1 (trigger interrupt at timer expiration)
  ; enable=1 (start counting)
  mov R0,#0x7 ;0x7 = 0b111 (CLK_SRC,INTEN,ENABLE)
  str R0,[R1,#0x10]

loop
   b loop ;wait for interrupt
```

expiration causes interrupt

ex: 50% duty cycle

(`SysTick` as src, `PD.0` as output)

```
SysTick_Handler
  ; toggle pin
  ldr R1, =PD0_DATA_B
  ldr R0,[R1]
  mvn R0,R0
  str R0,[R1]
  ; increment counter
  ldr R1,=CNT
  ldr R0,[R1]
  add R0,#1
  str R0,[R1]
  bx LR
```

must be same as Startup.s

note:
1. location of ISR
2. stack
3. xPSR

your response?

interrupts are nearly as exciting

ex: 50% duty cycle in C

(`SysTick` as src, `PD.0` as output)

global vars:

```
// m3 core peripherals base
unsigned char *M3CP = (unsigned char *) 0xE000E000;
// PD0 setup
// (chars at bb addr: take advantage of the fact
//  that write to bb addr only writes lsb)
unsigned char PD0_DATA_B __attribute__((at(0x420E7F80)));
unsigned char PD0_DIR_B __attribute__((at(0x420E8000)));
unsigned char PD0_EN_B __attribute__((at(0x420EA380)));
unsigned char CLK_PD_B __attribute__((at(0x43FC210C)));
// count the number of times systick isr called
unsigned int CNT = 0;
```

bit banding in C

(it's not bool, so be careful)

ex: 50% duty cycle in C

(`SysTick` as src, `PD.0` as output)

```
int main(void)
{
   SysTickInit();

   PD0Init();

  //could do other things, PD.0 now has 50% DC wave
  while(1);
}
```

again, as per Startup.s

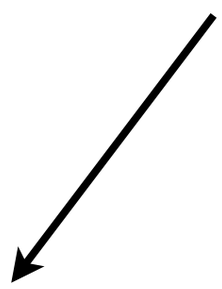
```
void SysTick_Handler(void)
{
   PD0_DATA_B = ~PD0_DATA_B; //bitwise not
  CNT++;
}
```

you think C is easier...ha!

gew...

so many things can go wrong...

1. if ISR changes function variables, must be volatile
2. interrupt in middle of instruction execution
   (say doing floating point ops)
3. stack overflow
4. non-reentrant functions
   (functions that can't be interrupted: data i/o e.g.)



have to think about what can happen when interrupt occurs (disable, if necessary)

tips for ISR in C

The following rules should be followed when using interrupts.

- Every global variable that can be written to inside an ISR and that is accessed outside the ISR must be declared as **volatile**. This will ensure that the optimizer does not remove instructions relating to this variable.

- Disable interrupts whenever using data in a non-atomic way (i.e., accessing 64-bit/128-bit variables). Access to a variable is atomic when the processor cannot interrupt (with an ISR) storing and loading data to and from memory.

- Avoid calling functions from within an ISR. If you must do this, declare the function as **reentrant** which allocates all local variables in the function on the stack instead of in RAM.

http://www.maxim-ic.com/app-notes/index.mvp/id/3477

using interrupts:

1. set priorities
2. enable interrupts for peripheral
3. write ISR
4. wait for IRQ

pending interrupts and priorities

Q: what if one interrupt is triggered while another is being serviced?

same or different source

pending interrupts and priorities

Q: what if one interrupt is triggered while another is being serviced?

same or different source

A: two things to think about

a. same interrupt

b. different interrupts

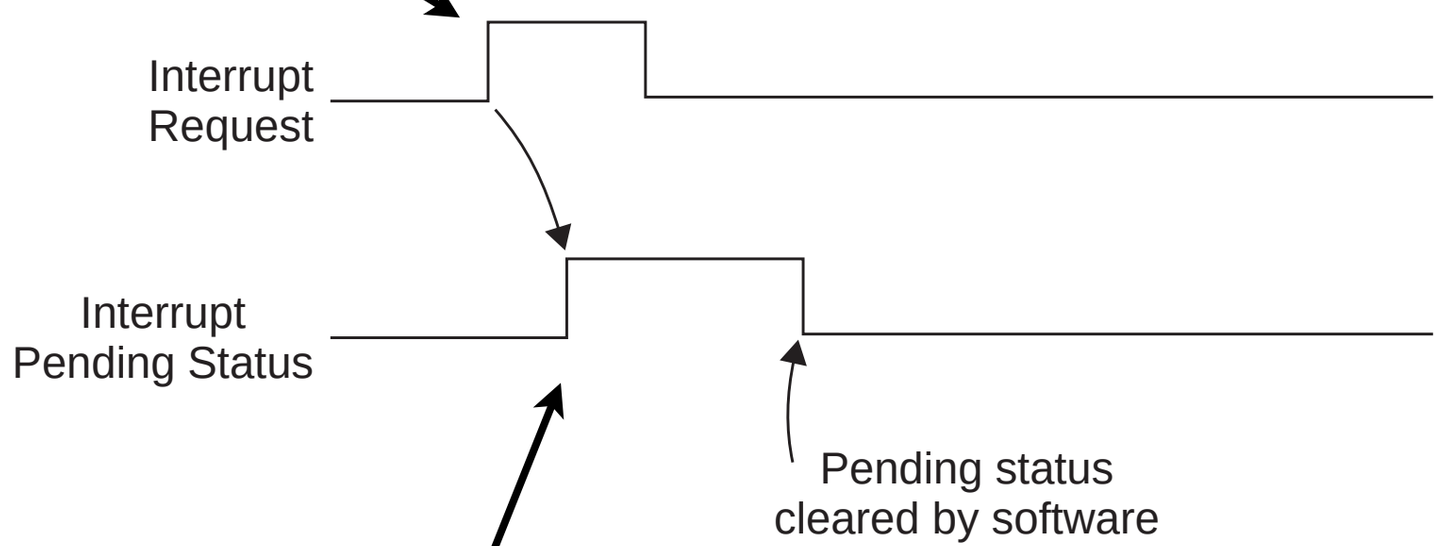
(w/ or w/o different priorities)

pending interrupts

(how uC manages interrupts)

peripheral IRQ

(asking to be serviced)



uC alerted that interrupt needs

(something needs to be done)

how does this happen?

(how does uC know that IRQ no longer pending?)

pending interrupts

(how uC manages interrupts)

clearing pending status:

1. write to NVIC

interrupt number

(see table 2-9, p81)

Interrupt 0-31 Clear Pending (UNPEND0)

Base 0xE000.E000
Offset 0x280
Type R/W, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	INT															
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	INT															
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

writing 1 to a bit clears pending flag for that interrupt

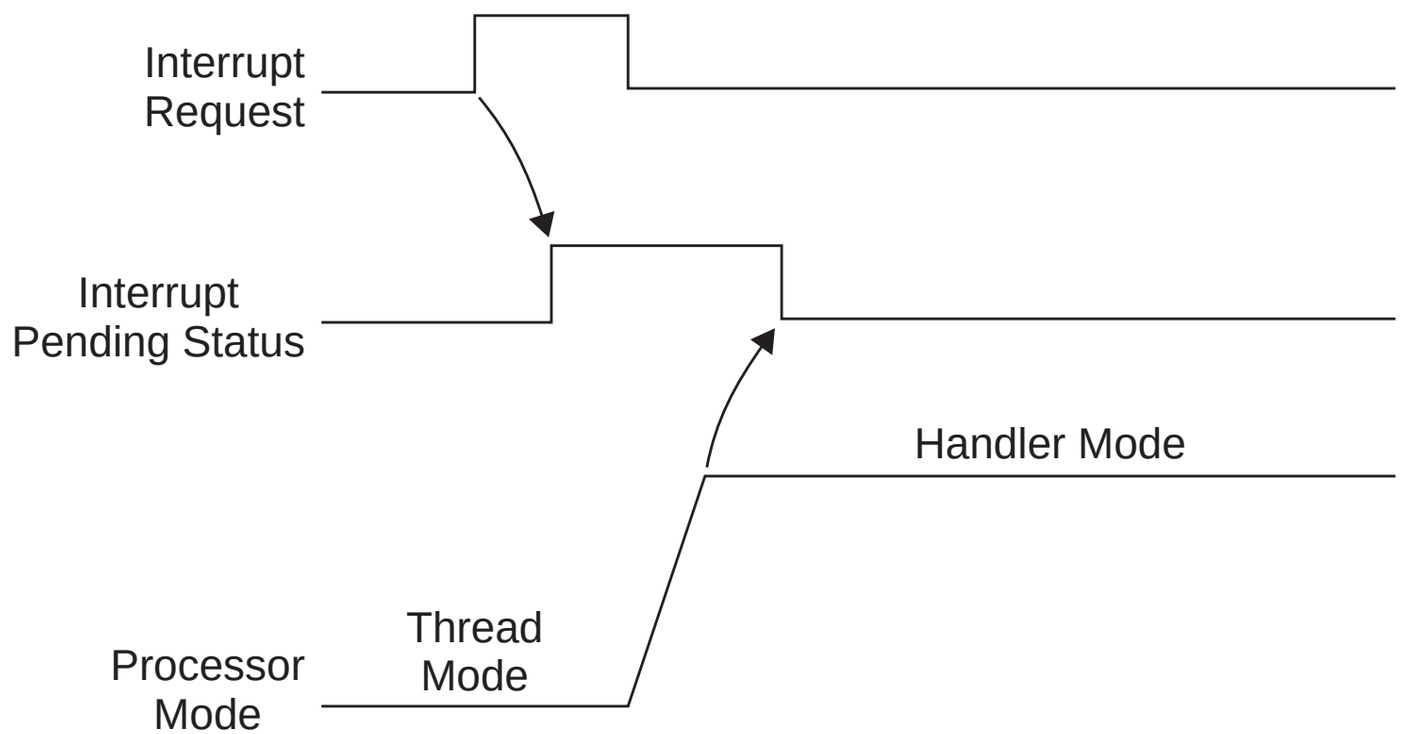
note: set pending will cause ISR to be executed

pending interrupts

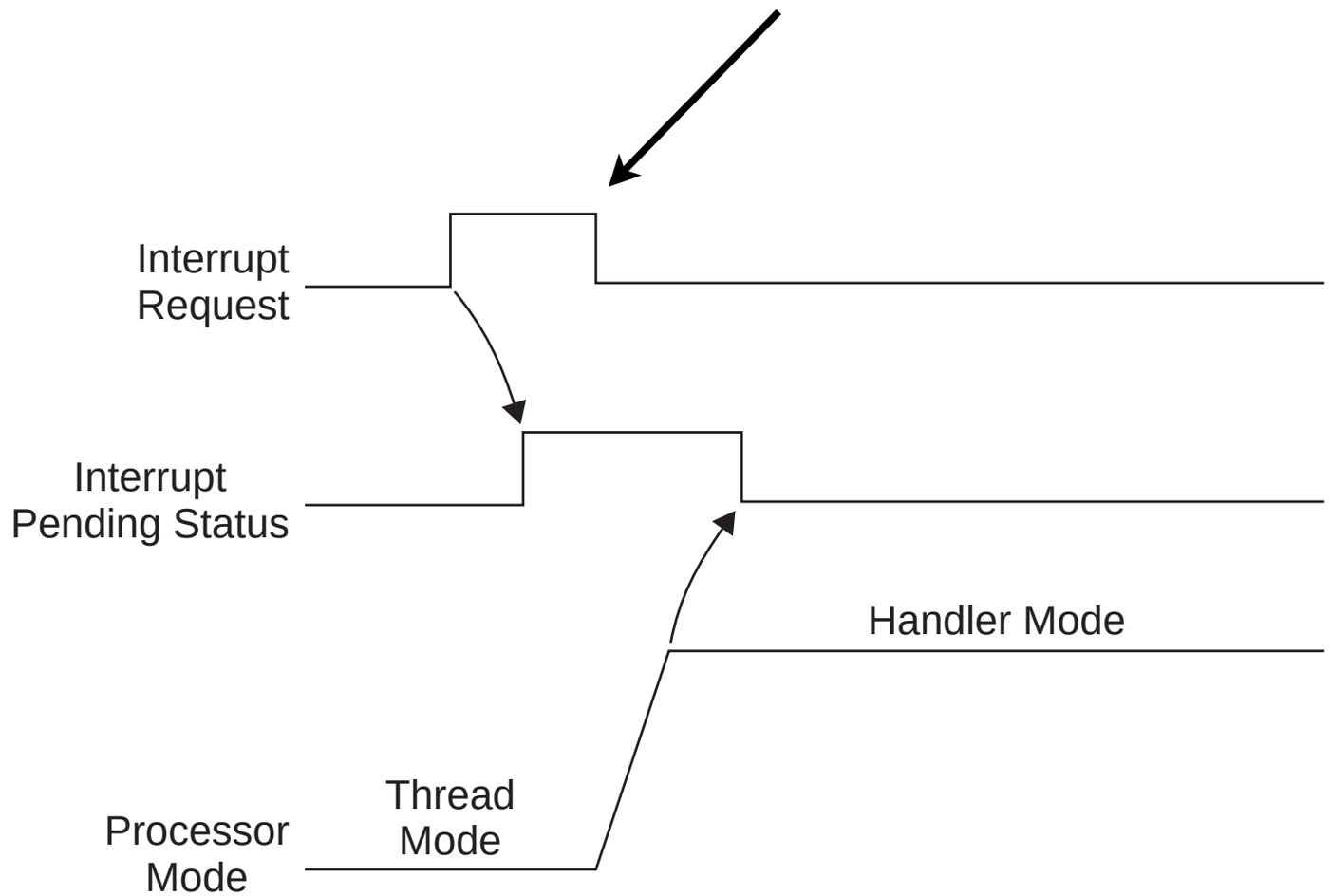
(how uC manages interrupts)

clearing pending status:

2. execution of ISR automatically clears



Q: what causes IRQ to go low?



A: the peripheral that caused it to go high

implies: aside from SysTick (because it's an internal peripheral), ISR must tell peripheral it has serviced interrupt

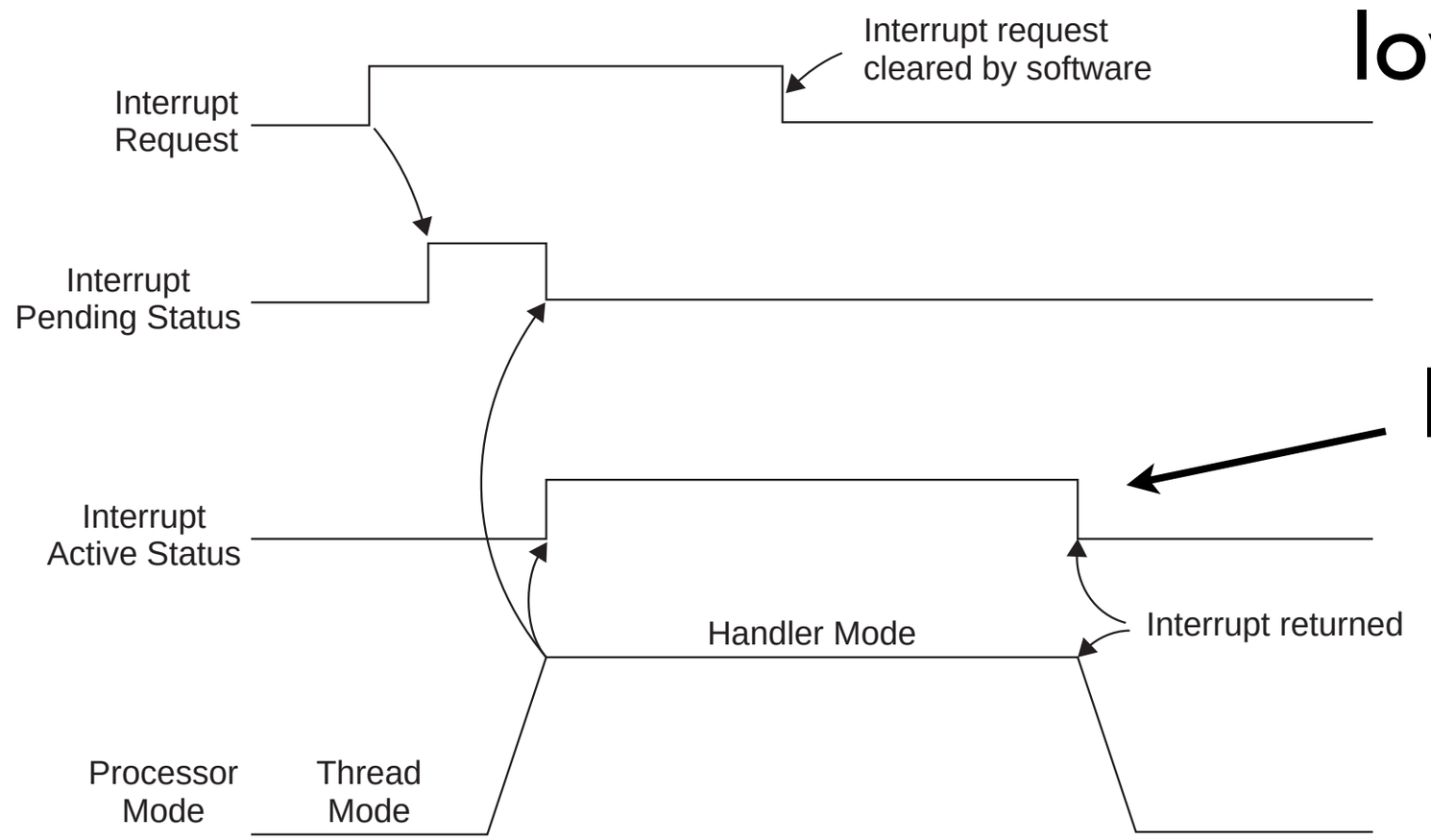
pending interrupts

(how uC manages interrupts)

note IRQ can stay high after pending is low:

so long as IRQ low before

ISR ends

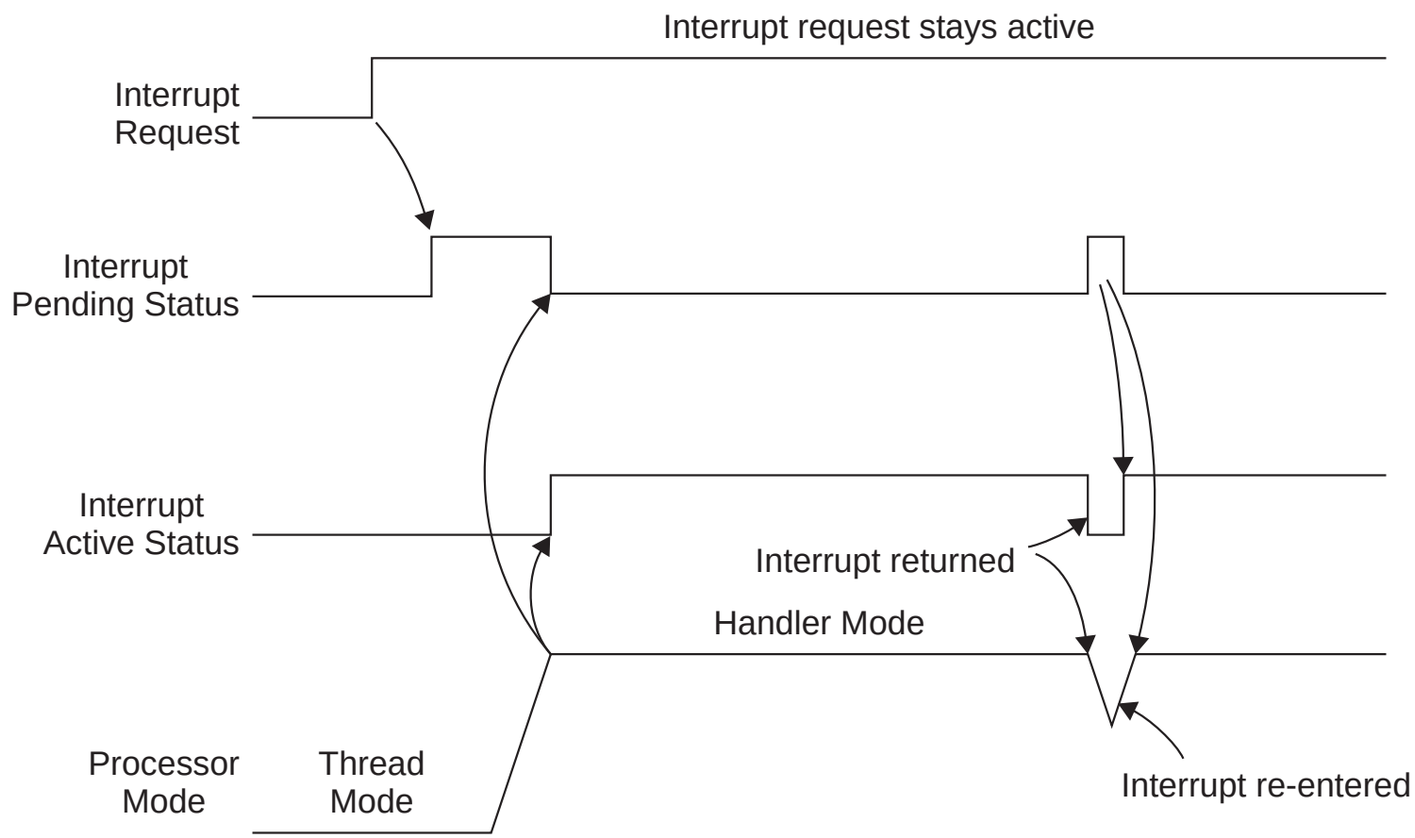


Q: what if IRQ still asserted after ISR exits?

pending interrupts

(how uC manages interrupts)

Q: what if IRQ still asserted after ISR exits?



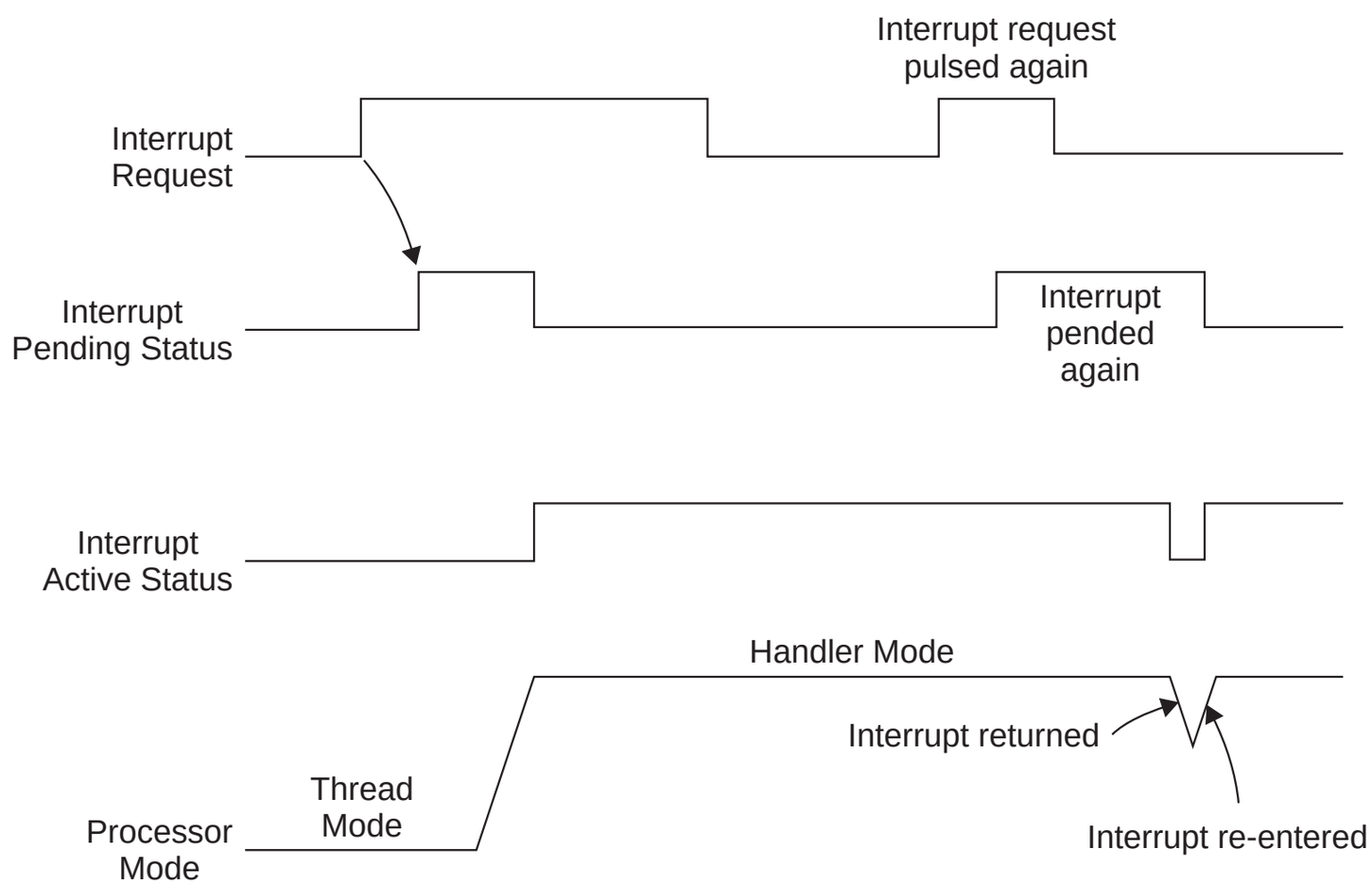
A: ISR is re-entered after it exits

Q: what if IRQ reasserted during ISR?

pending interrupts

(how uC manages interrupts)

Q: what if IRQ reasserted during ISR?



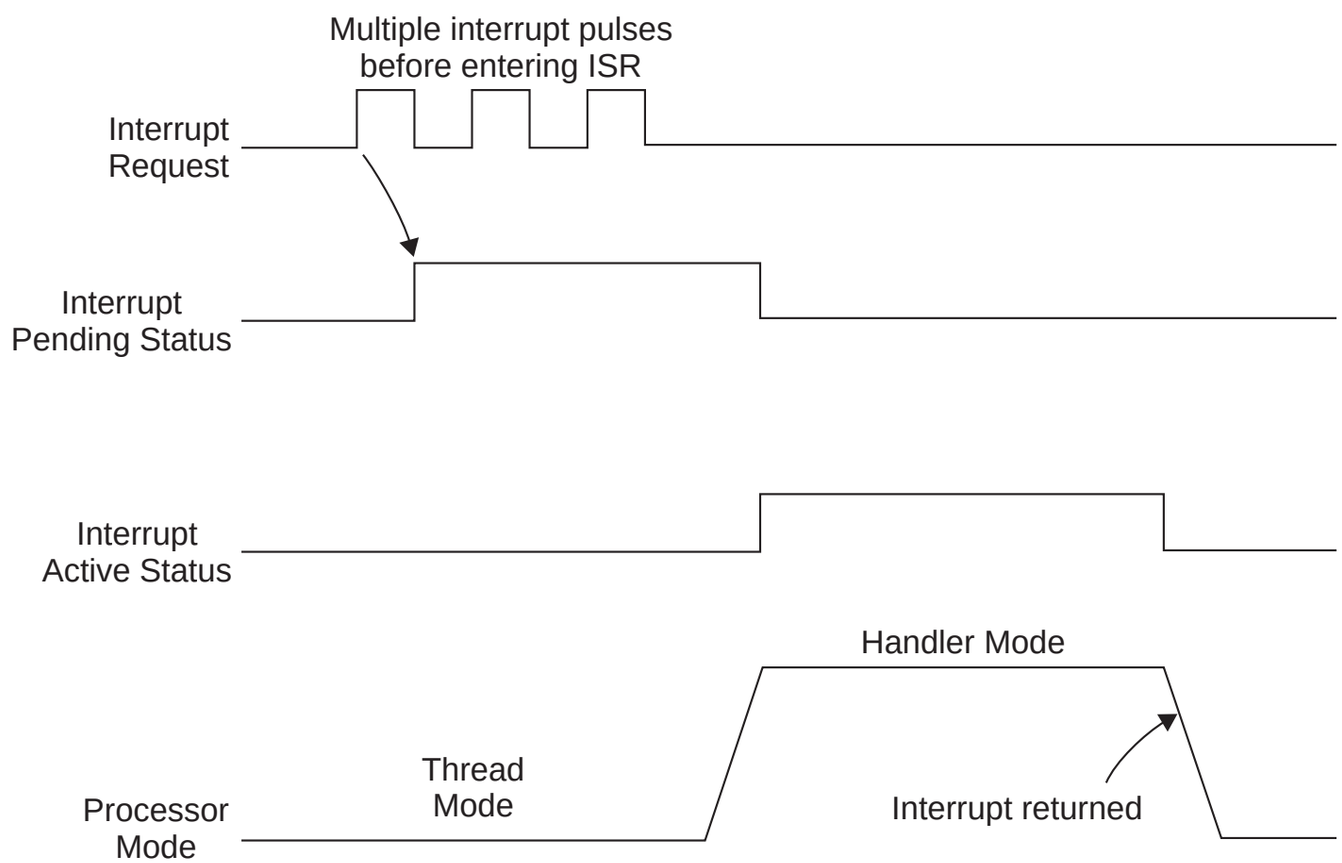
A: ISR is re-entered after it exits

Q: what if IRQ asserted multiple times before ISR entered?

pending interrupts

(how uC manages interrupts)

Q: what if IRQ asserted multiple times before ISR entered?



A: ISR entered only once