Presentation

of

Microprocessors and Microcontrollers

Unit-3: MSP430 Microcontroller Peripherals

By

K Venkata Siva Reddy

Assistant Professor

Department of ECE

Ravindra College of Engineering for Women

Kurnool – 518452, Andhra Pradesh, India

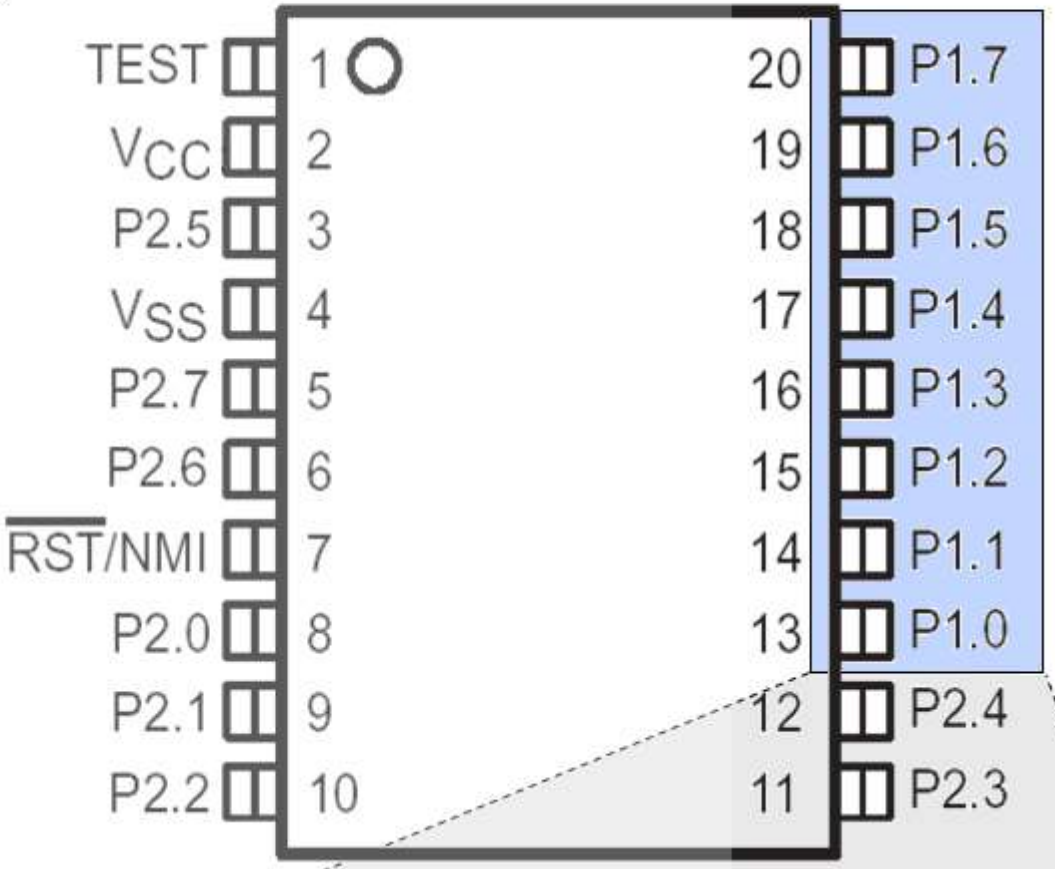
RCEW, Pasupula (V), Nandikotkur Road,
Near Venkayapalli, KURNOOL

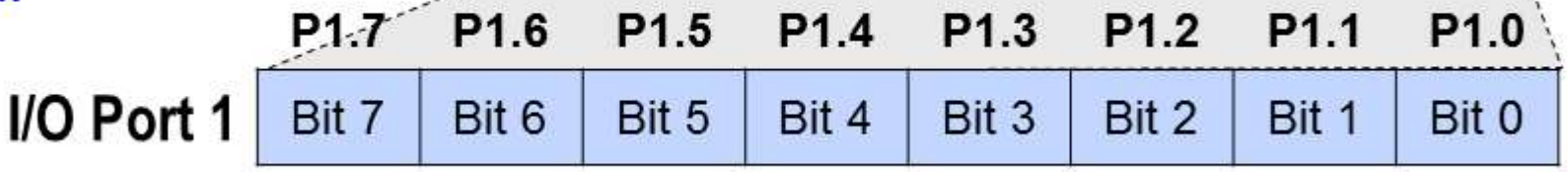
Syllabus

- I/O Ports & Pull up/down Resistors Concepts
- Interrupts and Interrupt Programming
- Watchdog Timer
- System Clocks
- Low Power aspects of MSP430: ***Low Power Modes***
- Active vs Standby Current Consumption
- FRAM ***vs*** Flash for Low Power & Reliability
- Timer & Real Time Clock (RTC)
- PWM Control
- ADC
- Comparator
- Data Transfer using DMA

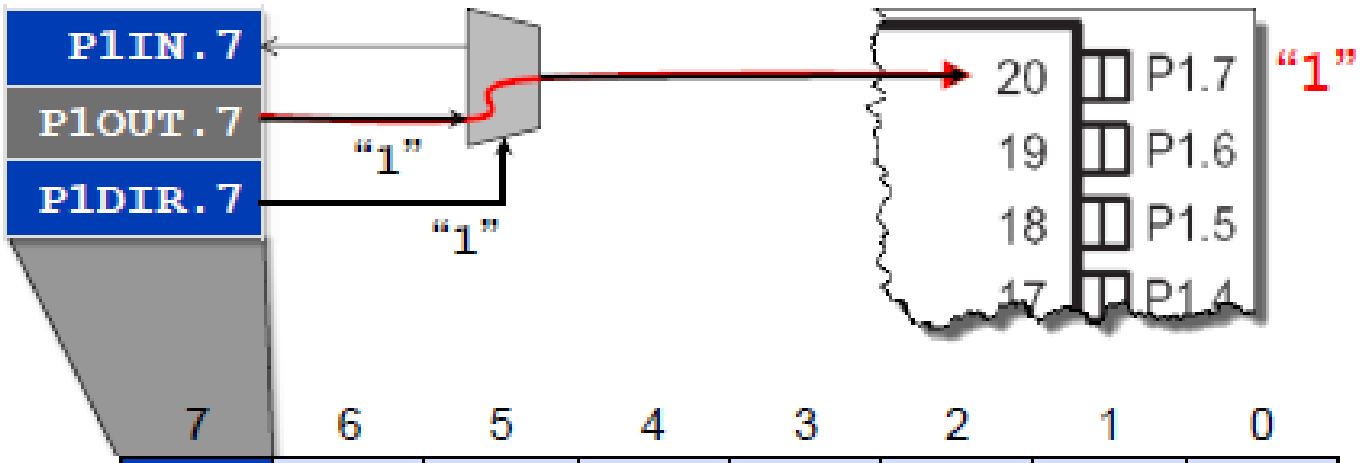
MSP430 GPIO Ports

- GPIO = General Purpose Bit Input/Output
- 8-bit I/O ports
- 1 to 12 ports, depending on family and pin-count
- Each pin is individually controllable
- Input pins can generate interrupts (Chapter 5)
- Controlled by memory-mapped registers:
  - IN
  - OUT
  - DIR
  - REN
  - SEL
  - …





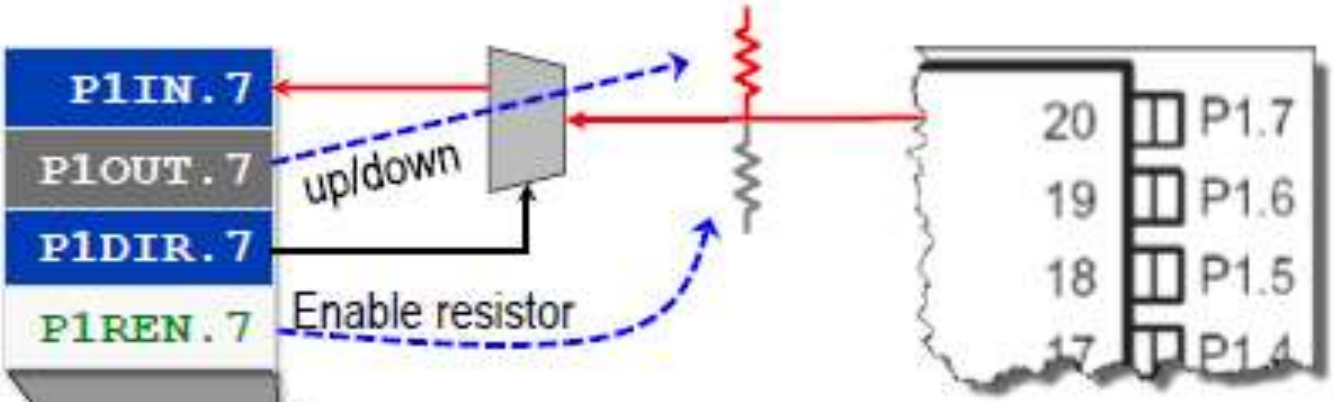
GPIO Output



	7	6	5	4	3	2	1	0
P1IN	x							
P1OUT	1							
P1DIR	1							

- PxOUT.y: 0 = low
  1 = high
- Register example:
  `P1OUT |= 0x80;`

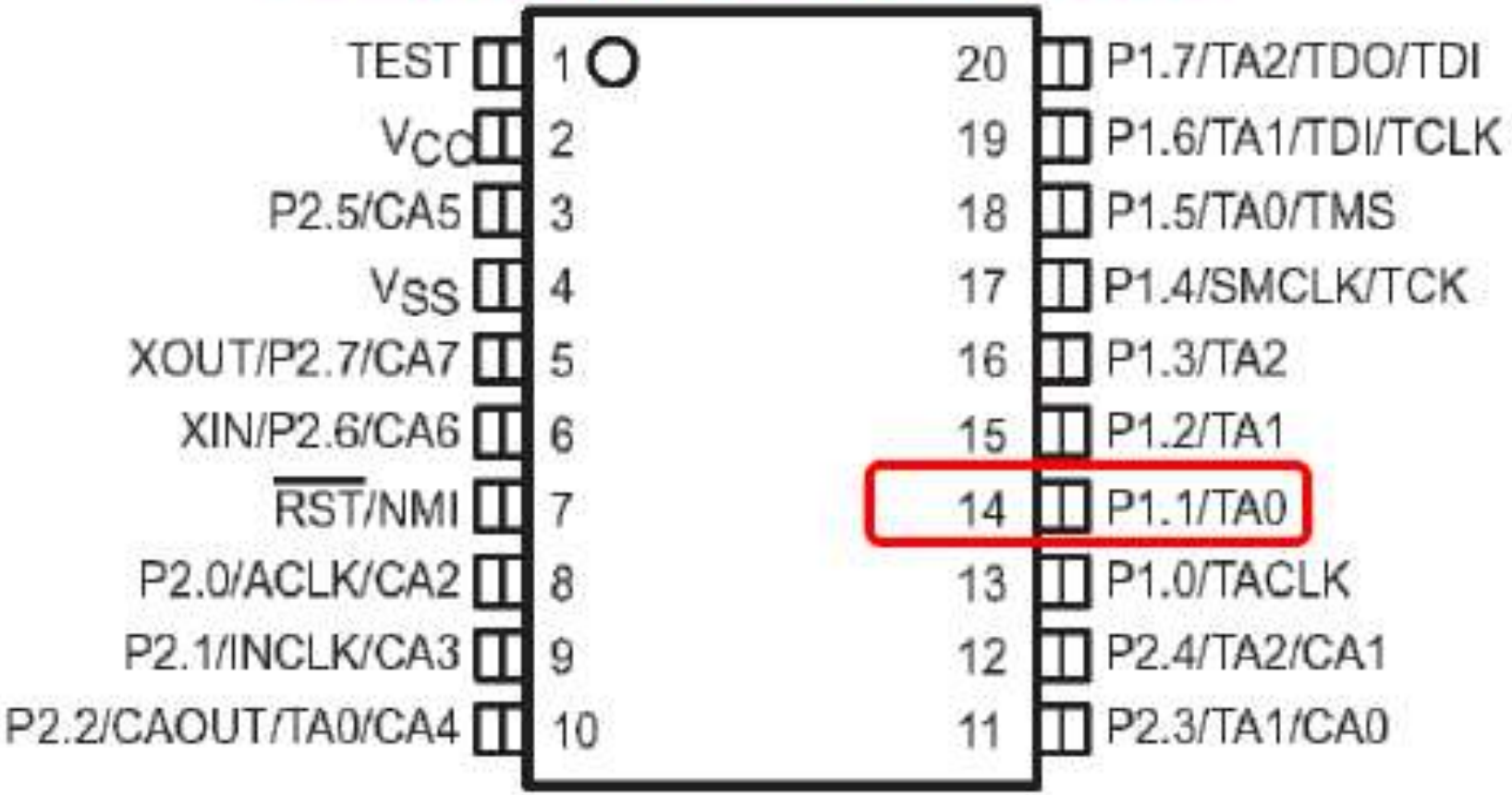
GPIO Input (Resistors)



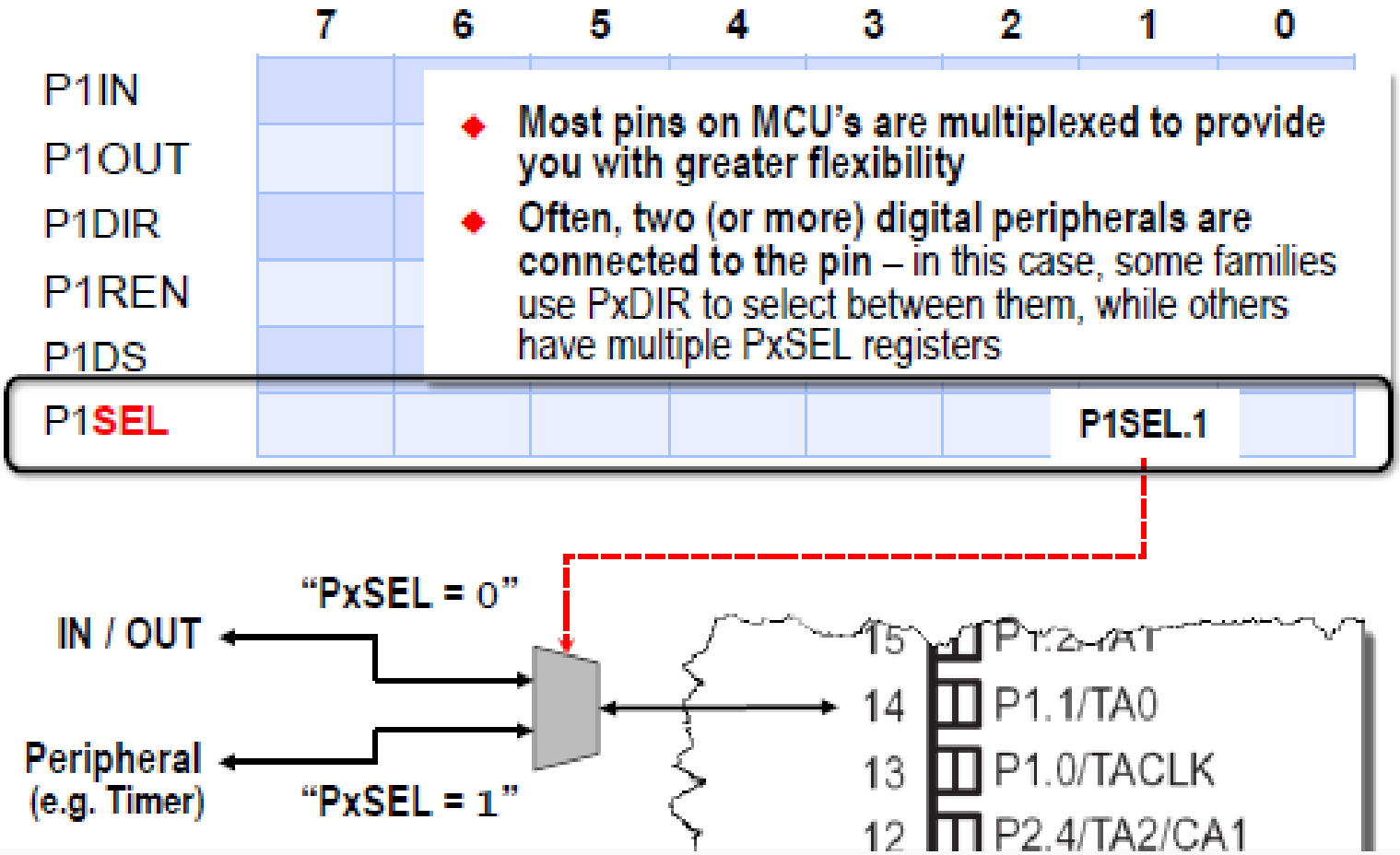
	7
P1IN	x
P1OUT	1
P1DIR	0
P1REN	1

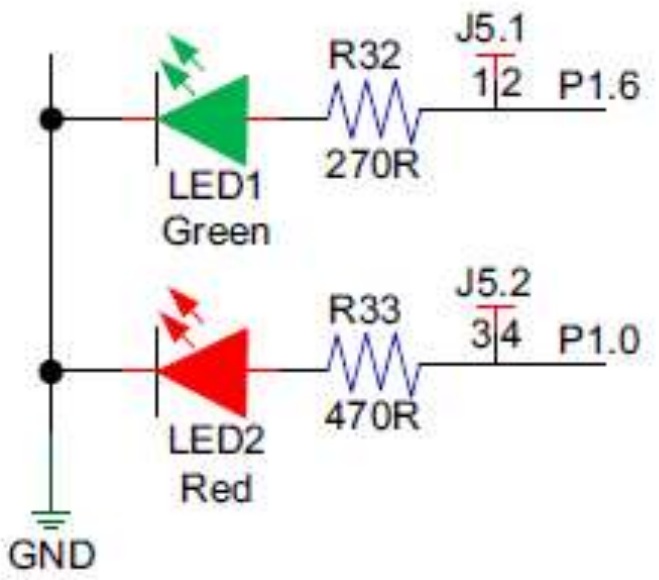
- Input pins are held in high-impedance (Hi-Z) state, so they can react to 0 or 1
- When not driven, Hi-Z inputs may float up/down ... prevent this with pullup/pulldown resistors
- PxREN enables resistors
  PxOUT selects pull-up (1) or -down (0)
- Lower cost devices may not provide up/down resistors for all ports

Controlling GPIO Ports



Pin Flexibility





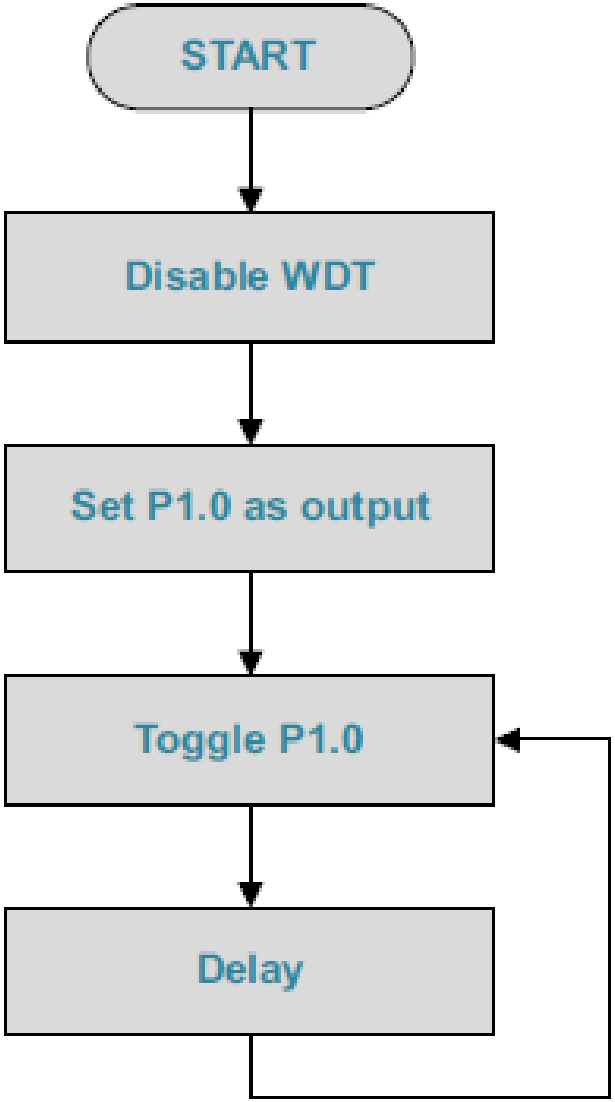


Figure 1-3 Flowchart to Blink the LED Using GPIO


```
#include<msp430.h>

int main(void) {
WDTCTL = WDTPW | WDTHOLD;          // Stop watchdog timer
P1DIR |= 0x01;                     // Set P1.0 to output direction

while(1) {
volatile unsigned long i;          // Volatile to prevent
                                   //optimization
P1OUT ^= 0x01;                     // Toggle P1.0 using XOR
i = 50000;                         // SW Delay
do i--;
while(i != 0);
}
}
```

I/O Ports Pull up/down resisters

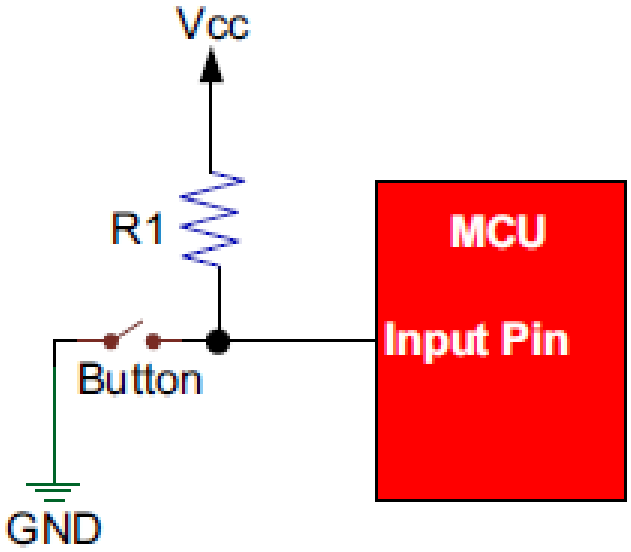


Figure 2-3 Pull-up Resistor for Input Pin

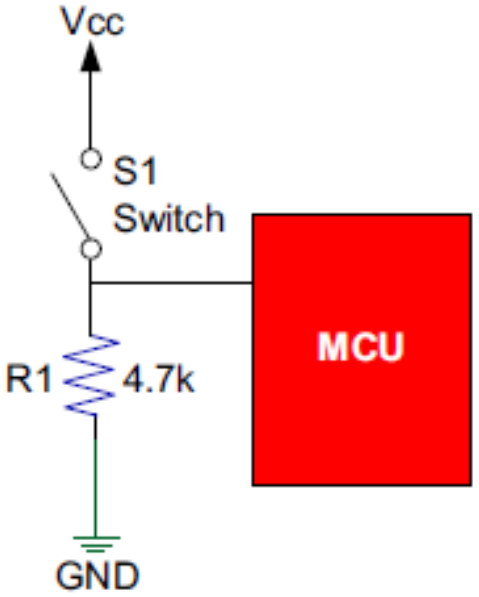


Figure 2-4 Pull-down Resistor for Input Pin

I/O Ports Pull up/down resisters

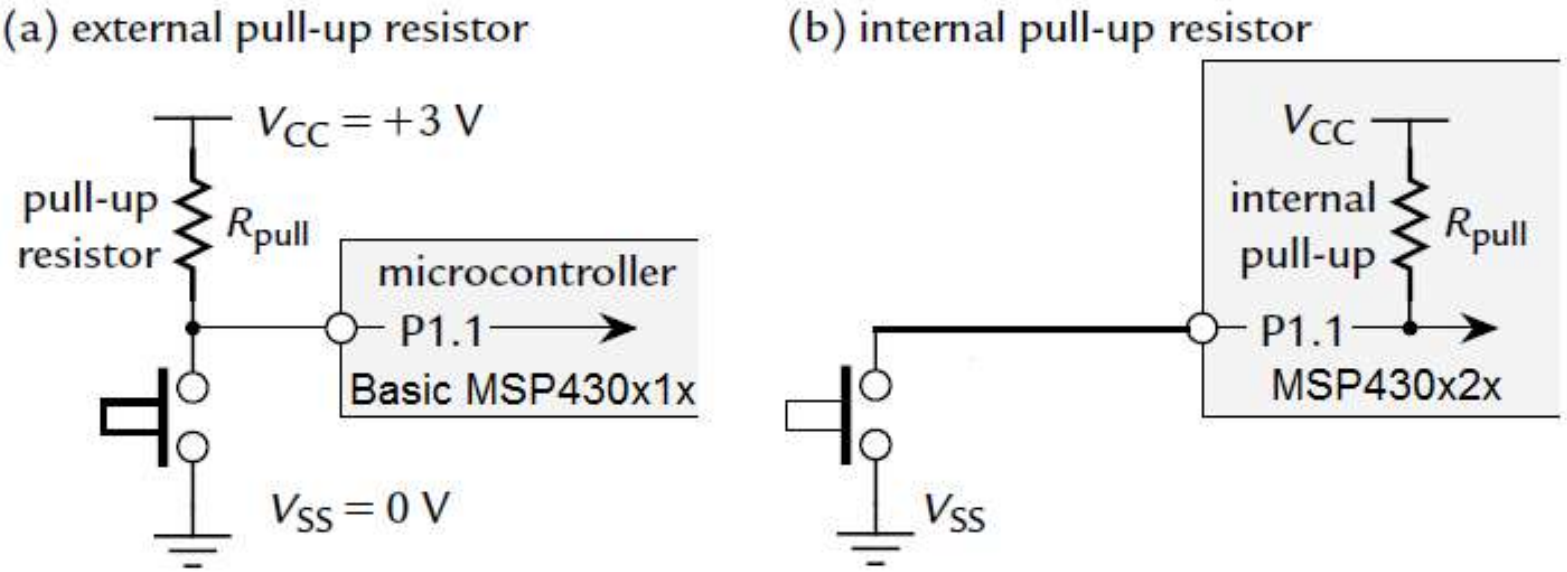


Figure: Standard, active low connection of a push button to an input with an (a) external and (b) internal pull-up resistor R_{pull}.

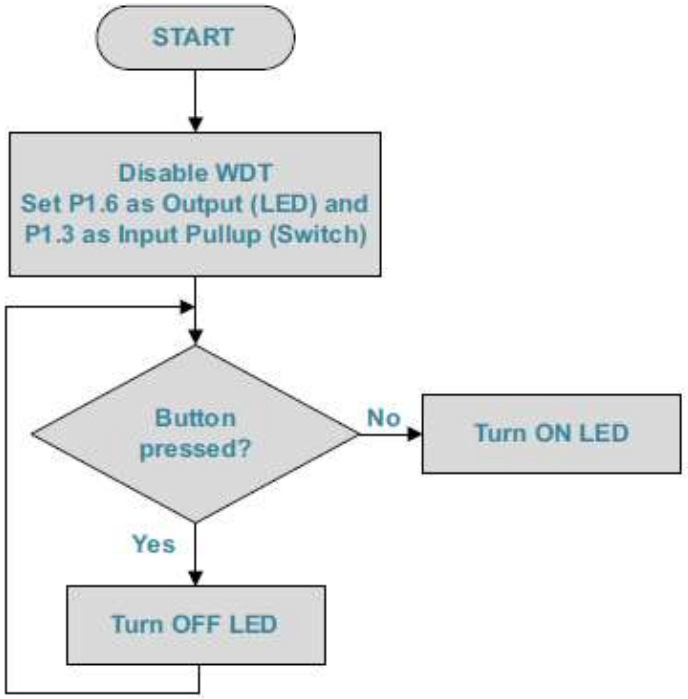


Figure 2-5 Flowchart for Controlling LED with a Switch

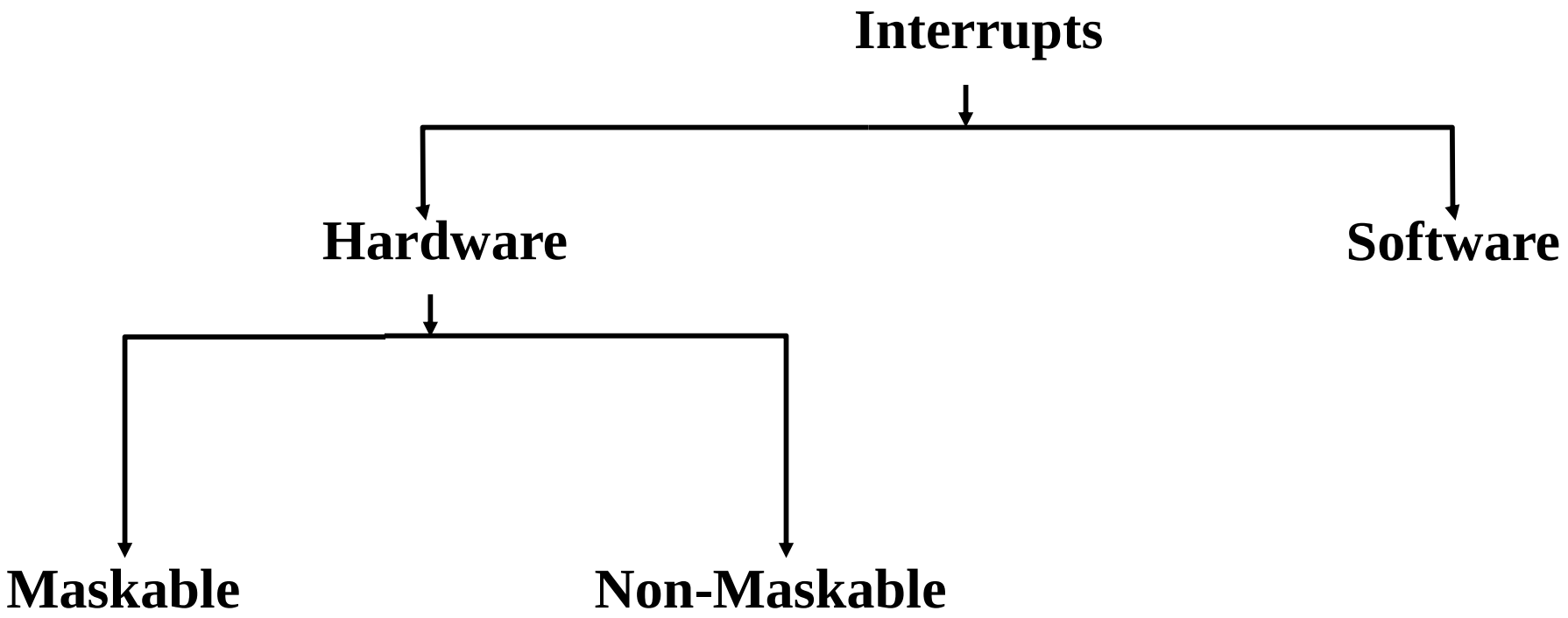
```
#include<msp430.h>


int main(void) {

WDTCTL = WDTPW | WDTHOLD;

P1DIR |= 0x40;

P1REN |= 0x08;

P1OUT |= 0x08;
```

Interrupt & Interrupt Programming

Generally

- An **interrupt request** (usually called an **interrupt**) is generated by an **interrupt source**.
- An **interrupt** need to be served by a special program, the **interrupt service routine (ISR)**.
- An interrupt can take place any time independent of program flow (in difference to subroutine calls).
- Interrupts can be **maskable or non-maskable**.
- Different interrupt sources have different **priority**.
- To react on an interrupt the most microcontroller containing a **vector interrupt system**.

CLASSIFICATION OF INTERRUPTS



Task Handling

Event/Task/Service routine can handle by two methods.

They are: 1) Polling method

2) Interrupt method

1) Polling method is continuous checking of condition or status of device by program or device to see what state they are in. 100% CPU Load

2) In interrupt method continuous checking of condition or status is not required because in this if condition or status met the required condition/state then automatically a signal is send to processor. <0.1% CPU Load

Waiting for an Event: Family Vacation

Polling

Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?

Waiting for an Event: Family Vacation

Polling

Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?

Interrupts

Wake me up when we get there...

ula (V), Nandikotkur Road,
nkayapalli, KURNOOL

An engineering example...

Waiting for an Event: Button Push

Polling

```
while(1)
{
if(P1IFG & BIT2) // P1.2 IFG cleared
{
P1IFG &= ~BIT2; // P1.2 IFG cleared
P1OUT ^= BIT0; // Toggle LED at P1.0
}
```

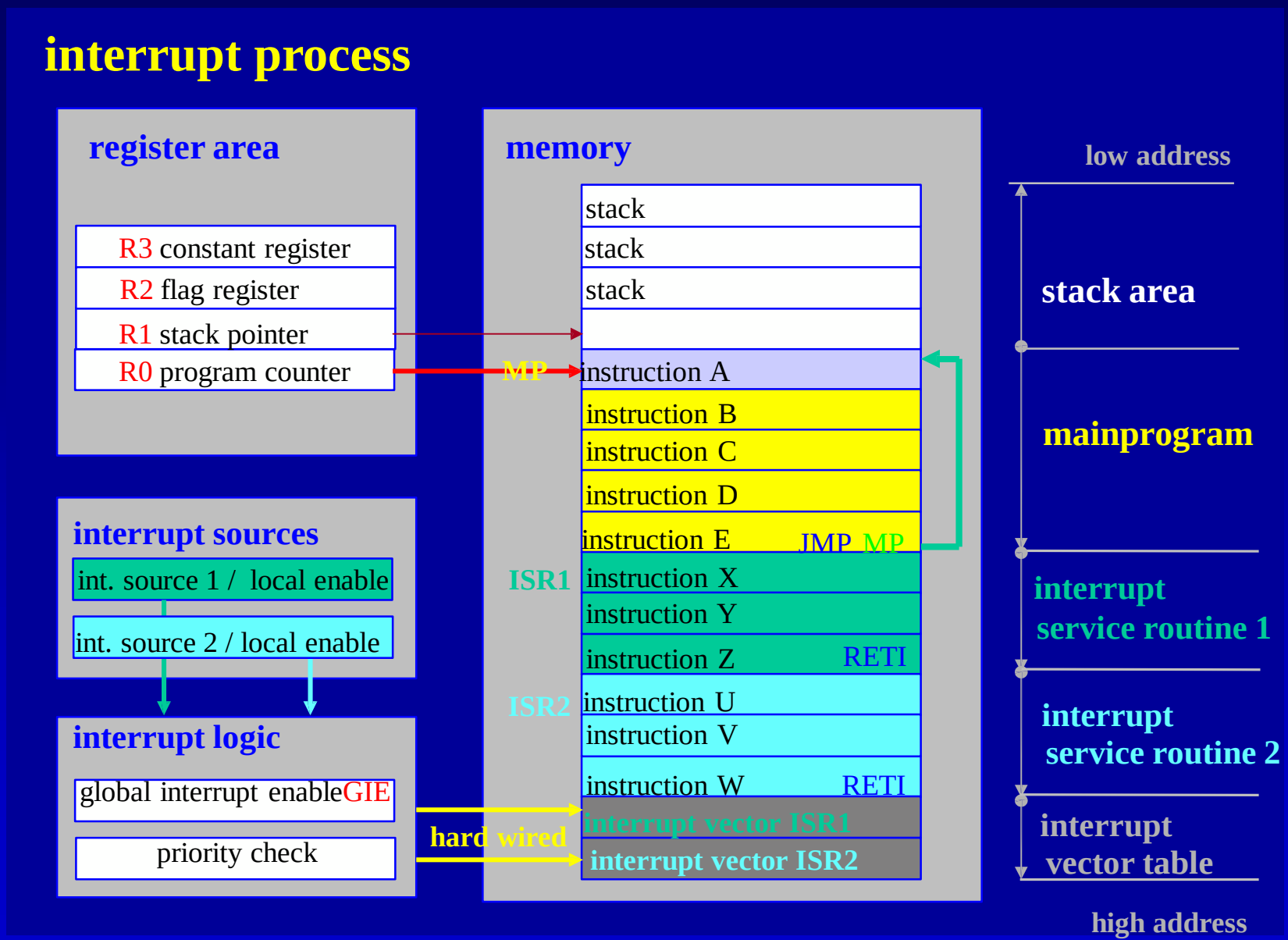
100% CPU Load

Interrupts

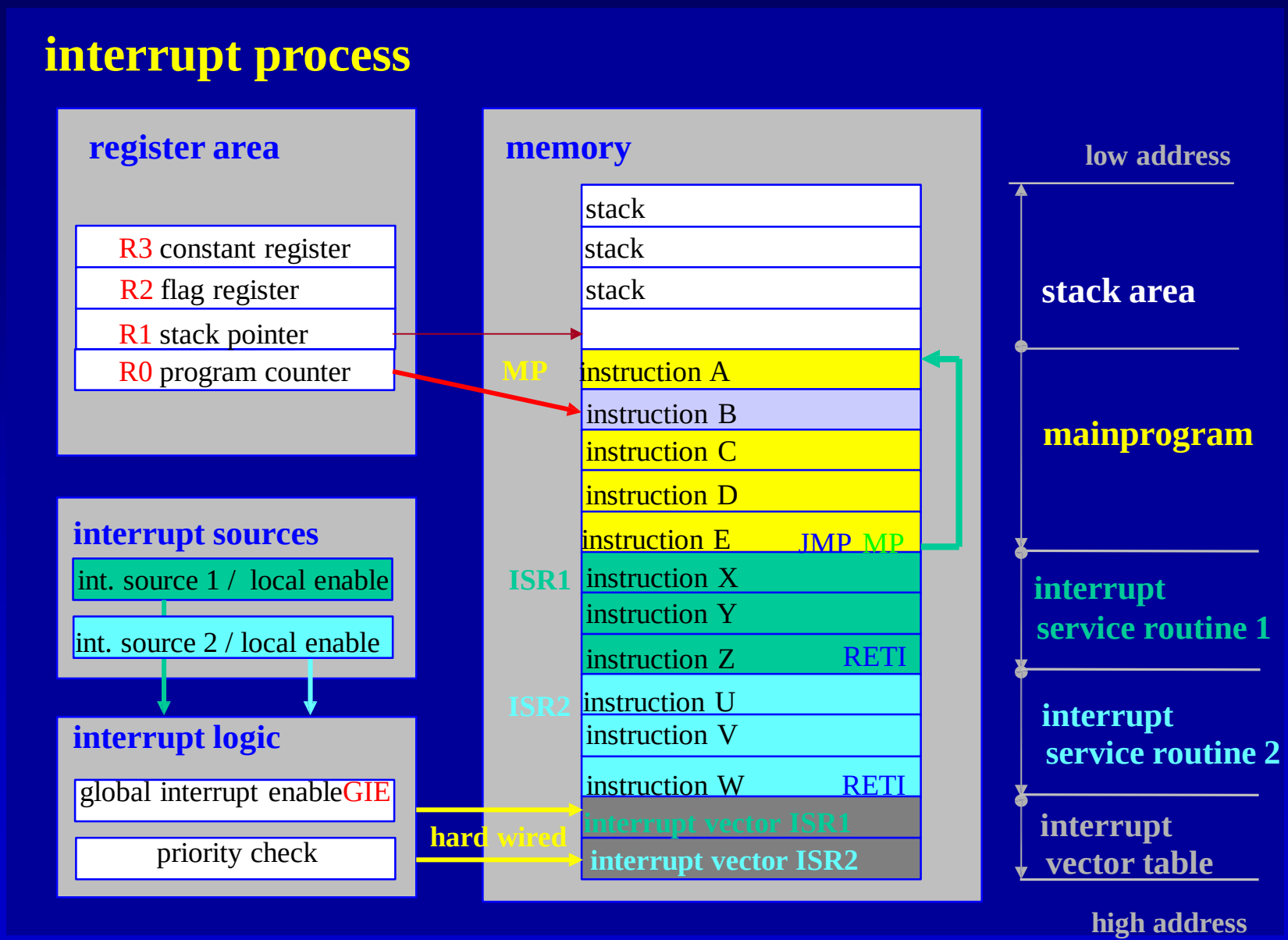
```
// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
P1OUT ^= BIT0; // Toggle LED at P1.0
P1IFG &= ~BIT2; // P1.2 IFG cleared
}
```

> 0.1% CPU Load

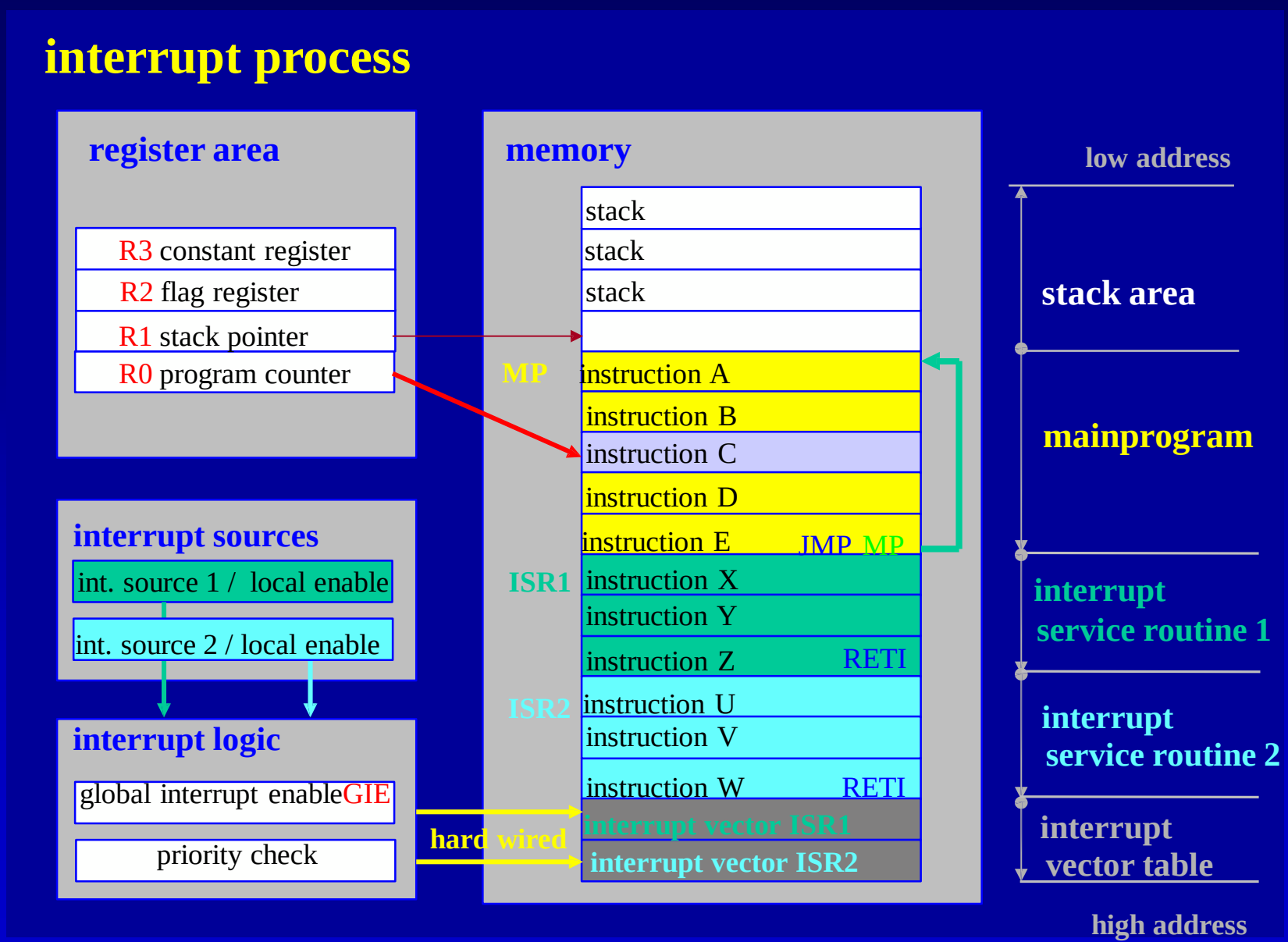
What Happens when an Interrupt Is Requested?



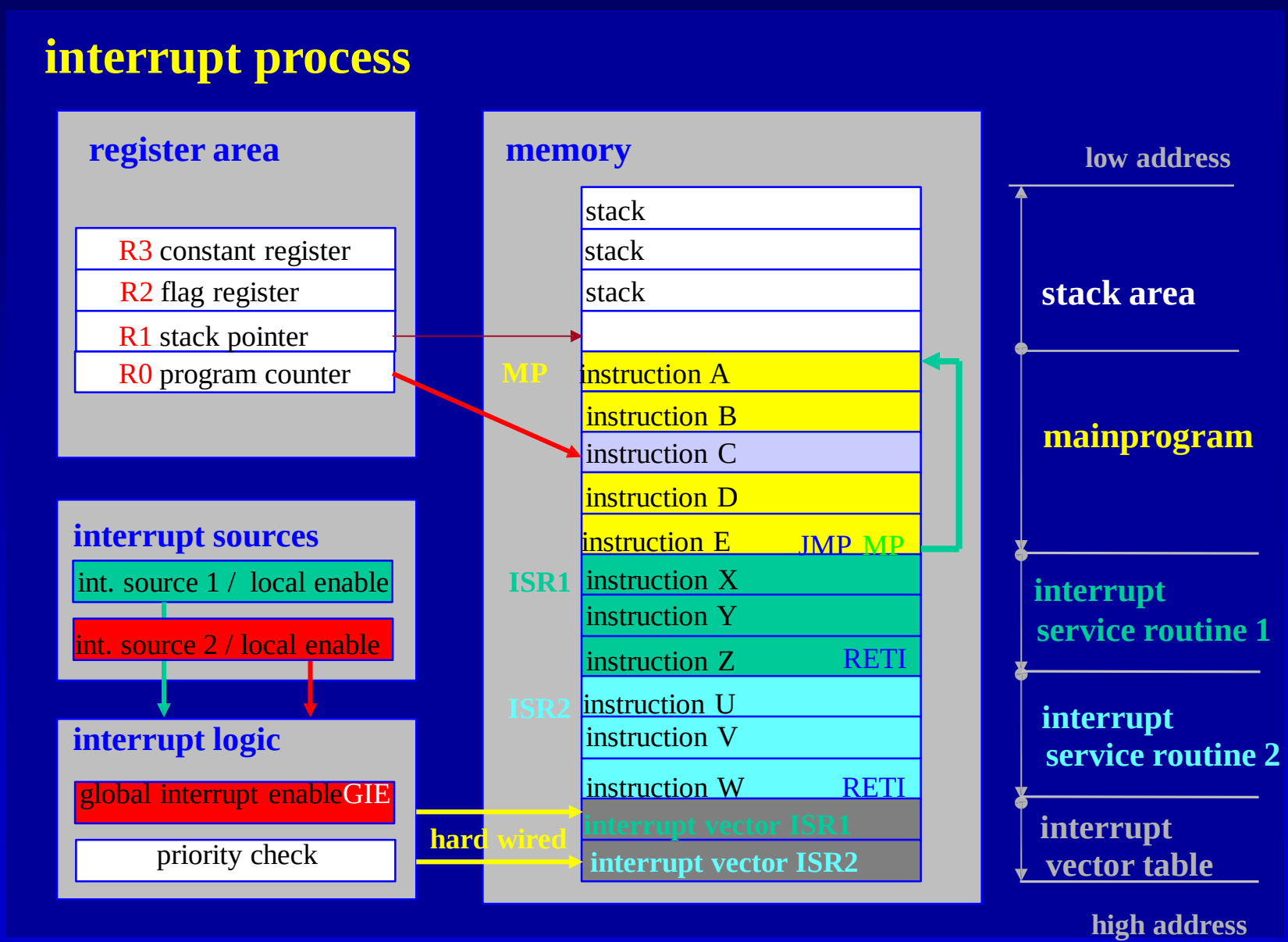
What Happens when an Interrupt Is Requested?



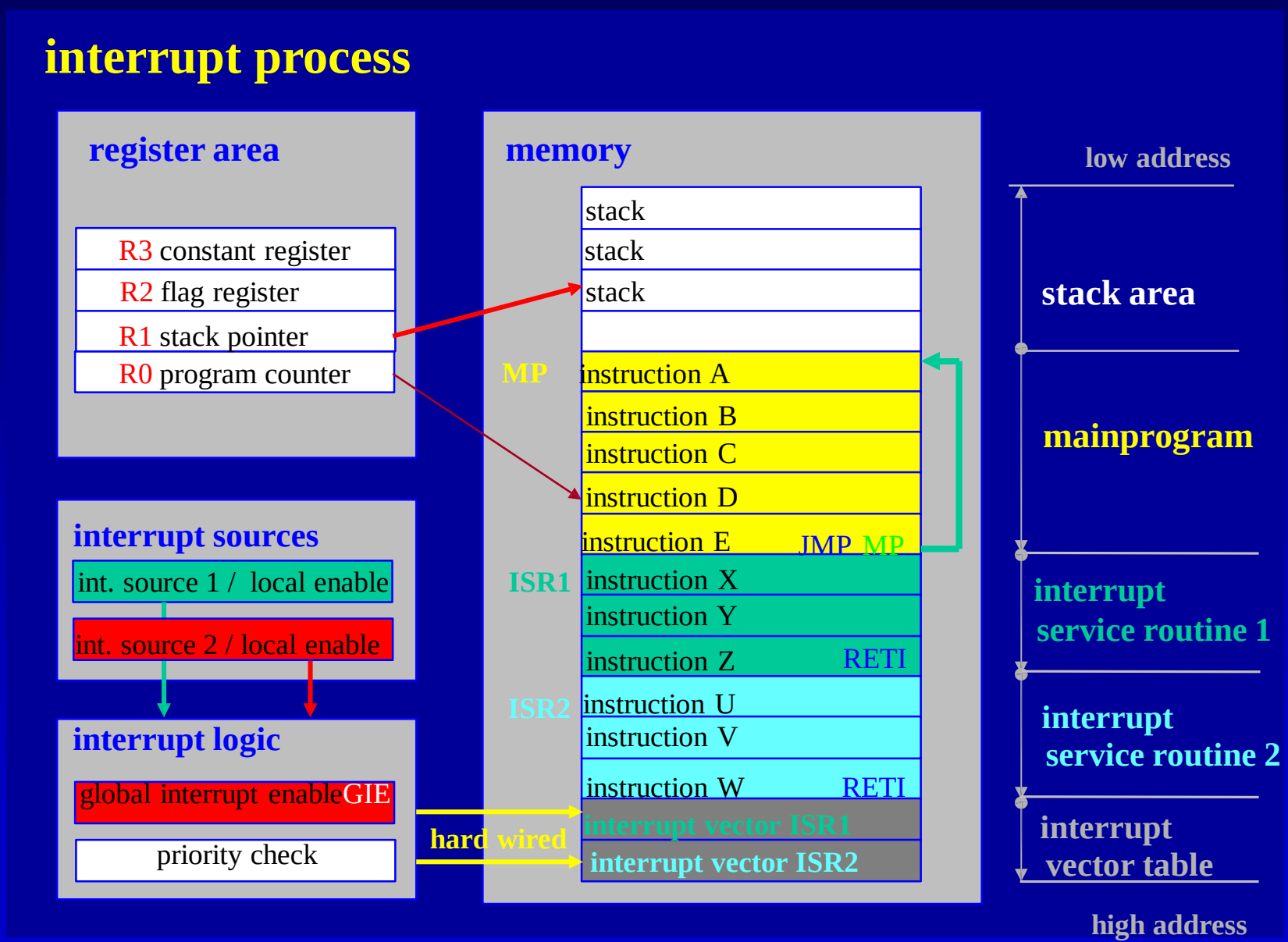
What Happens when an Interrupt Is Requested?



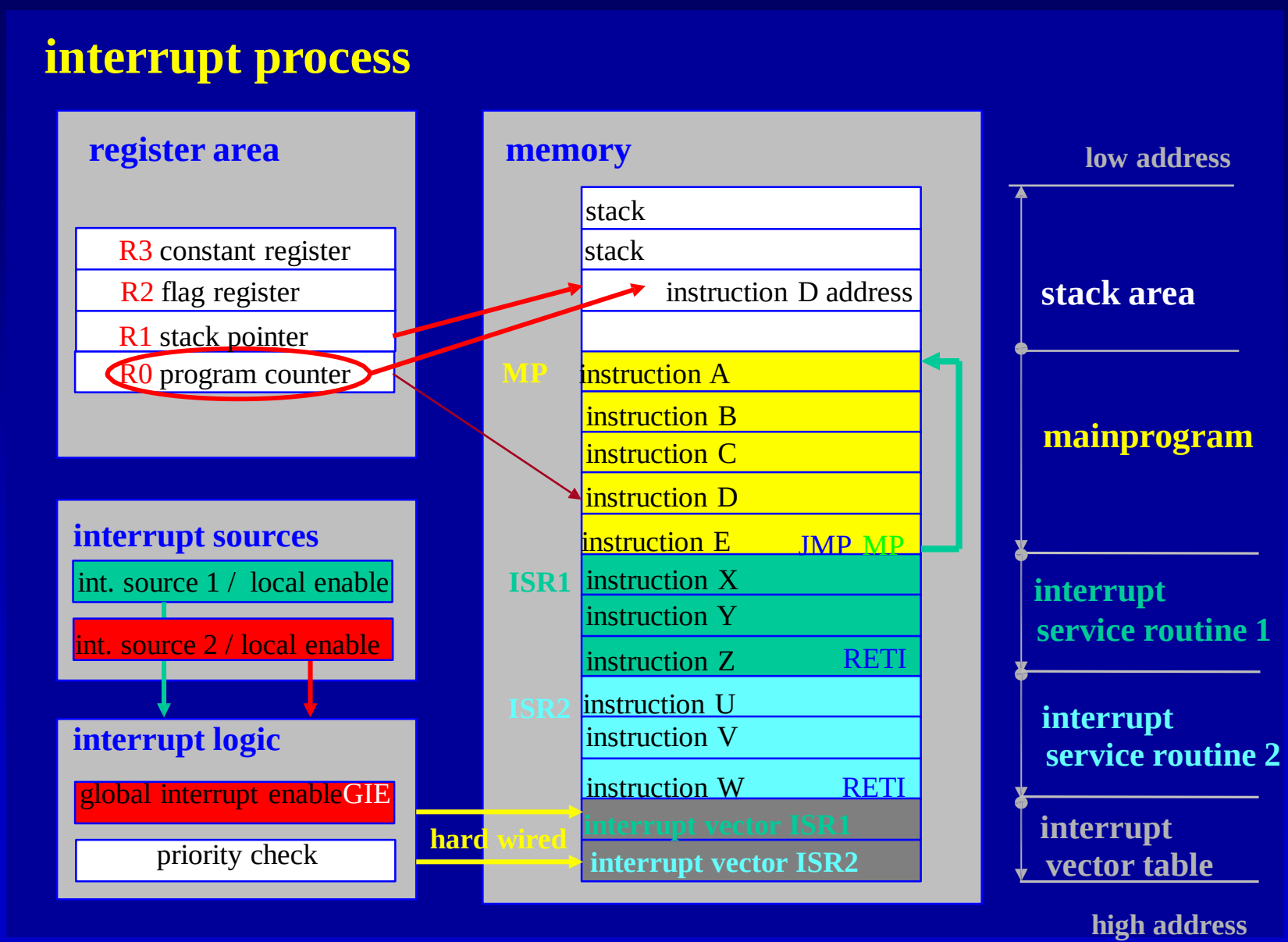
What Happens when an Interrupt Is Requested?



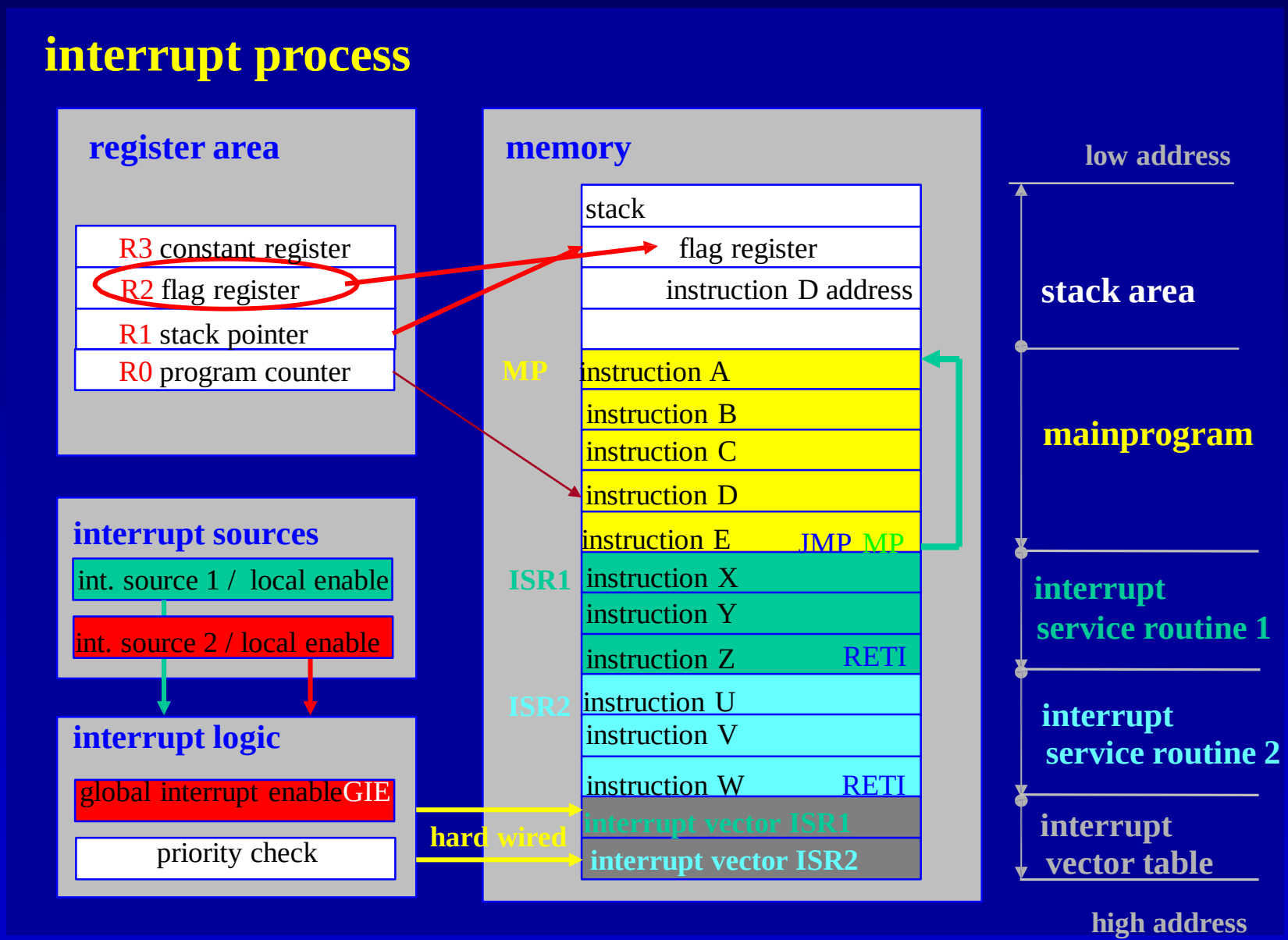
What Happens when an Interrupt Is Requested?



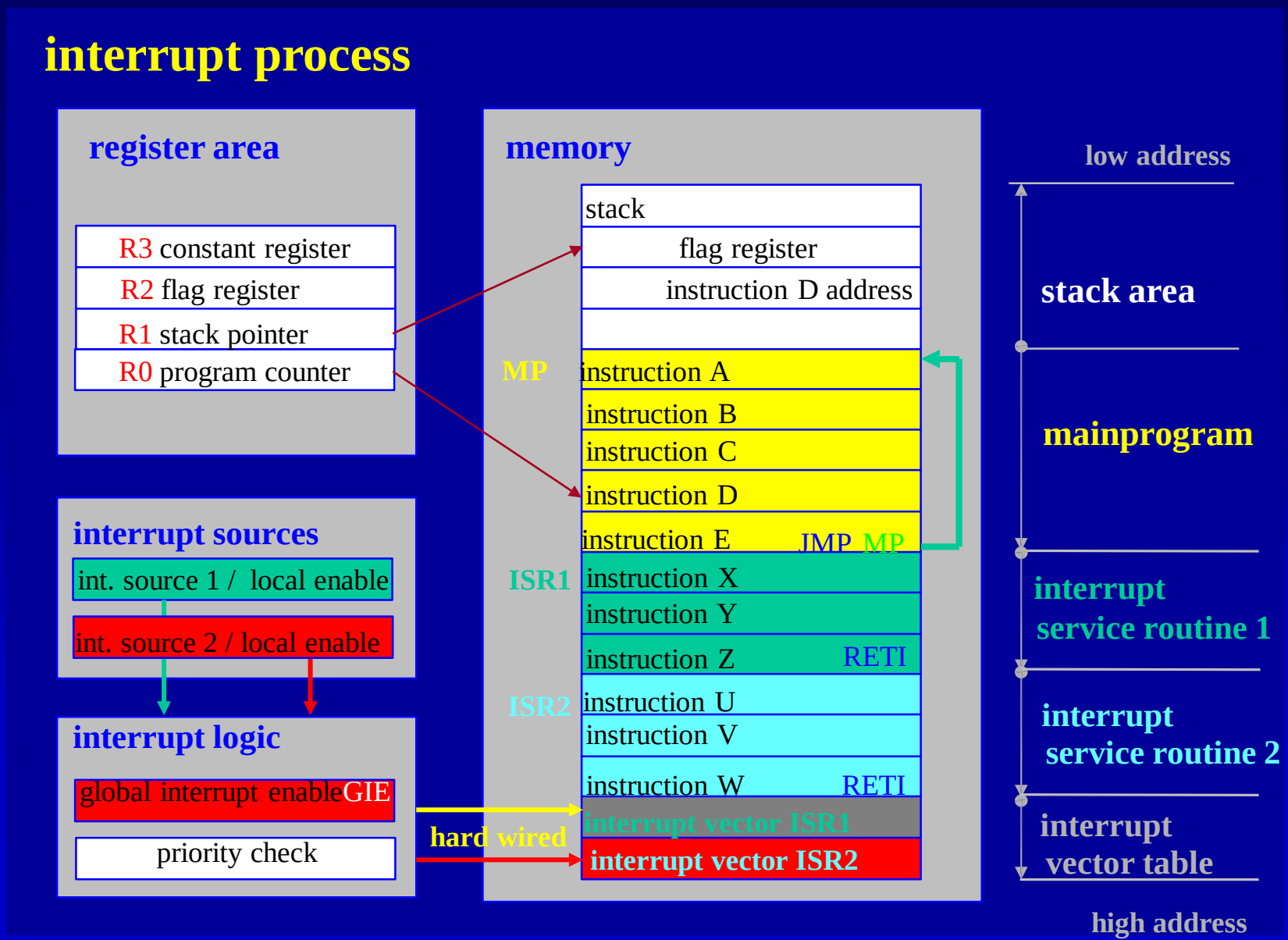
What Happens when an Interrupt Is Requested?



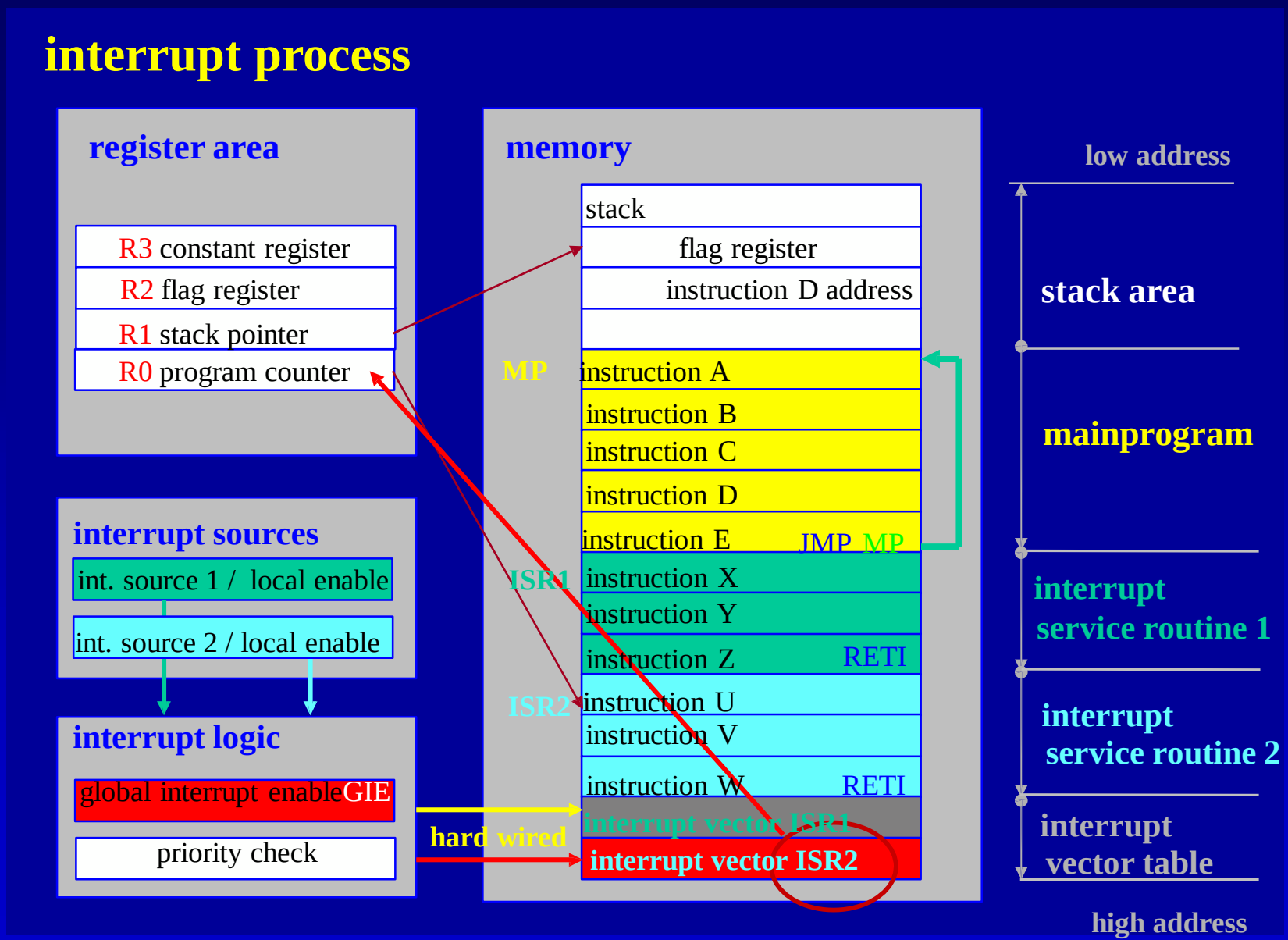
What Happens when an Interrupt Is Requested?



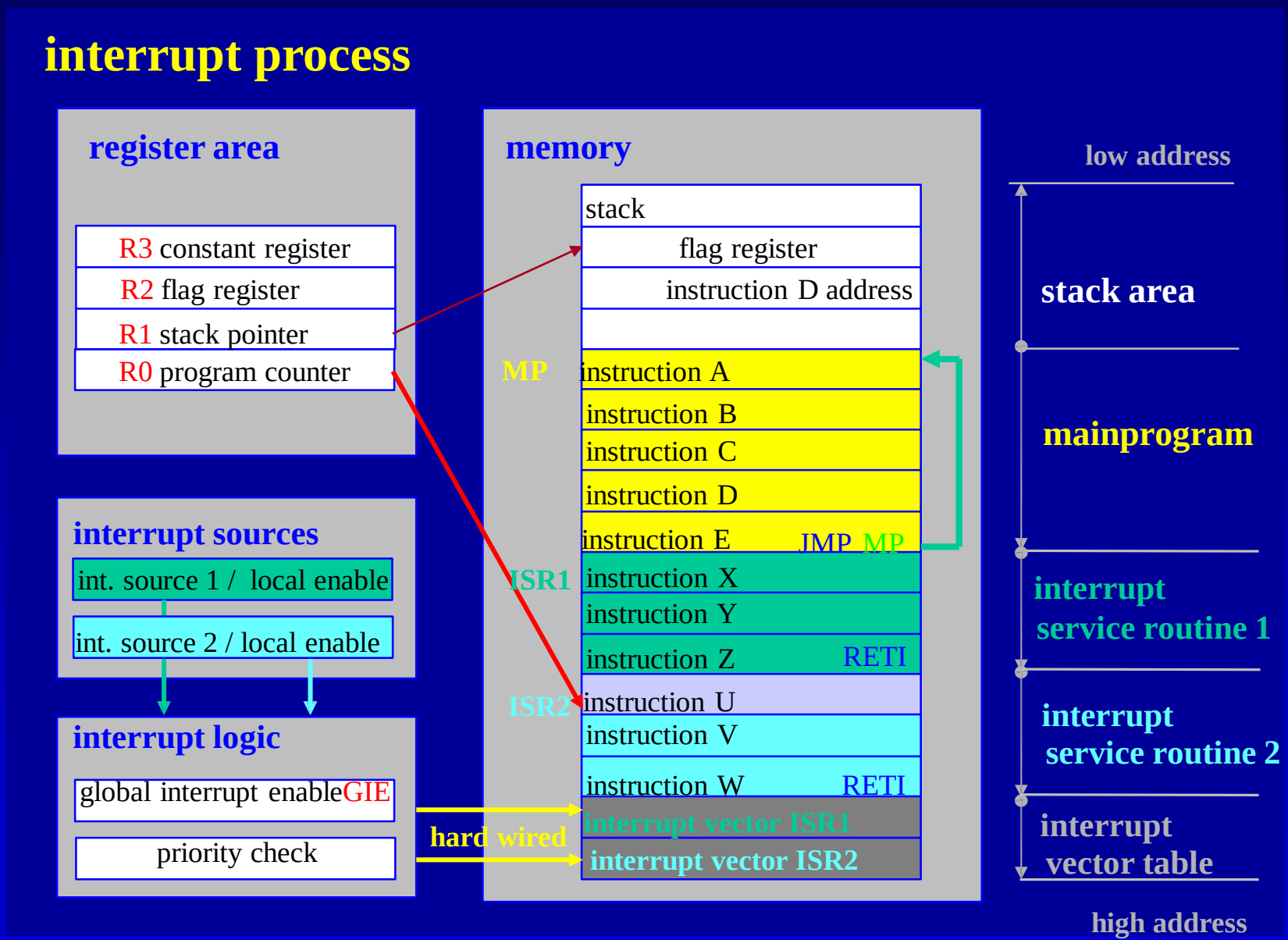
What Happens when an Interrupt Is Requested?



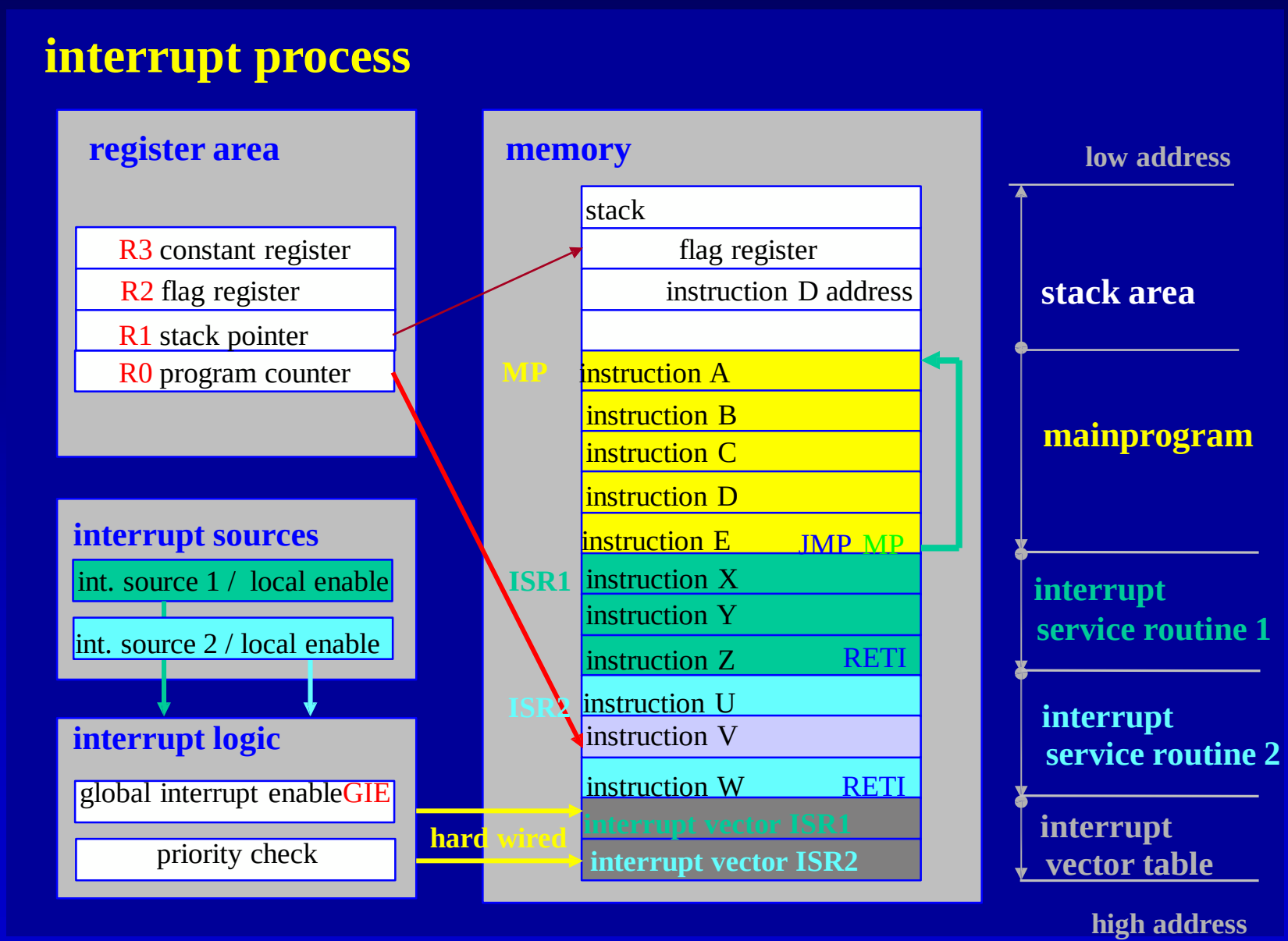
What Happens when an Interrupt Is Requested?



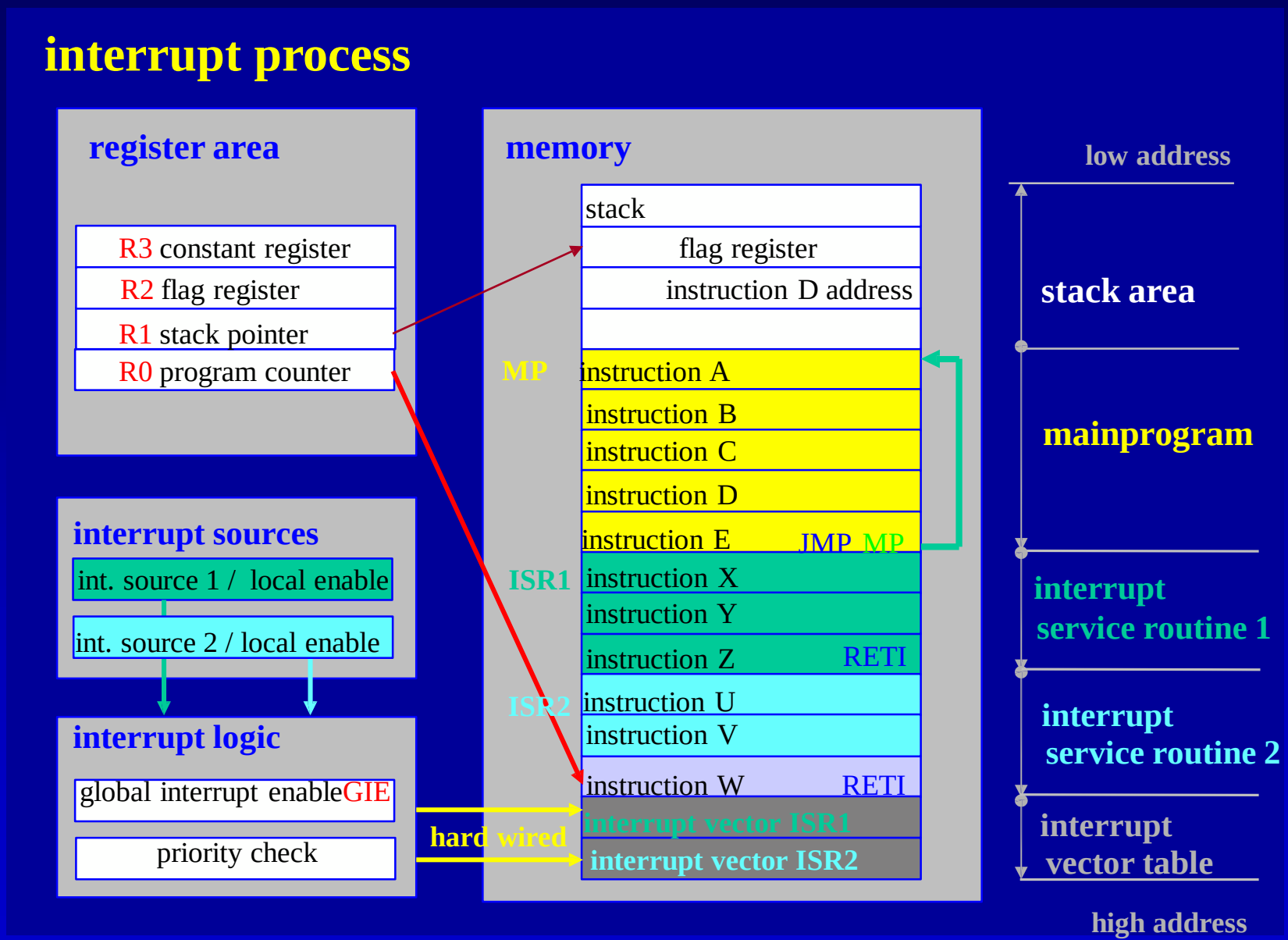
What Happens when an Interrupt Is Requested?



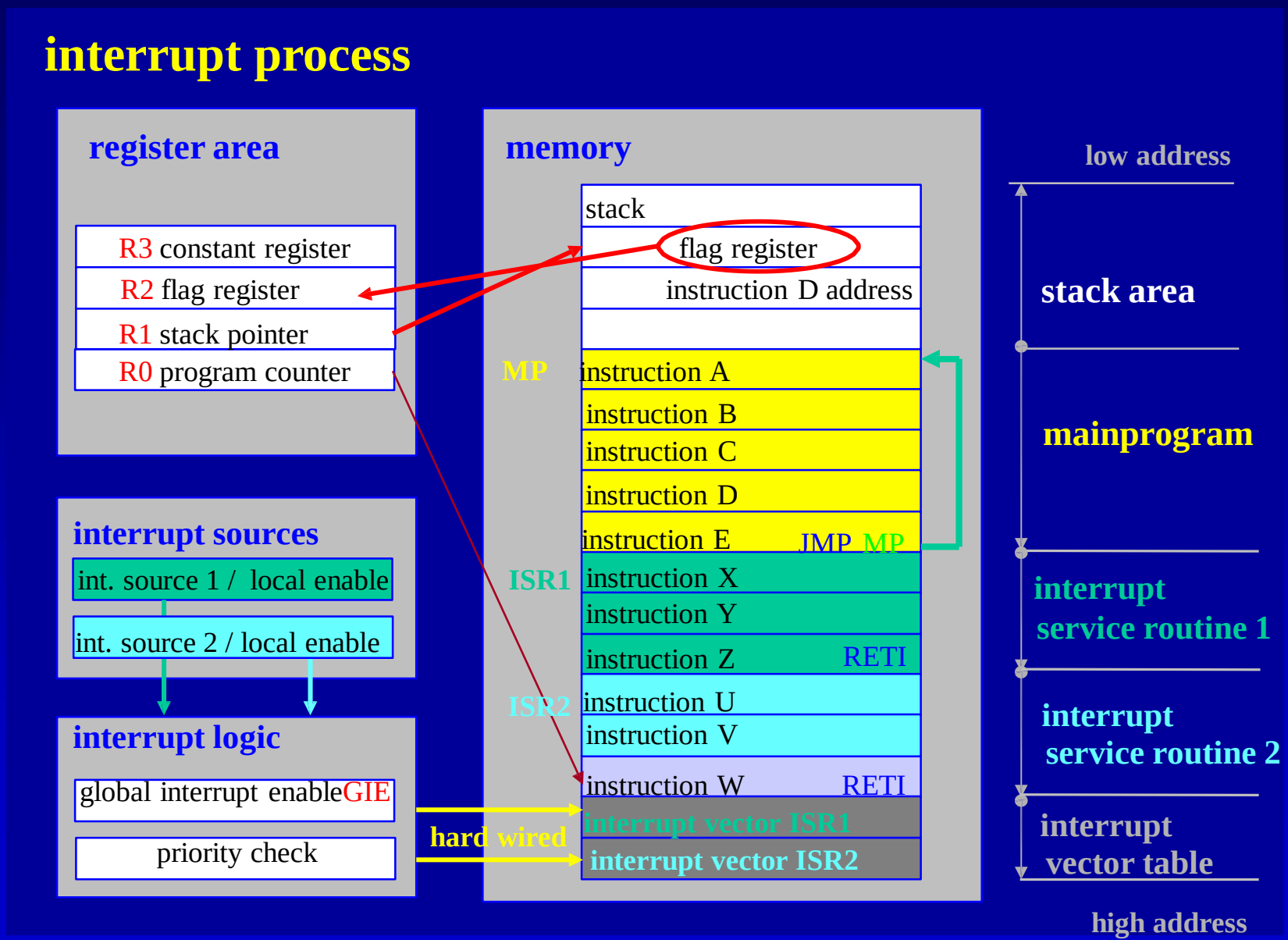
What Happens when an Interrupt Is Requested?



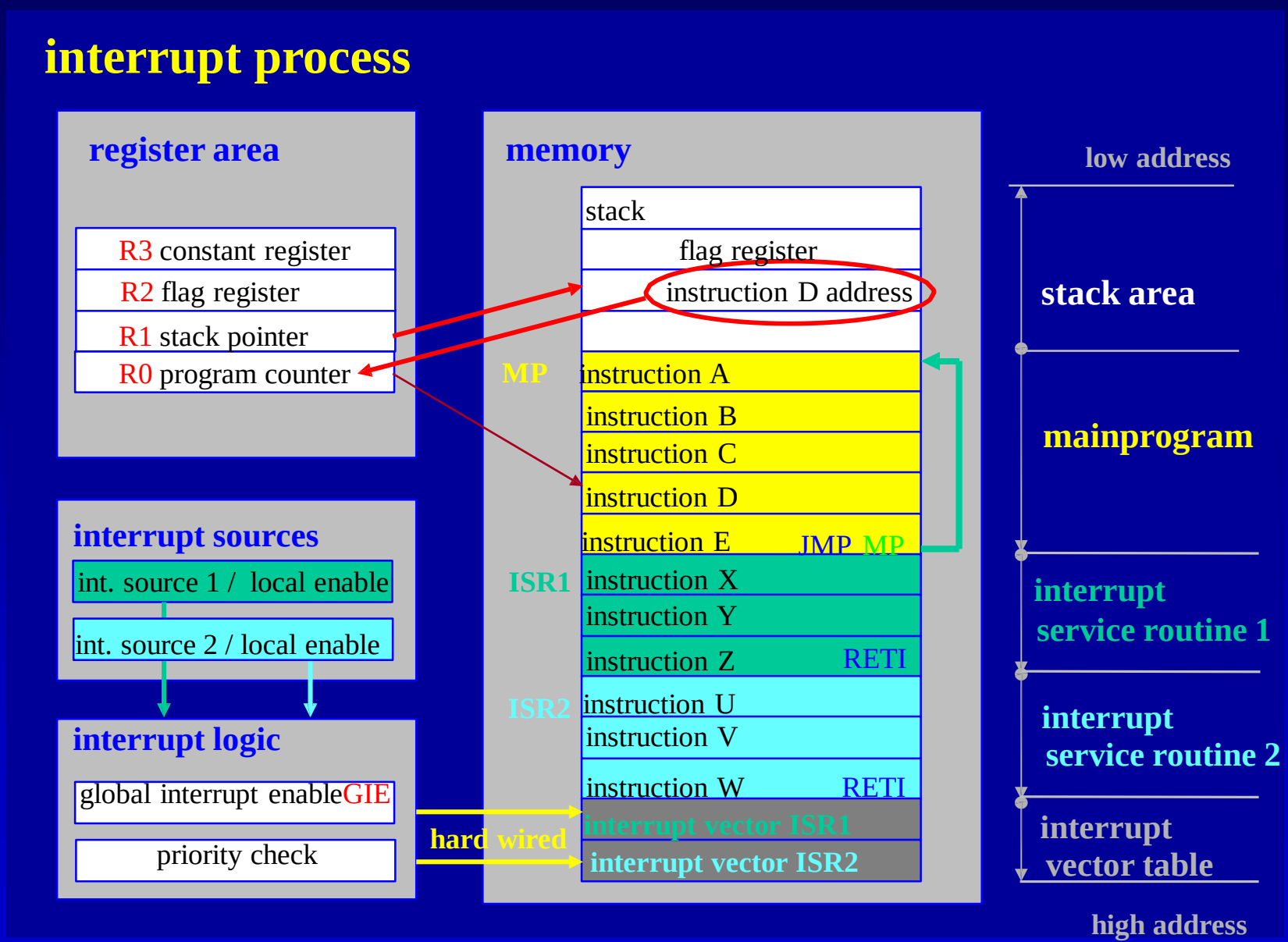
What Happens when an Interrupt Is Requested?



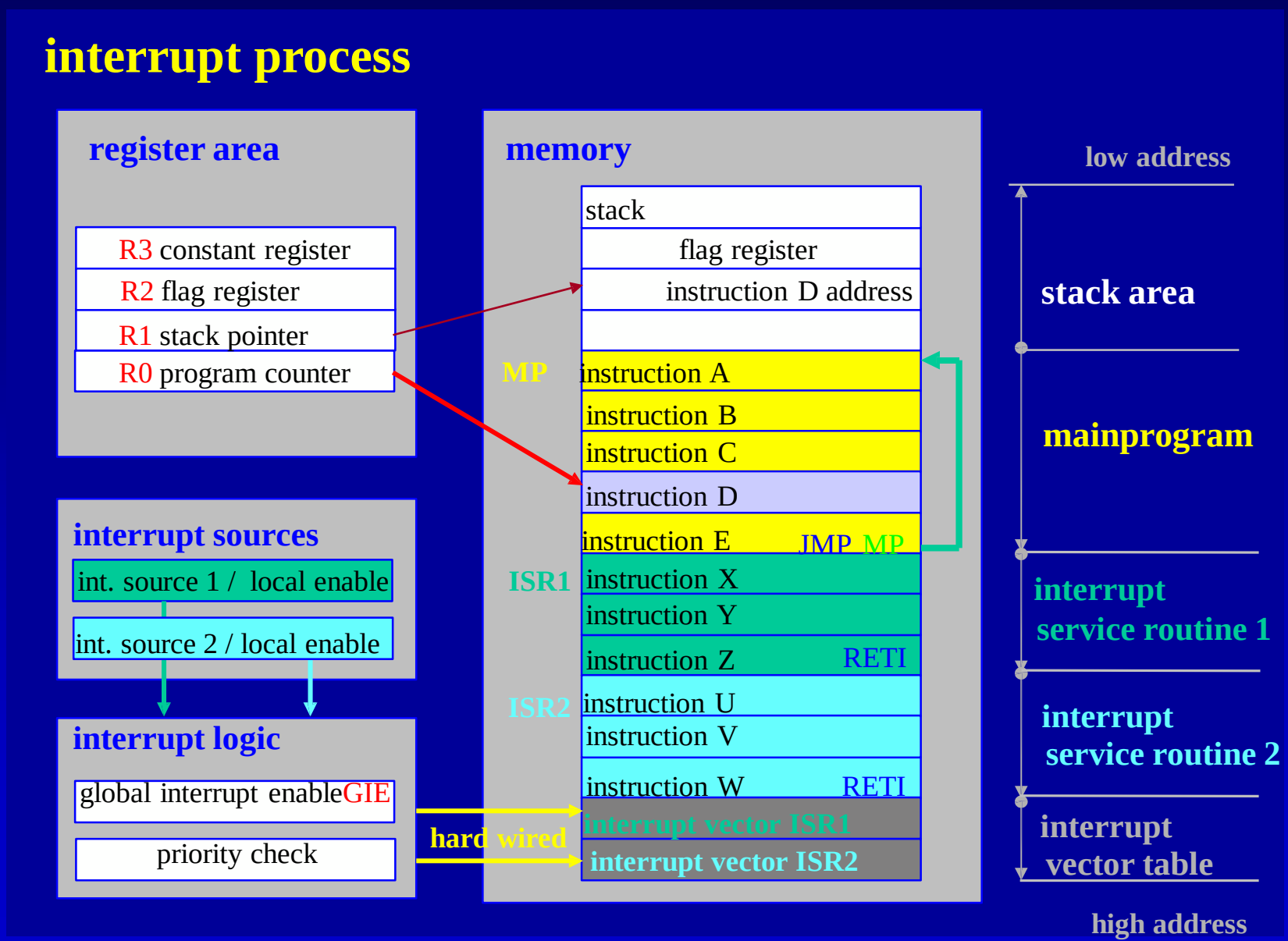
What Happens when an Interrupt Is Requested?



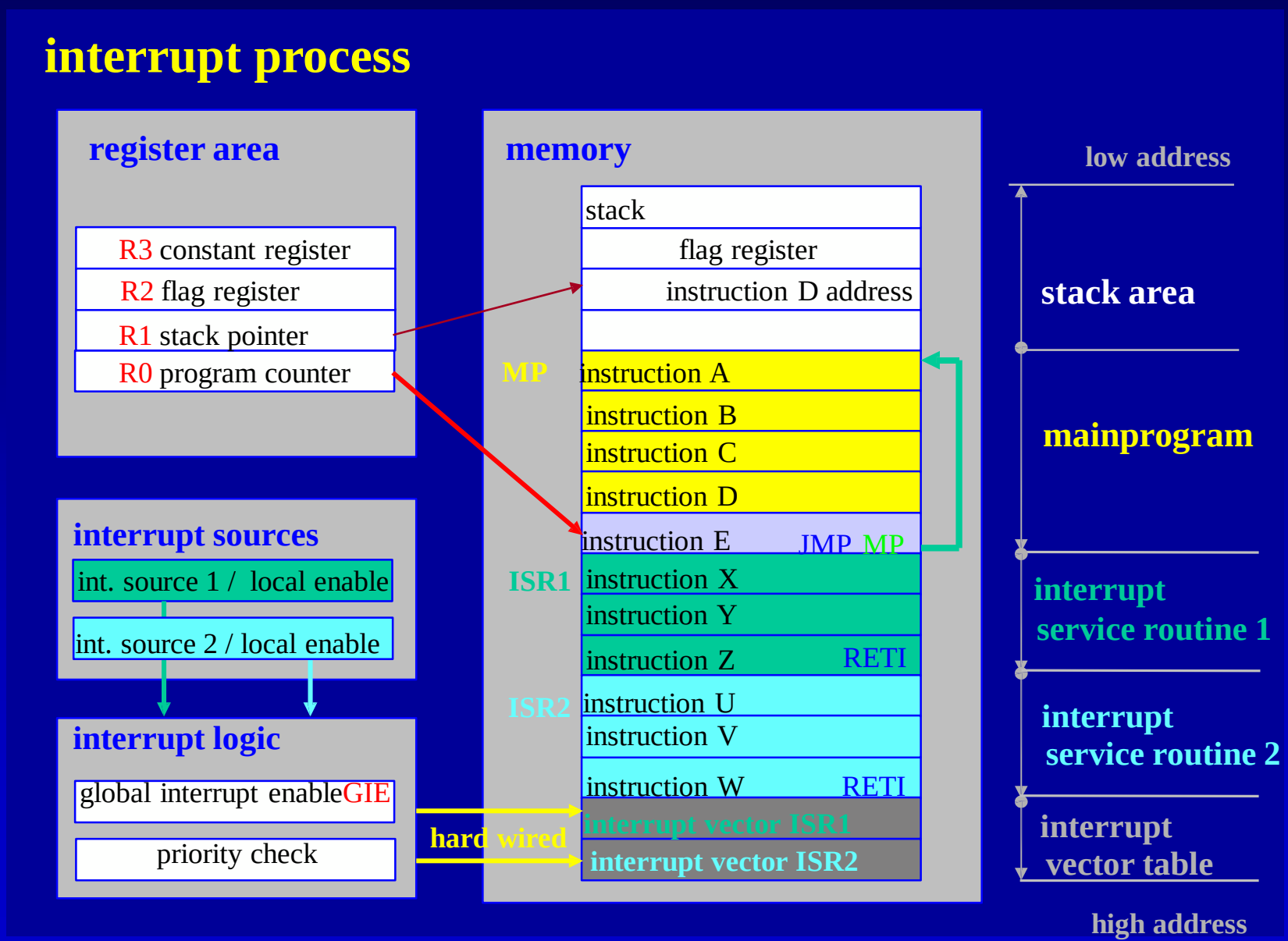
What Happens when an Interrupt Is Requested?



What Happens when an Interrupt Is Requested?



What Happens when an Interrupt Is Requested?



What Happens when an Interrupt Is Requested?

Hardware performs the following steps to launch the ISR:

- Any currently executing instruction is completed if the CPU was active when the interrupt was requested. MCLK is started if the CPU was off.
- The PC, which points to the next instruction, is pushed onto the stack.
- The SR is pushed onto the stack.
- The interrupt with the highest priority is selected if multiple interrupts are waiting for service.
- The interrupt request flag is cleared automatically for vectors that have a single source. Flags remain set for servicing by software if the vector has multiple sources
- The SR is cleared, which has two effects. First, further maskable interrupts are disabled because the GIE bit is cleared; non-maskable interrupts remain active. Second, it terminates any low-power mode
- The interrupt vector is loaded into the PC and the CPU starts to execute the interrupt service routine at that address.

What happens when interrupt occurs

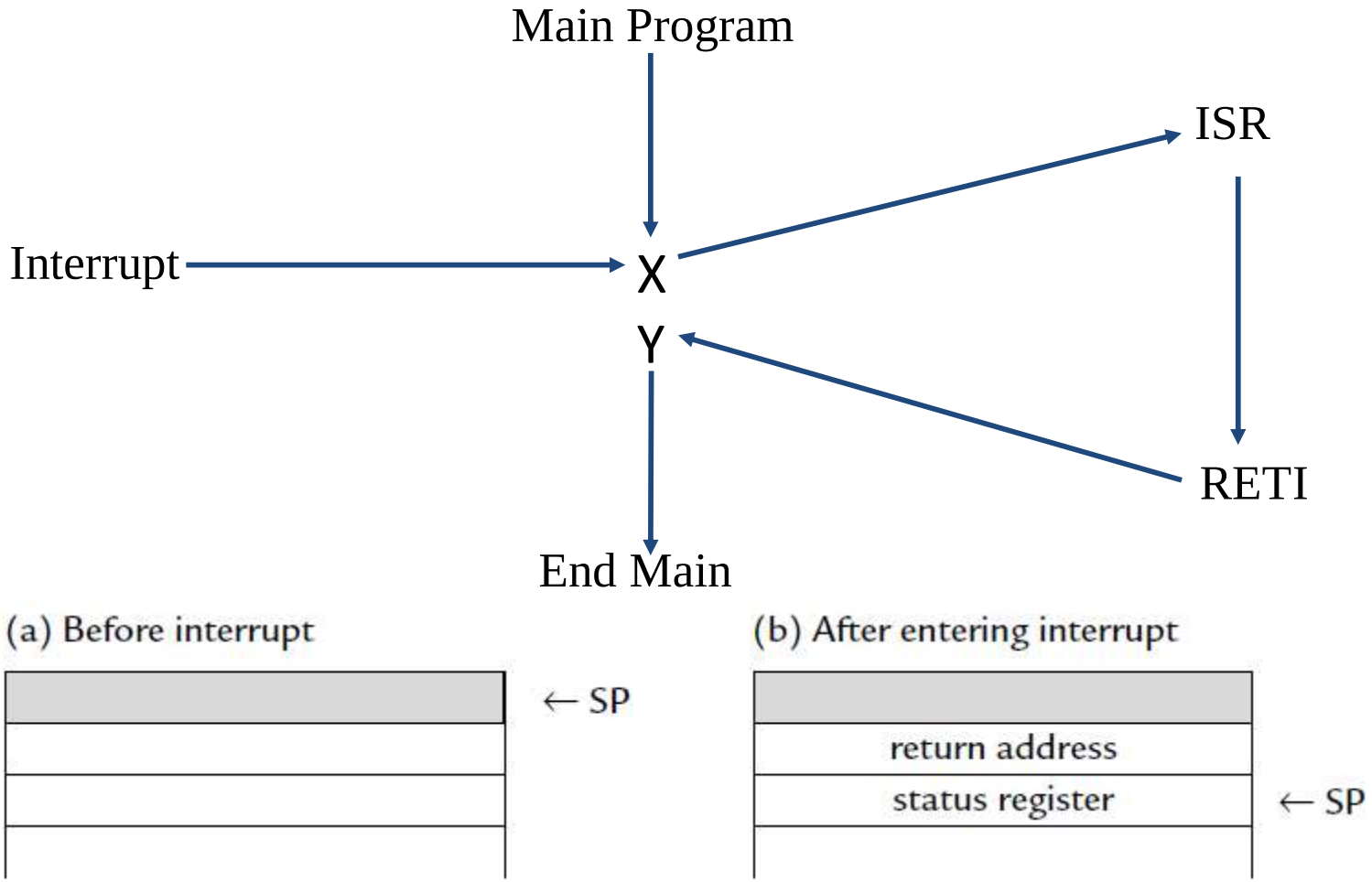
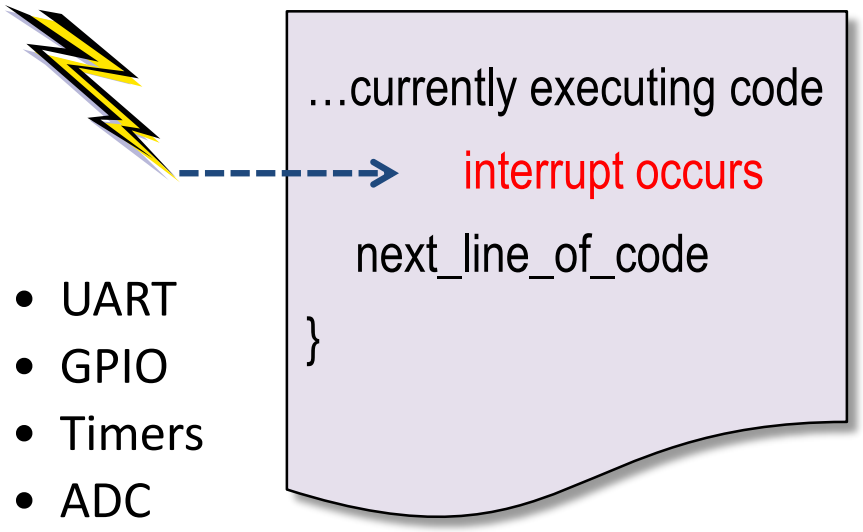


Figure: Stack before and after entering an interrupt service routine. The return address (PC) and status register (SR) have been saved, with SR on the top of the stack.

How do Interrupts Work?

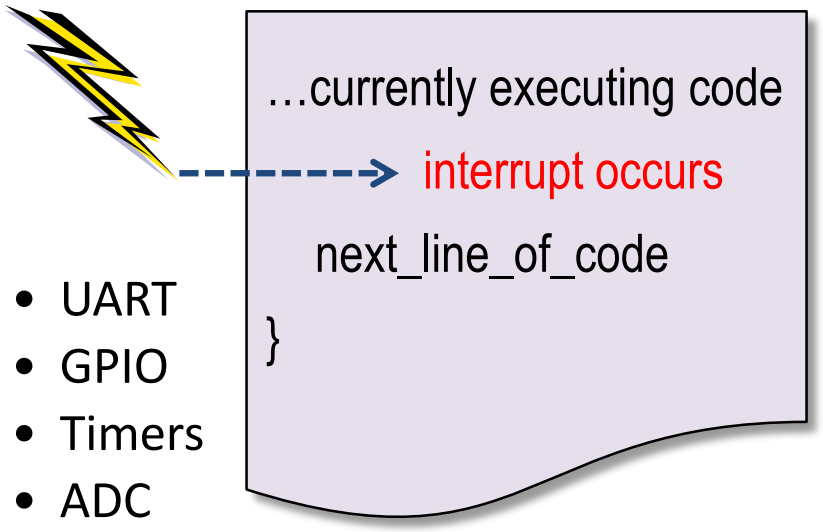
1. **An interrupt occurs**



- UART
- GPIO
- Timers
- ADC
- Etc.

How do Interrupts Work?

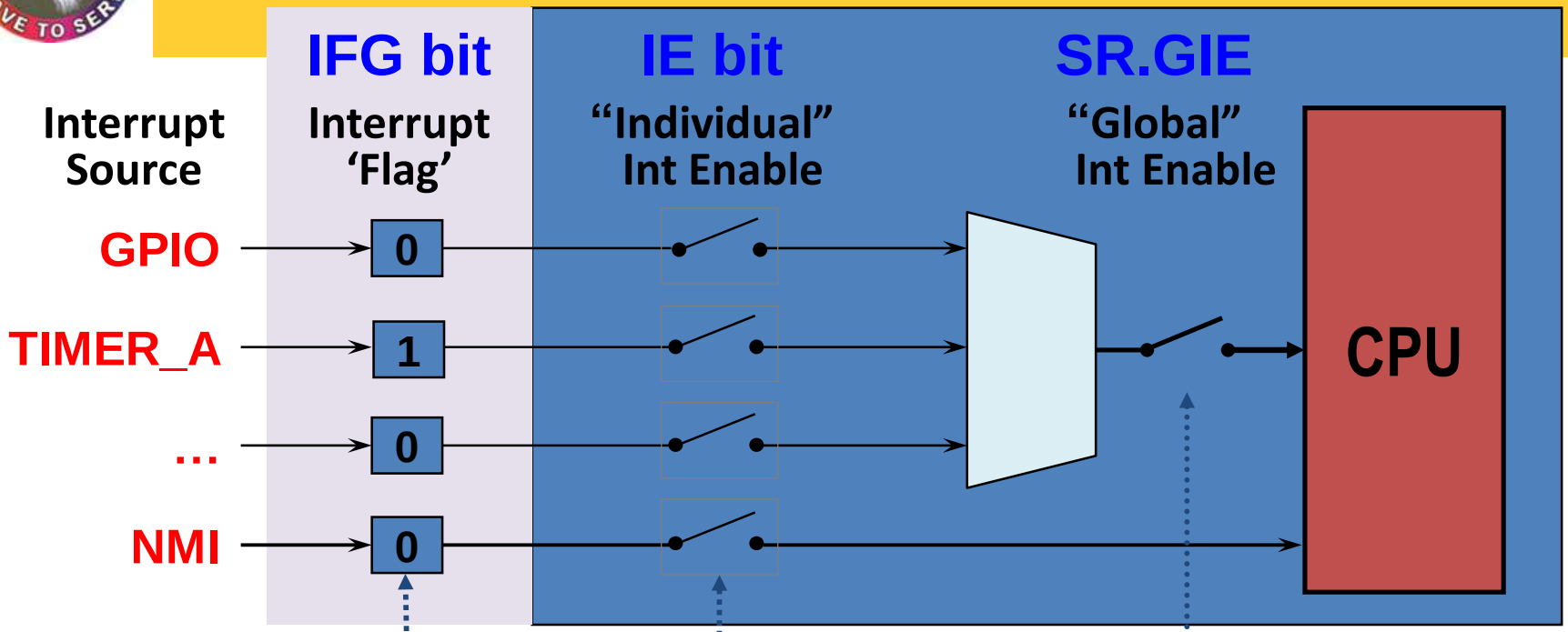
1. **An interrupt occurs**



- UART
- GPIO
- Timers
- ADC
- Etc.

2. **It sets a flag bit in a register**

Interrupt Flow



NMI

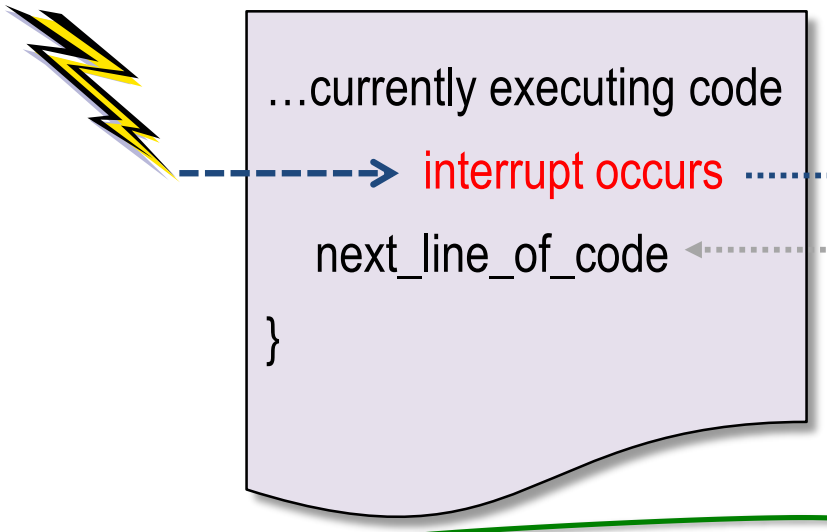
The MSP430 supports a Non-Maskable Interrupt (NMI). This interrupt is not disabled by the GIE bit. It's main used for critical external system events.

Global Interrupt Enable (GIE)
Enables ALL maskable interrupts

Enable: __bis_SR_register(GIE);
Disable: __bic_SR_register(GIE);

Nandikotkur Road,
balli, KURNOOL

Interrupt Service Routine (ISR)



Vector Table

```
&myISR
```

```
#pragma vector=WDT_VECTOR
interrupt myISR(void) {
```

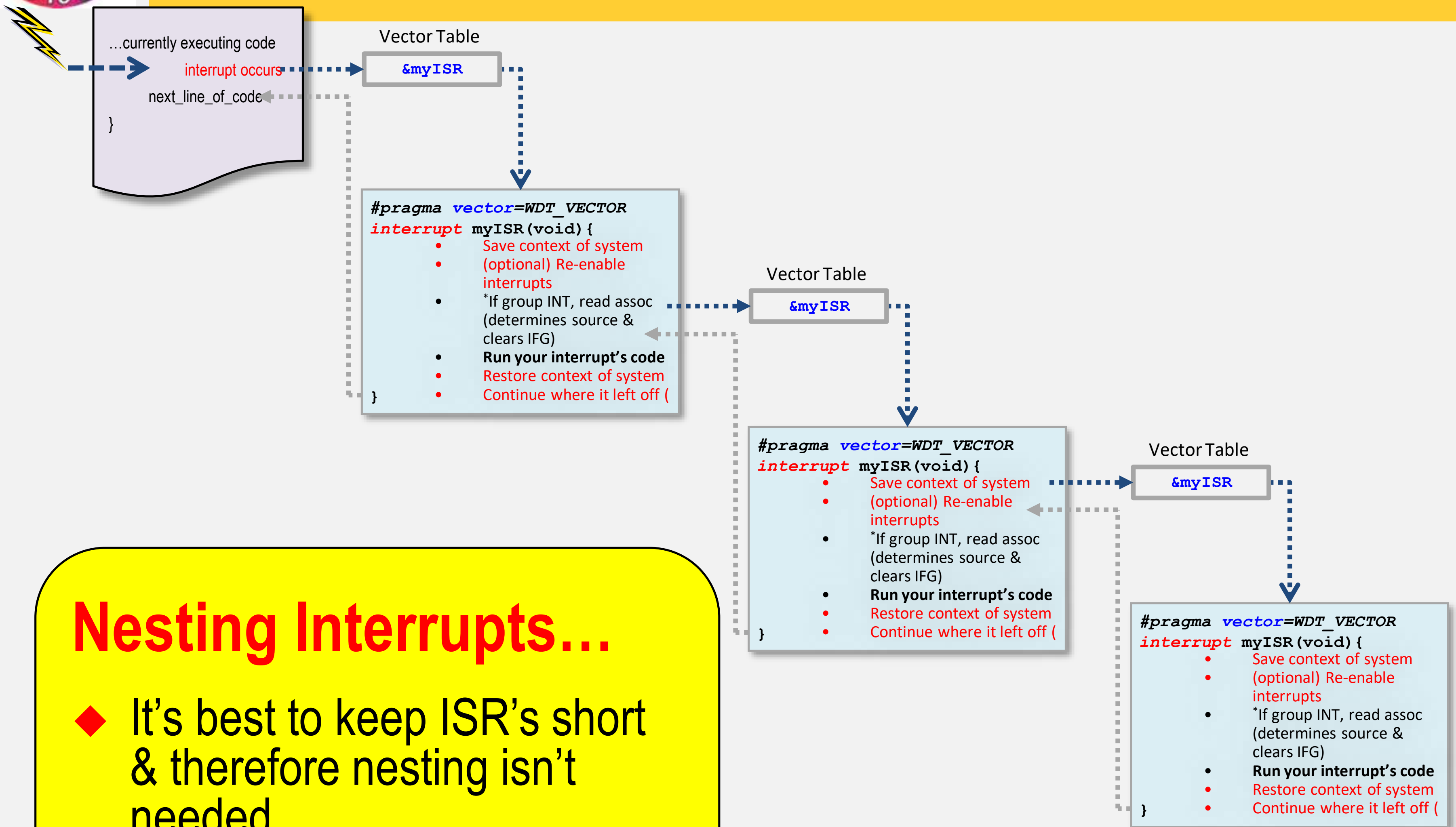
- Save context of system
- (optional) Re-enable interrupts
- *If group INT, read assoc IV Reg (determines source & clears IFG)
- **Run your interrupt's code**
- Restore context of system
- Continue where it left off (RETI)

```
}
```

Using Interrupt Keyword

- Compiler handles context save/restore
- Call a function? Then full context is saved
- No arguments, no return values
- You cannot call any TI-RTOS scheduler functions (e.g. Swi_post)
- Nesting interrupts is MANUAL

Sidebar on Nesting Interrupts



Nesting Interrupts…

- It's best to keep ISR's short & therefore nesting isn't needed

Interrupt Priorities (F5529)

INT Source	Priority
System Reset	high
System NMI	
User NMI	
Comparator	
Timer B (CCIFG0)	
Timer B	
WDT Interval Timer	
Serial Port (A)	
Serial Port (B)	
A/D Convertor	
GPIO (Port 1)	
GPIO (Port 2)	
Real-Time Clock	low

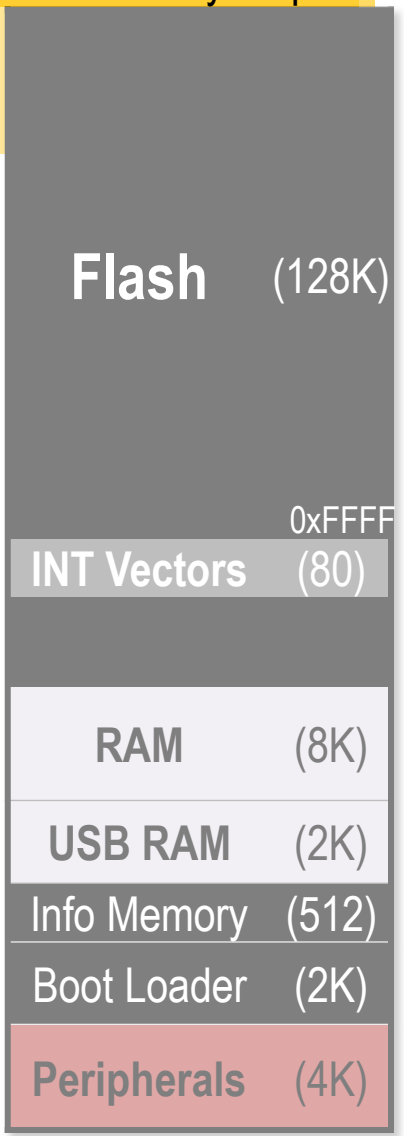
- **There are 23 interrupts** (partially shown here)
- **If multiple interrupts (of the 23) are pending, the highest priority is responded to first**
- **By default, interrupts are not nested …**
  - That is, unless you re-enable INT's during your ISR, other interrupts will be held off until it completes
  - It doesn't matter if the new INT is a higher priority
  - As already recommended, you should keep your ISR's short
- **Most of these represent 'groups' of interrupt source flags**
  - 145 IFG's map into these 23 interrupts

Interrupt Vectors & Priorities (F5529)

INT Source	IV Register	Vector Address	Loc'n
System Reset	SYSRSTIV	RESET_VECTOR	63
System NMI	SYSSNIV	SYSNMI_VECTOR	62
User NMI	SYSUNIV	UNMI_VECTOR	61
Comparator	CBIV	COMP_B_VECTOR	60
Timer B (CCIFG0)	CCIFG0	TIMER0_B0_VECTOR	59
Timer B	TB0IV	TIMER0_B1_VECTOR	58
WDT Interval Timer	WDTIFG	WDT_VECTOR	57
Serial Port (A)	UCA0IV	USCI_A0_VECTOR	56
Serial Port (B)	UCB0IV	USCI_B0_VECTOR	55
A/D Convertor	ADC12IV	ADC12_VECTOR	54
GPIO (Port 1)	P1IV	PORT1_VECTOR	47
GPIO (Port 2)	P12V	PORT2_VECTOR	42
Real-Time Clock	RTCIV	RTC_VECTOR	41

Priority: high → low

Memory Map



Legend:
- Non-maskable
- Maskable
- Grouped IFG bits
- Dedicated IFG bits

- **PxIES: Interrupt Edge Select Register -** Each bit of the register selects the interrupt edge for the corresponding I/O pin. When the bit is set, the PxIFGx flag is set with a high-to-low transition.

  - Here, the bit is set to interrupt on high to low transition.

- **PxIFG: Interrupt Flag Register -** Each PxIFGx bit is the interrupt flag for its corresponding I/O pin and is set when the selected input signal edge transition occurs at the pin.

- **PxIE: Interrupt Enable -** Each PxIE bit enables the associated PxIFG interrupt flag. Here P1.3 is interrupt enabled.

WatchDog Timer

- This is a safety feature, which resets the processor if the program becomes stuck in an infinite loop/hang.
- If the selected time interval expires, a system reset is generated.
- The operation of the watchdog is controlled by the 16-bit register **WDTCTL**

- *If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.*

Contd..

Features of the watchdog timer module include:

- Eight software-selectable time intervals
- Watchdog mode
- Interval mode
- Access to WDT Control register is password protected
- Selectable clock source
- Can be stopped to conserve power
- Clock fail-safe feature

Contd..

Watchdog Mode:

- After a PUC condition, the WDT module is configured in the watchdog mode with an initial ~32-ms reset interval using the SMCLK.
- When the watchdog timer is configured to operate in watchdog mode, either writing to WDTCTL with an incorrect password, or expiration of the selected time interval triggers a PUC.

Interval Timer Mode:

- Setting the WDTTMSEL bit to 1 selects the interval timer mode. This mode can be used to provide periodic interrupts. In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval. A PUC is not generated in interval timer mode

Contd..

Clock Fail-Safe Feature:

- The WDT_A provides a fail-safe clocking feature, ensuring the clock to the WDT_A cannot be disabled while in watchdog mode. This means that the low-power modes may be affected by the choice for the WDT_A clock.

- If SMCLK or ACLK fails as the WDT_A clock source, VLOCLK is automatically selected as the WDT_A clock source.

- When the WDT_A module is used in interval timer mode, there is no fail-safe feature within WDT_A for the clock source.

Watchdog Timer Control (WDTCTL) Register: Contd..

WDTCTL Register

15	14	13	12	11	10	9	8
WDTPW							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
WDTHOLD	WDTSSEL		WDTTMSEL	WDTCNTCL	WDTIS		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	rw-0	rw-0

RCEW, Pasupula (V), Nandikotkur Road,
Near Venkayapalli, KURNOOL

				1b = Watchdog timer is stopped.
6-5	WDTSSEL	RW	0h	Watchdog timer clock source select 00b = SMCLK 01b = ACLK 10b = VLOCLK 11b = X_CLK; VLOCLK in devices that do not support X_CLK
4	WDTTMSEL	RW	0h	Watchdog timer mode select 0b = Watchdog mode 1b = Interval timer mode
3	WDTCNTCL	RW	0h	Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset. 0b = No action 1b = WDTCNT = 0000h
2-0	WDTIS	RW	4h	Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC. 000b = Watchdog clock source /(2^31) (18h:12m:16s at 32.768 kHz) 001b = Watchdog clock source /(2^27) (01h:08m:16s at 32.768 kHz) 010b = Watchdog clock source /(2^23) (00h:04m:16s at 32.768 kHz) 011b = Watchdog clock source /(2^19) (00h:00m:16s at 32.768 kHz) 100b = Watchdog clock source /(2^15) (1 s at 32.768 kHz) 101b = Watchdog clock source /(2^13) (250 ms at 32.768 kHz) 110b = Watchdog clock source /(2^9) (15.625 ms at 32.768 kHz) 111b = Watchdog clock source /(2^6) (1.95 ms at

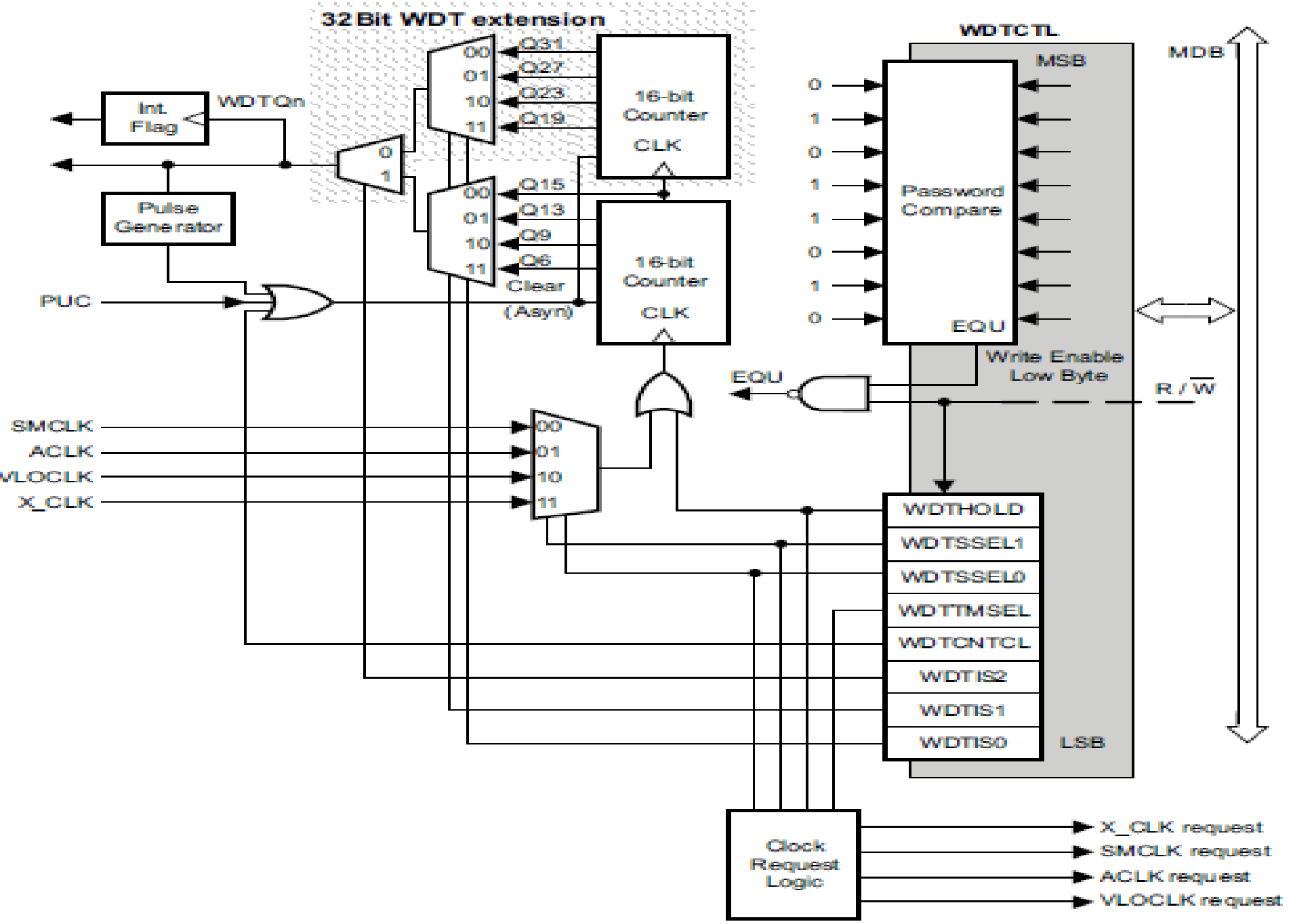


Figure: Watchdog Timer Block Diagram

SYSTEM CLOCKs (CLOCK SYSTEM)

- All microcontrollers contain a clock module to drive the CPU and peripherals
- clock signal is a square wave whose edges trigger hardware throughout the device so that the changes in different components are synchronized.
- A crystal with a frequency of a few MHz would be connected to two pins. It would drive the CPU directly and was typically divided down by a factor of 2 or 4 for the main bus
- Modern Microcontrollers demands high performance and low power complicated clocks, often with two or more sources

Contd..

- **Two clocks with quite different specifications are often needed:**
  - A fast clock to drive the CPU, which can be started and stopped rapidly to conserve energy but usually need not be particularly accurate.
  - A slow clock that runs continuously to monitor real time, which must therefore use little power and may need to be accurate.

Contd..

Several types of oscillator are used to generate the clock signal. In these **Crystal** & **RC** are most common

Crystal: Accurate and stable.

- Crystals for microcontrollers typically run at either a high frequency of a few MHz to drive the main bus or a low frequency of 32,768 Hz for a real-time clock.
- The disadvantages are that crystals are expensive and delicate, the oscillator draws a relatively large current, particularly at high frequency.
- the crystal is an extra component and may need two capacitors as well. Crystal oscillators also take a long time to start up and stabilize, often around 10^5 cycles

Contd..

Resistor and capacitor *(RC):*

- Cheap and quick to start but used to have poor accuracy and stability.
- The components can be external but are now more likely to be integrated within the MCU.
- The quality of integrated *RC oscillators has improved dramatically in* recent years
- There are also ceramic resonators that lie between these extremes, being cheaper but less accurate and stable than crystals.

Contd..

The MSP430 uses three internal clocks:

- **Master clock, MCLK**, is used by the CPU and a few faster peripherals.
- **Subsystem master clock, SMCLK**, is distributed to faster peripherals.
- **Auxiliary clock, ACLK,** is also distributed to slower peripherals.

❑ SMCLK & MCLK both in the **M**ega **H**ert**z**(MHz) range.

- ACLK is often derived from a watch crystal and therefore runs at a much lower frequency(32KHz).
- Most peripherals can select their clock from either SMCLK or ACLK.

Contd..

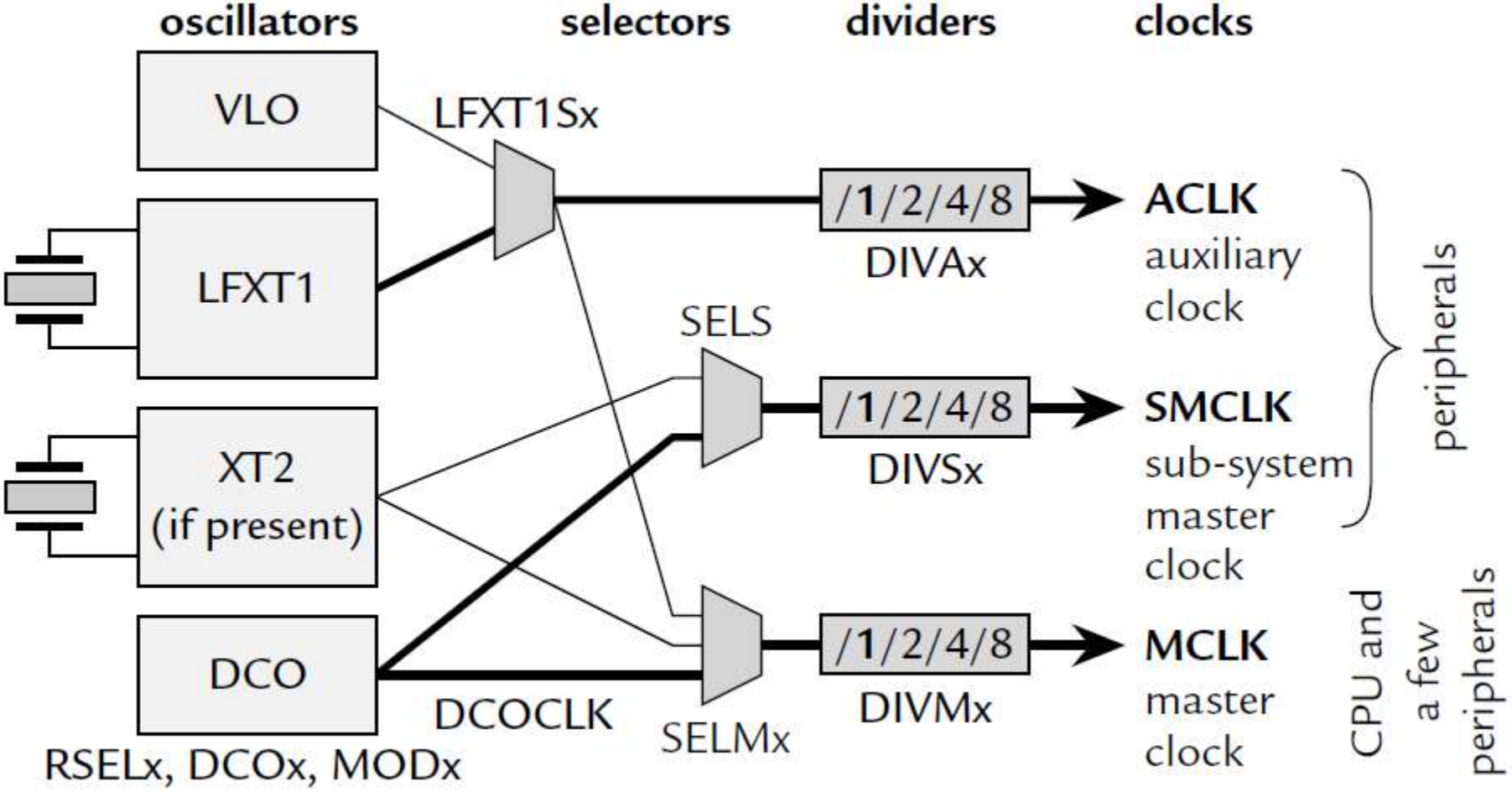


Fig: Simplified block diagram of the clock module of the MSP430F2xx family

Control of the Clock Module through the Status Register

- **CPUOFF** disables MCLK, which stops the CPU and any peripherals that use MCLK.
- **SCG1** disables SMCLK and peripherals that use it.
- **SCG0** disables the DC generator for the DCO
- **OSCOFF** disables VLO and LFXT1.

LOW POWER MODES

The MSP430 is designed for low-power applications and uses different operating modes.

The MSP430 has one active mode and five software selectable low-power modes of operation.

An interrupt event can wake up the device from any of the low-power modes, service the request, and restore back to the low-power mode on return from the interrupt program.

Contd..

The following six operating modes can be configured by software:

- Active mode (AM)
- Low-power mode 0 (LPM0)
- Low-power mode 1 (LPM1)
- Low-power mode 2 (LPM2)
- Low-power mode 3 (LPM3)
- Low-power mode 4 (LPM4)

The Active mode and low-power modes 0 to 4 are configured with the CPUOFF, OSCOFF, SCG0, and SCG1 bits in the status register. The advantage of including the CPUOFF, OSCOFF, SCG0, and SCG1 mode-control bits in the status register is that the present operating mode is saved onto the stack during an interrupt service routine.

Contd..

Active mode: CPU, all clocks, and enabled modules are active, I ≈ 300μA.

LPM0: CPU and MCLK are disabled, SMCLK and ACLK remain active, I ≈ 85μA. This is used when the CPU is not required but some modules require a fast clock from SMCLK and the DCO.

LPM1: CPU, MCLK, SMCLK, DCO and DC generator are disabled, SMCLK and ACLK remains active.

LPM2: CPU, MCLK, SMCLK, DCO are disabled, DC generator and ACLK remains active.

Contd..

LPM3: CPU, MCLK, SMCLK, DCO and DC generator are disabled; only ACLK remains active; I ≈ 1μA. This is the standard low-power mode when the device must wake itself at regular intervals and therefore needs a (slow) clock. It is also required if the MSP430 must maintain a real-time clock.

LPM4: CPU and all clocks are disabled, I ≈ 0.1μA. The device can be wakened only by an external signal. This is also called *RAM retention mode*.

Contd..

CPUOFF → Controls the MCLK

OSCOFF → Controls the VLO & LFXT1

SCG0 → Controls the DC generator

SCG1 → Controls the SMCLK

0 → No Action (or) Active
1 → Action (or) Disables the CLK

Table: Operating Modes of MSP430

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
1	1	1	1	LPM4	CPU and all clocks disabled

Timer_A

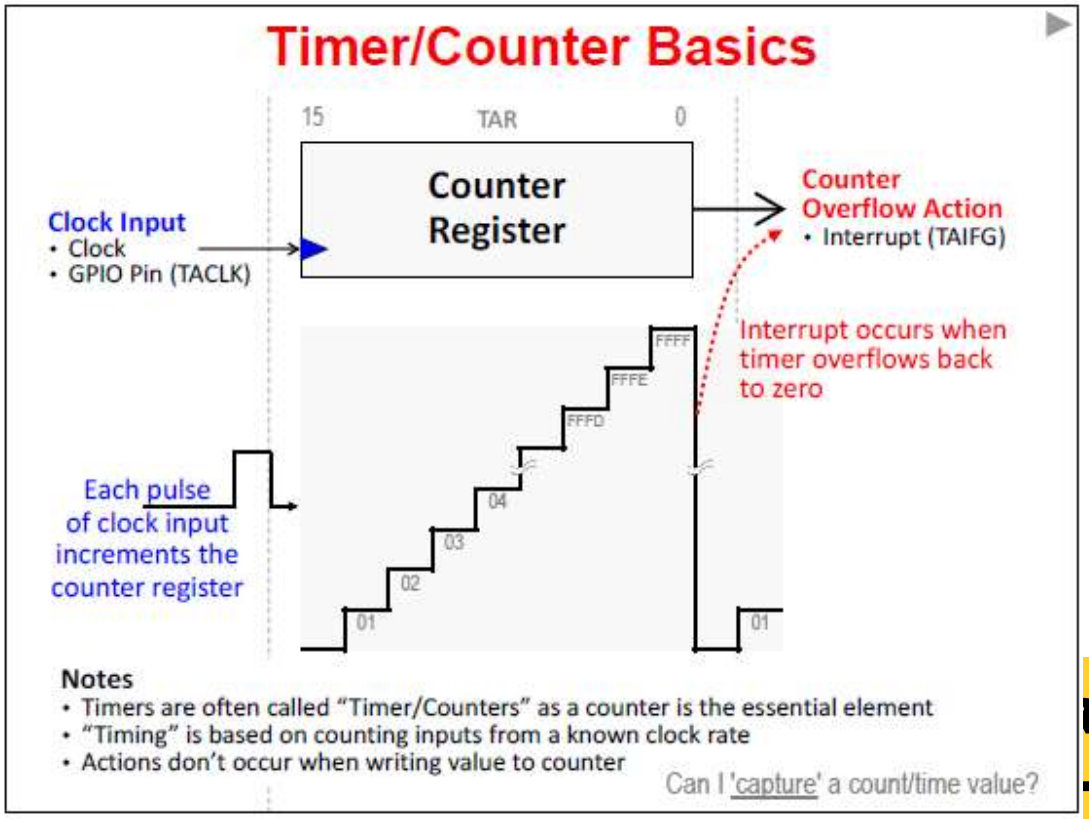
- Timer_A is a 16-bit timer/counter with up to seven capture/compare registers.
- Timer_A can support multiple capture/compares, PWM outputs, and interval timing.
- Timer_A also has extensive interrupt capabilities.

Counter

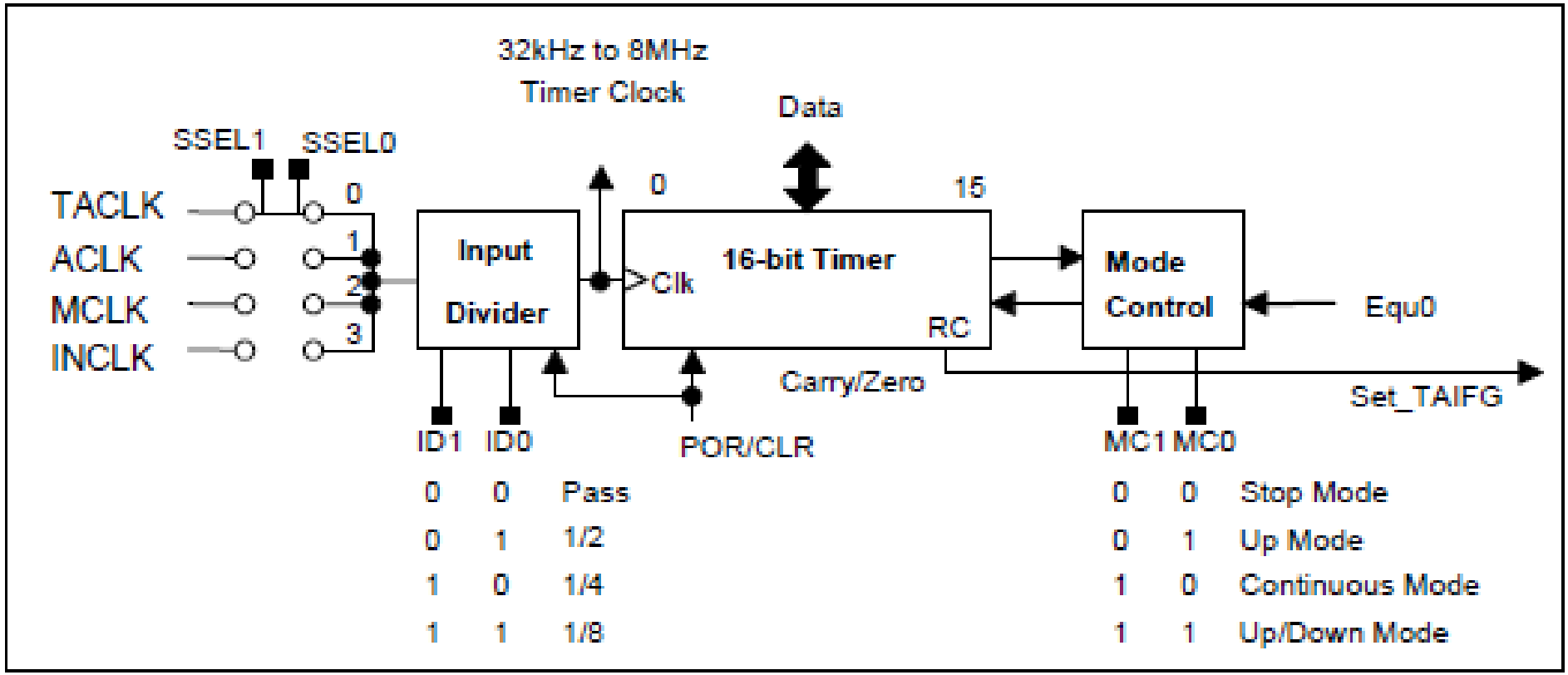
A **counter** is the fundamental hardware element found inside a timer.

The other essential element is a **clock** input. The counter is incremented each time a clock pulse is applied to its clock input. Therefore, a 16-bit timer will count from zero (0x0000) up to 64K (0xFFFF).

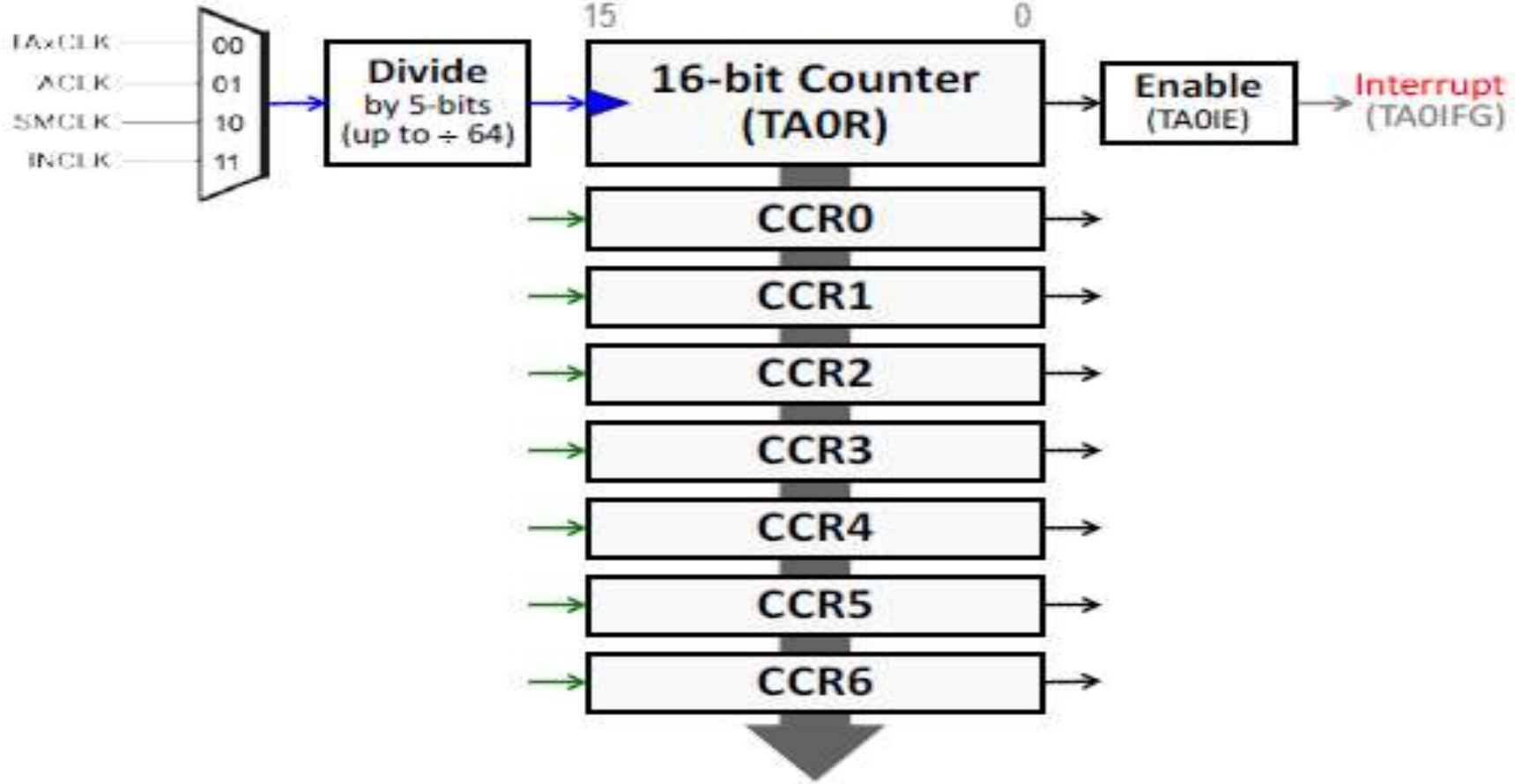
When the counter reaches it reaches its maximum value, it overflows – that is, it returns to zero and starts counting upward again. Most timer peripherals can generate an interrupt when this overflow event occurs; on TIMER_A, the interrupt flag bit for this event is called TAIFG (TIMER_A Interrupt Flag).



Timer_A


32kHz to 8MHz
Timer Clock
Data
SSEL1
SSEL0
TACLK
ACLK
MCLK
INCLK
0
1
2
3
Input
Divider
0
15
16-bit Timer
Clk
RC
Mode
Control
Equ0
Carry/Zero
Set_TAIFG
ID1
ID0
POR/CLR
MC1
MC0
0 0 Pass
0 1 1/2
1 0 1/4
1 1 1/8
0 0 Stop Mode
0 1 Up Mode
1 0 Continuous Mode
1 1 Up/Down Mode

Example: Timer0_A7



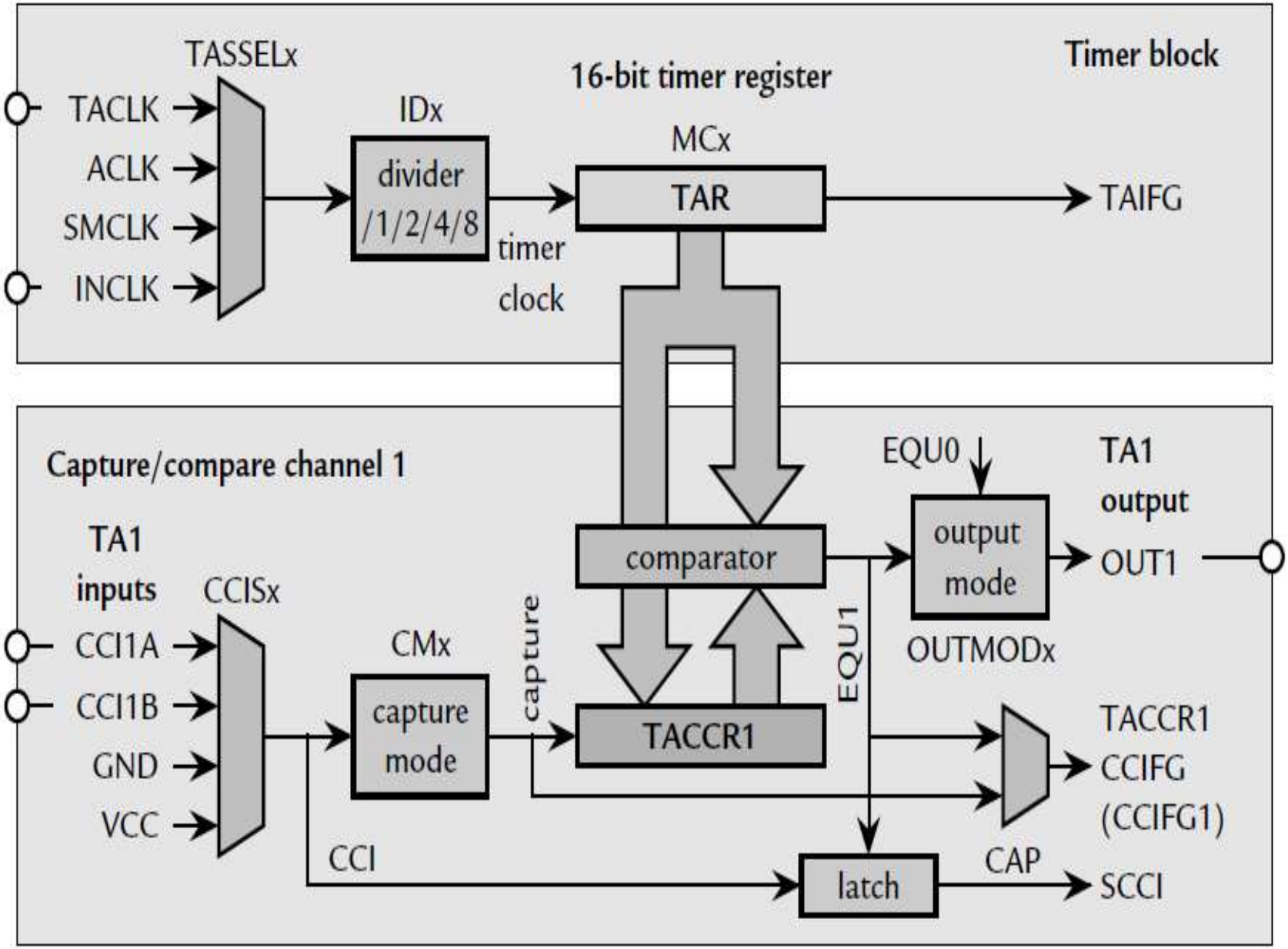


Figure 8.5: Simplified block diagram of Timer_A showing the timer block and

Timer_A Operation

- The Timer_A module is configured with user software
- ***16-Bit Timer Counter:***
- The 16-bit timer/counter register, TAR, increments or decrements (depending on mode of operation) with

  each rising edge of the clock signal.
- TAR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.
- TAR may be cleared by setting the TACLR bit. Setting TACLR also clears the clock divider and count direction for up/down mode.

- **Clock Source Select and Divider:**
- The timer clock TACLK can be sourced from ACLK, SMCLK, or externally via TACLK.
- The clock source is selected with the TASSELx bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the IDx bits .
- The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the IDEXx bits. The TACLK dividers are reset when TACLR is set.
- ***Starting the Timer***
- The timer may be started, or restarted in the following ways:

  · The timer counts when MCx > 0 and the clock source is active.

  · When the timer mode is either up or up/down, the timer may be stopped by writing 0 to TACCR0.

Timer Mode Control

The timer has four modes of operation as described in Table 12-1: stop, up, continuous, and up/down. The operating mode is selected with the MCx bits.

Table 12-1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TACCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TACCR0 and backdown to zero.

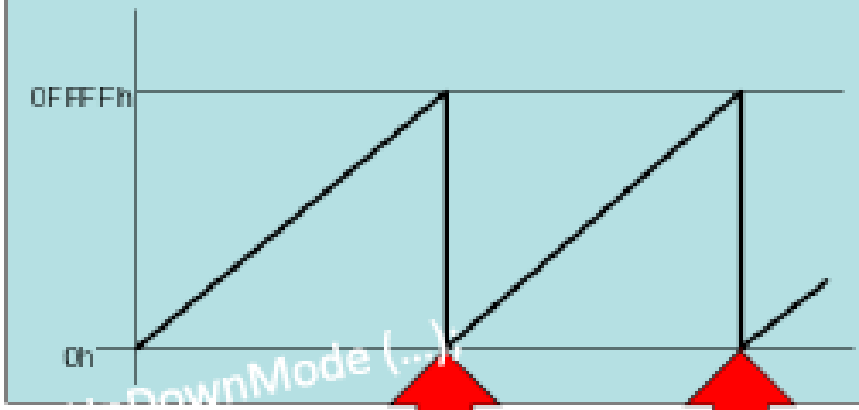
Timer Counting Modes Summary

Stop/Halt

Timer is halted

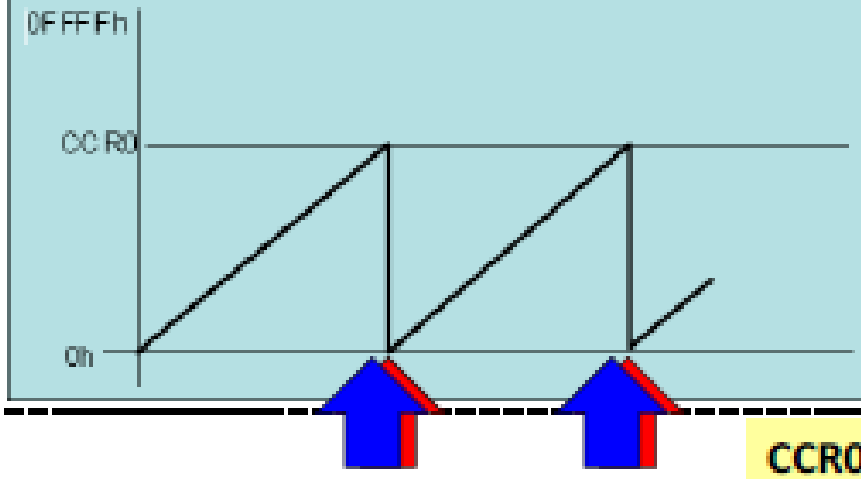
Continuous

Timer continuously counts up



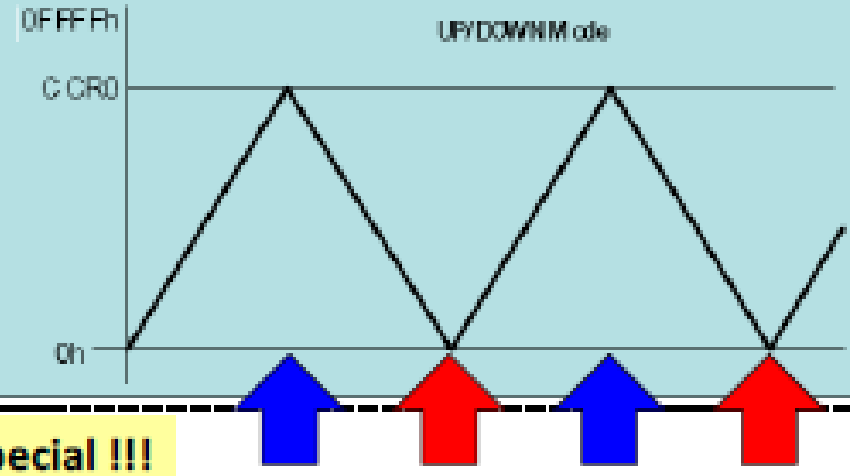
Up

Timer counts between 0 and CCR0



Up/Down

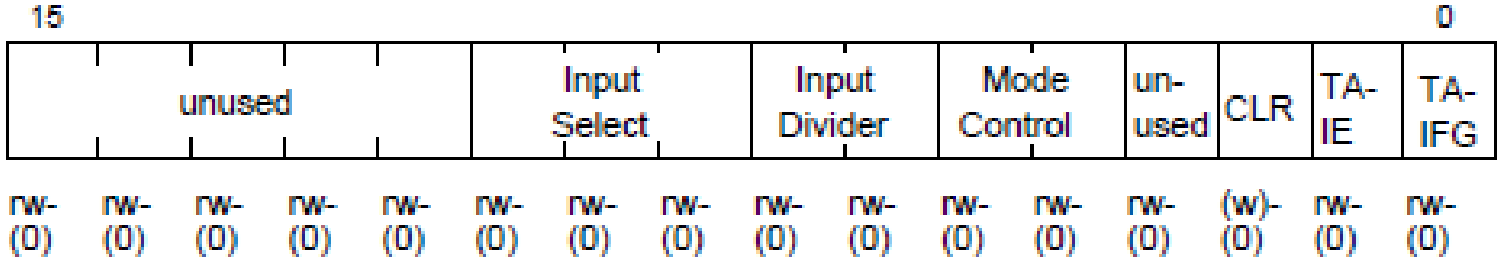
Timer counts between 0 and CCR0 and 0



CCR0 is special !!!

TACTL

160h



Bit 0: TAIFG: This flag indicates a timer overflow event.
UP mode: TAIFG is set if the timer counts from CCR0 value to 0000h.
Continuous mode: TAIFG is set if the timer counts from 0FFFFh to 0000h.
UP/DOWN mode: TAIFG is set if the timer counts down to 0000h.

Bit 1: Timer Overflow Interrupt Enable TAIE bit. An interrupt request from the timer overflow bit is enabled if set, and it is disabled if reset.

Bit 2: Timer Clear CLR bit. The timer and the input divider are reset after POR, or if bit CLR is set. The CLR bit is automatically reset by the hardware and always read as zero. The timer starts operation with the next valid input edge. The timer starts in an upward direction if it is not halted by cleared mode control bits.

Bit 3: Not used

Bit 4 to 5: Mode Control Description

MC1	MC0	Count Mode	Comment, Timer ...
0	0	Stop	is halted
0	1	Up to CCR0	counts up to CCR0 and restarts at 0
1	0	Cont. Up	counts up cont. all 65536 steps
1	1	Up/Down	counts up to CCR0, down to 0,.....

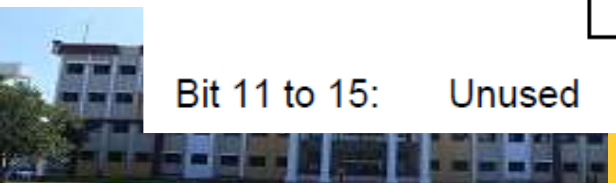
Bit 6 to 7: Input Divider control bits

ID1	ID0	Operation	Comment
0	0	Pass	Input signal is passed to the timer
0	1	/2	Input signal is divided by two
1	0	/4	Input signal is divided by four
1	1	/8	Input signal is divided by eight

Bit 8 to 10: Select source of timer input clock signal - preprocessed in the Input Divider

SSEL2	SSEL1	SSEL0	O/P signal	Comment
0	0	1	TACLK	The signal at dedicated ext. pin is used
0	0	1	ACLK	Auxillary clock ACLK is used
0	1	0	MCLK	System clock MCLK is used
0	1	1	INCLK	See device description
1	X	X	-----	Reserved

Bit 11 to 15: Unused



Capture/Compare Blocks

- Three or five identical capture/compare blocks, TACCRx, are present in Timer_A. Any of the blocks may be used to capture the timer data, or to generate time intervals.

Capture Mode:

- The capture mode is selected when CAP = 1. Capture mode is used to record time events.
- The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCISx bits.
- The CMx bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal.
- If a capture occurs:
  - The timer value is copied into the TACCRx register
  - The interrupt flag CCIFG is set

Compare Mode

- The compare mode is selected when CAP = 0. The compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TAR *counts to the value in a TACCRx:*
  - · Interrupt flag CCIFG is set
  - · Internal signal EQUx = 1
  - · EQUx affects the output according to the output mode

Output Unit

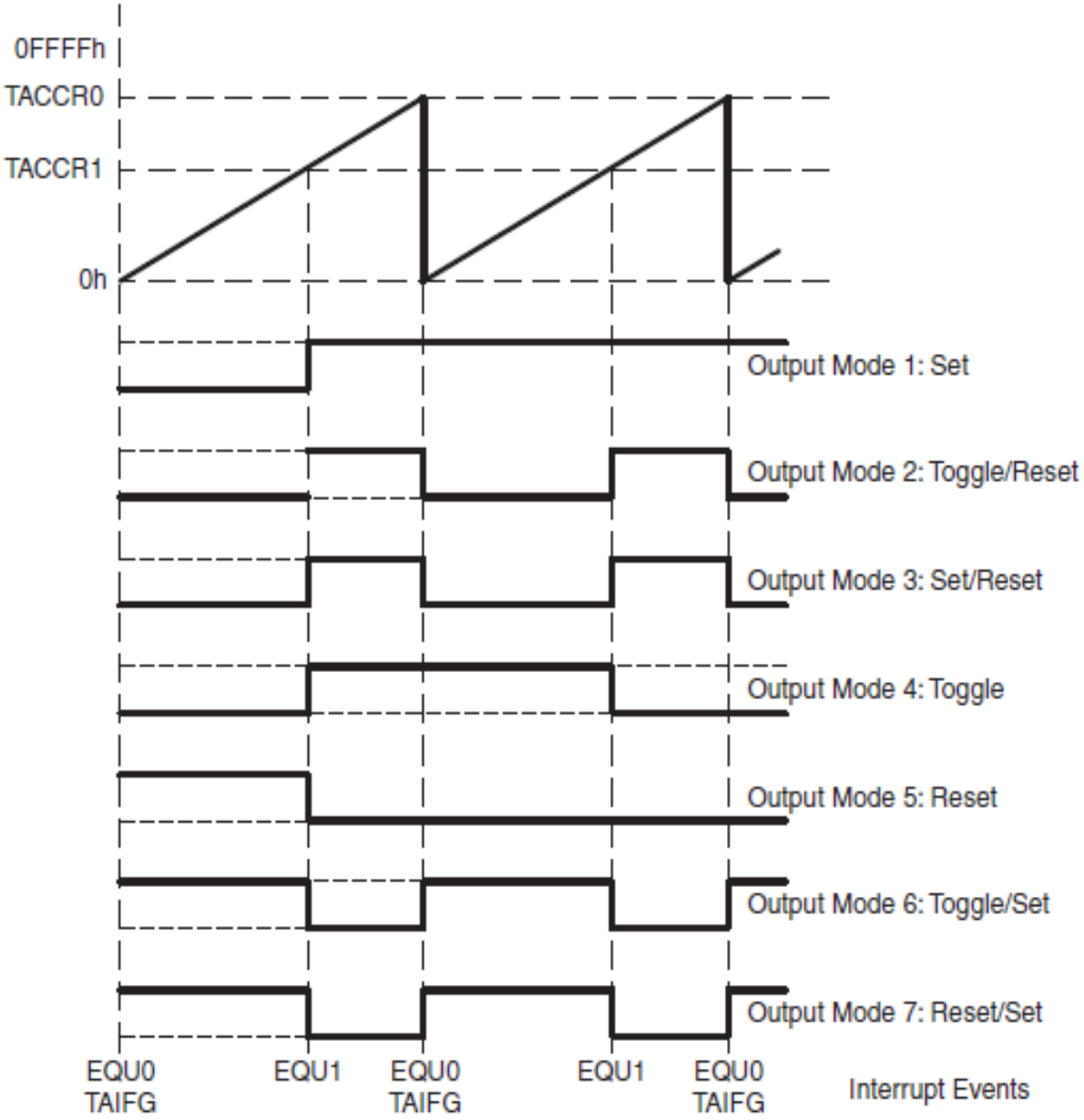
- Each capture/compare block contains an output unit. The output unit is used to generate output signals such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUx signals.

Output Modes

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated.
001	Set	The output is set when the timer *counts* to the TACCRx value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer *counts* to the TACCRx value. It is reset when the timer *counts* to the TACCR0 value.
011	Set/Reset	The output is set when the timer *counts* to the TACCRx value. It is reset when the timer *counts* to the TACCR0 value.
100	Toggle	The output is toggled when the timer *counts* to the TACCRx value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TACCRx value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer *counts* to the TACCRx value. It is set when the timer *counts* to the TACCR0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TACCRx value. It is set when the timer *counts* to the TACCR0 value.

Output Example—Timer in Up Mode

The OUTx signal is changed when the timer *counts* up to the TACCRx value, and rolls from TACCR0 to zero, depending on the output mode. An example is shown in Figure 12-12 using TACCR0 and TACCR1.



Real-Time Clock

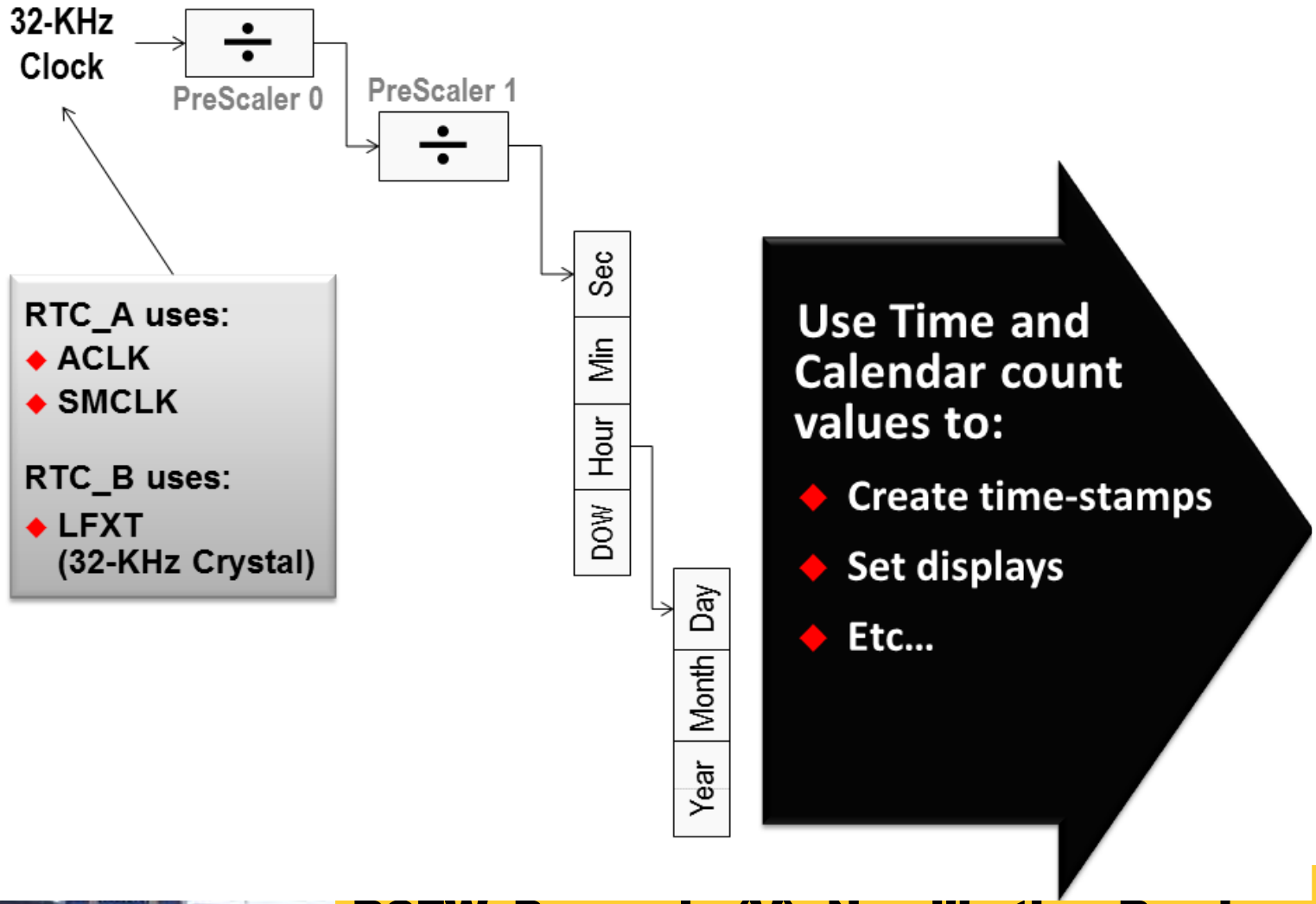
- The Real-Time Clock (RTC) peripheral is a sophisticated timer that keeps track of Calendar, Month, and Time information. It operates in Binary or BCD modes; whichever is most useful for many applications.

- It counts seconds, minutes, hours, days, months, and years. Alternatively it can be used as a straightforward counter.

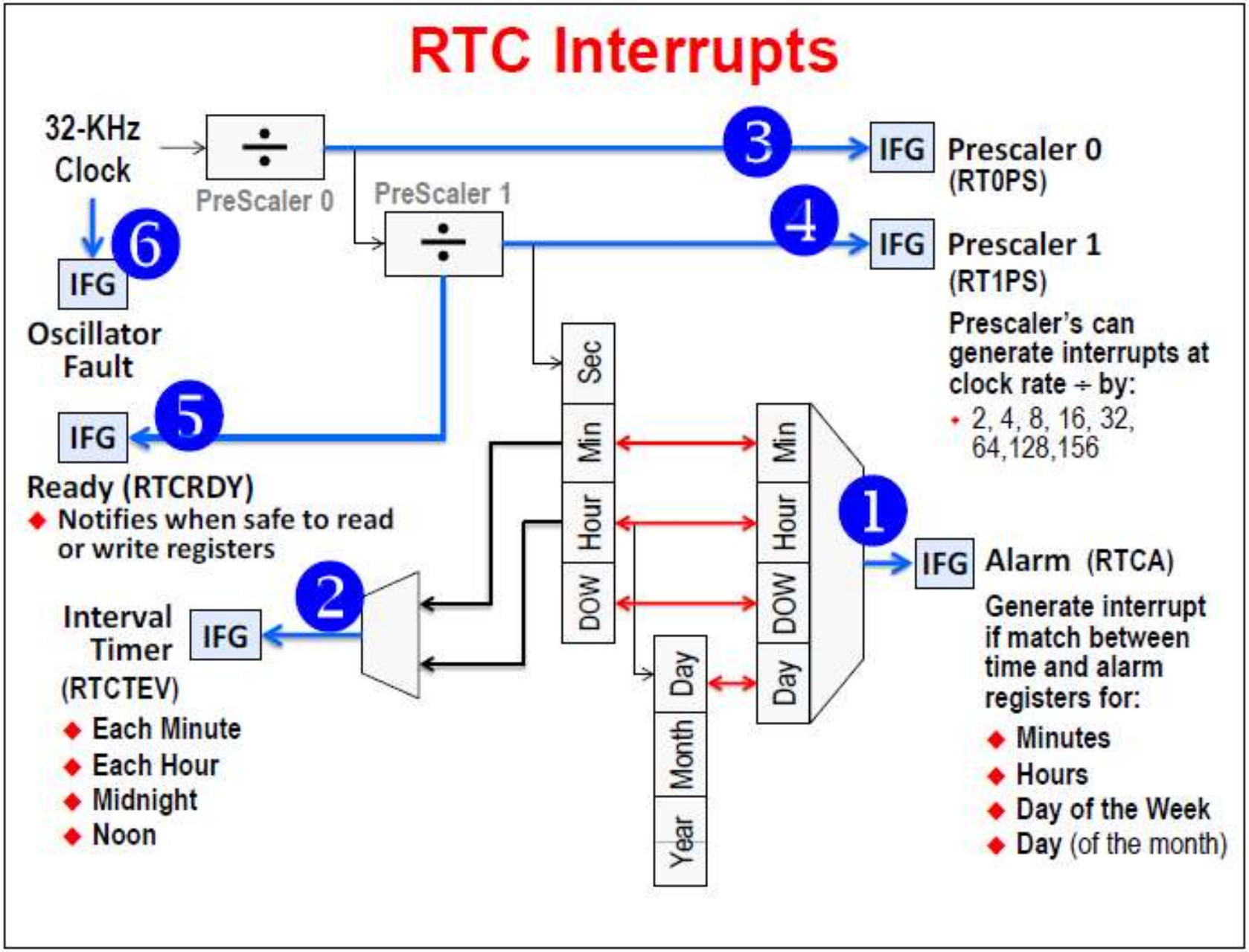
- This can be configured in calendar mode or Counter mode.

- The RTC affords the ability to set Calendar/Time based Alarms (i.e. Interrupts).

RTC Block Diagram



RTC Interrupts



RTCCTL1, Real-Time Clock Control Register 1

7	6	5	4	3	2	1	0
RTCBCD	RTCHOLD	RTCMODE	RTCRDY	RTCSSEL		RTCTEV	
rw-(0)	rw-(1)	rw-(0)	r-(0)	rw-0	rw-0	rw-(0)	rw-(0)

RTCBCD Bit 7

Real-time clock BCD select. Selects BCD counting for real-time clock. Applies to calendar mode (RTCMODE = 1) only - setting will be ignored in counter mode. Changing this bit will clear seconds, minutes, hours, day of week, and year are to 0 and sets day of month and month to 1. The real-time clock registers need to be set by software afterwards.

0 Binary/hexadecimal code selected

1 BCD (Binary Coded Decimal) code selected

RTCHOLD Bit 6

Real-time clock hold

0 Real-Time Clock (32-bit counter or calendar mode) is operational

1 In counter mode (RTCMODE = 0) only the 32-bit counter is stopped. In calendar mode (RTCMODE = 1) the calendar is stopped as well as the Prescale counters, RT0PS and RT1PS. RT0PSHOLD and RT1PSHOLD are do not care.

RTCMODE Bit 5

Real-time clock mode

0 32-bit counter mode

1 Calendar modeSwitching between counter and calendar mode will reset the real-time clock/counter registers. Switching to calendar mode clears seconds, minutes, hours, day of week, and year are to 0 and sets day of month and month to 1. The real-time clock registers need to be set by software afterwards. The Basic Timer counters, BT0CNT and BT1CNT, are also cleared.

RTCRDY Bit 4

Real-time clock ready

0 RTC time values in transition (calendar mode only).

1 RTC time values safe for reading (calendar mode only)This bit indicates when the RTC time values are safe for reading (calendar mode only). In counter mode, RTCRDY signal remains cleared.

RTCSSEL Bits 3-2

Real-time clock source select. Selects clock input source to the RTC/32-bit counter. In Real-Time Clock calendar mode, these bits are do not care. The clock input is automatically set to the output of RT1PS.

00 ACLK

01 SMCLK

10 Output from RT1PS

11 Output from RT1PS

RTCTEV Bits 1-0

Real-time clock time event

RTC Mode	RTCTEVx	Interrupt Interval
Counter Mode (RTCMODE = 0)	00	8-bit overflow
	01	16-bit overflow
	10	24-bit overflow
	11	32-bit overflow
Calendar Mode (RTCMODE = 1)	00	Minute changed
	01	Hour changed
	10	Every day at midnight (00:00)
	11	Every day at noon (12:00)

- The current time and date held in registers.
  - RTCYEARH - Century
  - RTCYEARL - Year
  - RTCMON - Month
  - RTCDAY - Day of Month
  - RTCDOW - Day of Week - (0- 6)
  - RTCHOUR - Hours - 0 : 24
  - RTCMIN - Minutes - 0 : 59
  - RTCSEC - Seconds - 0 : 59

- The registers sometimes will be as word length for accessing the “year”. – RTCYEAR
- These values can be stored either as normal binary numbers or as BCD.

- MSP stands for “mixed signal processor” and most MSP430s are used in applications that handle both analog and digital data
- The MSP430 offers three methods of conversion with quite different characteristics.

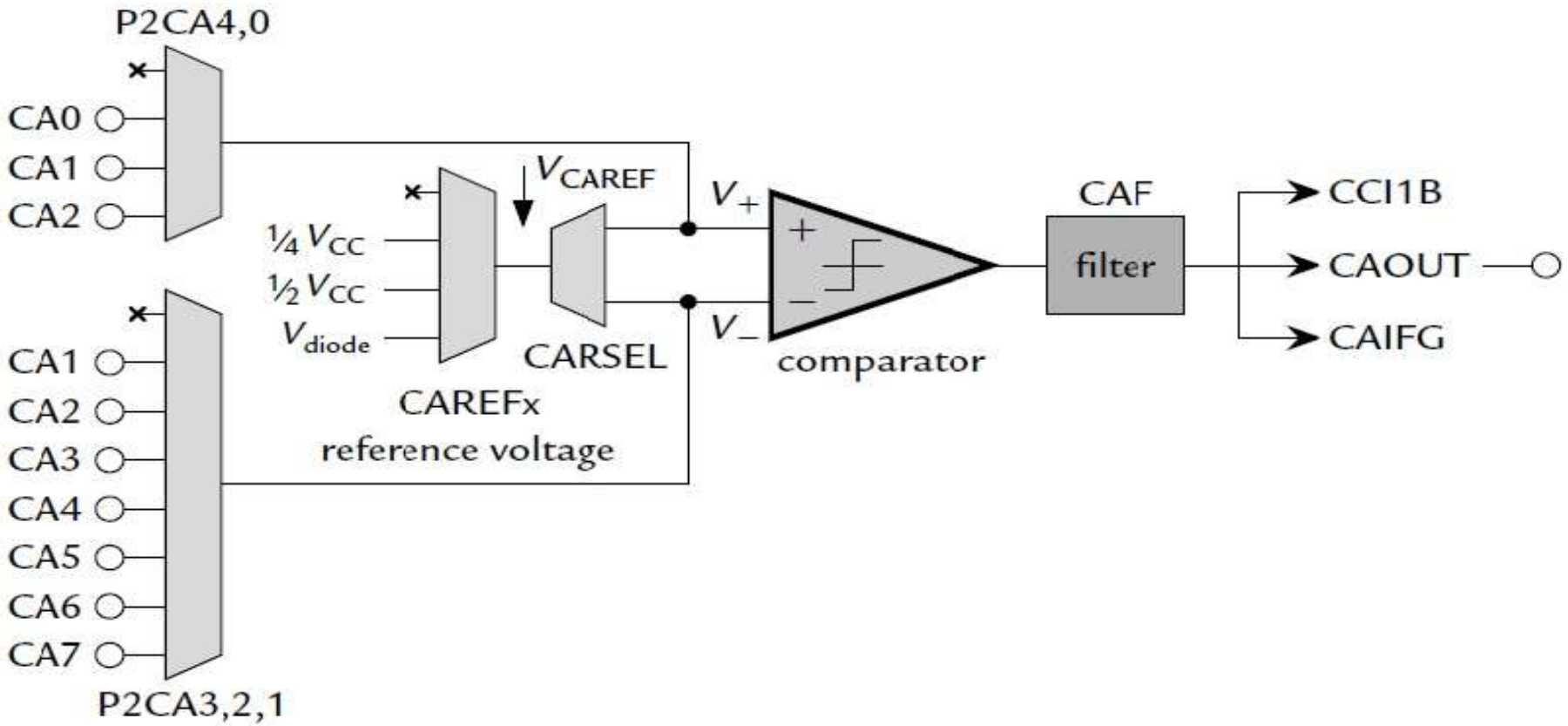
  Comparator: Simple and cheap module that cannot perform a conversion by itself but is usually used with Timer_A to measure the time-constant of an external *RC circuit.* There are two versions, Comparator_A and Comparator_A+.

  Successive-approximation ADC: The general-purpose type of ADC for many years. It is fast and relatively straightforward to understand. There are two versions, ADC10 and ADC12, which give 10 and 12 bits of output.
- **Sigma–delta ADC:**A more complicated ADC that works in a quite different way to give higher resolution (more bits) but at a slower speed. There are two versions, SD16 and SD16_A, both of which give a 16-bit output.

COMPARATOR A/A+

- An analog comparator compares the voltages on its two input terminals, *V+ and V−. Its* output is high if *V+ >V− and low if V+ <V−. Thus it provides a basic bridge between* the analog and digital domains and acts like a 1-bit ADC.

Architecture of Comparator A+



- The entire module is switched on and off with the CAON bit. It is off by default to save current.
- The non inverting input *V+ can be connected to external signals CA0–CA2 or left* without an external connection. This is selected using bits P2CA4 and P2CA0. An internal voltage reference can also be connected to *V+.*
- Similarly, the inverting input *V– can be connected to external signals CA1–CA7* (but not CA0) or left unconnected, according to bits P2CA[3:1]. It can also be connected to an internal reference.
- The internal reference voltage *VCAREF can be chosen from ¼ VCC, ½ VCC or a* nominally fixed voltage from a transistor, *Vdiode. This is selected with the CAREFx* bits. It can be applied to either input of the comparator according to the CARSEL bit.

- The raw output of the comparator can optionally be filtered through an *RC circuit* to reduce oscillations in the signal, which may arise if the inputs vary slowly. This is selected with the CAF bit.
- The output is brought to an external pin CAOUT. It is also connected internally to capture input CCI1B of Timer_A, which allows precise timing without delays that would be introduced if software were needed for communication between the modules.
- The flag CAIFG is raised on either a rising or falling edge of the comparator output, selected with the CAIES bit. This can in turn request an interrupt if CAIE is set.
- Comparator_A+ has its own interrupt vector and the flag is cleared automatically when the interrupt is serviced.

The ADC10 Successive-Approximation ADC

- There are two SAR ADC modules available on the MSP430, the ADC10 and ADC12.
- The ADC12 has a more elaborate set of registers for storing batches of results. It needs external capacitors on its voltage reference but the ADC10 does not.

Architecture of the ADC10

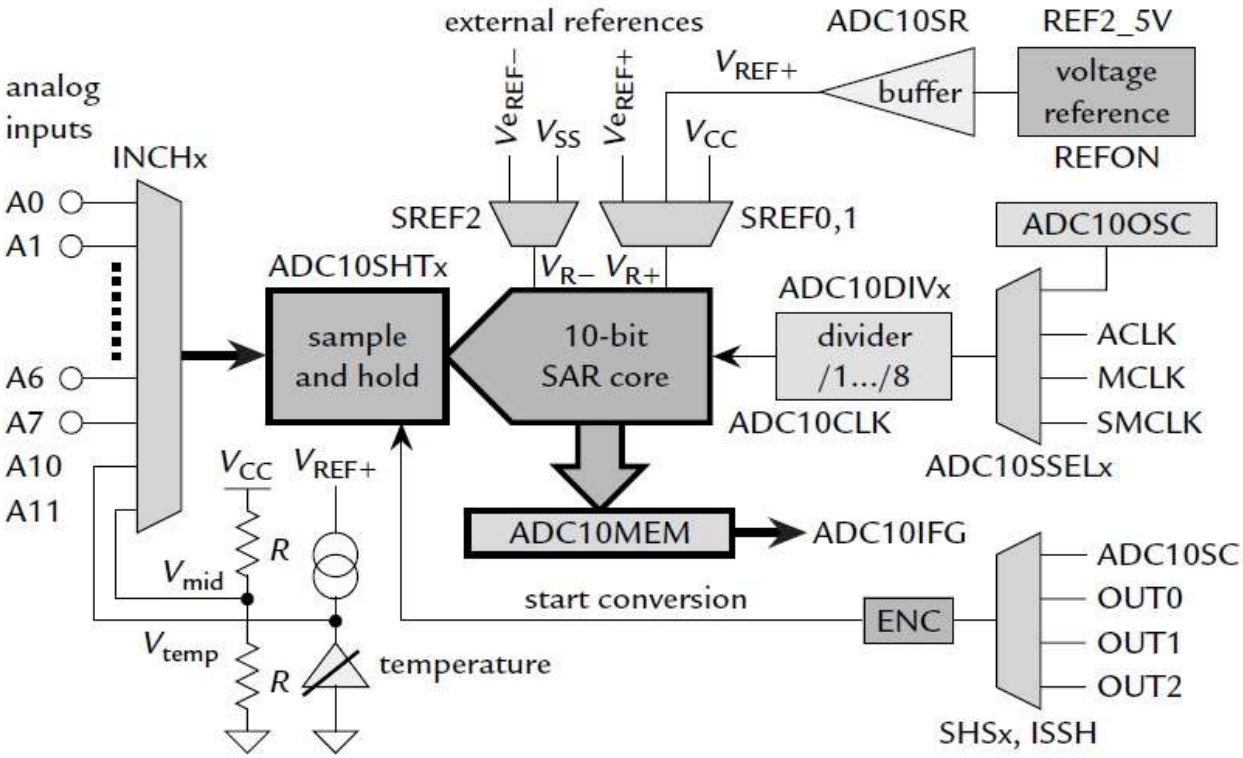


Figure 9.16: Simplified block diagram of the ADC10. The connections for externa references, automatic sequences of conversions, and the data transfer controller a omitted for clarity.

- Figure shows a simplified block diagram of the ADC10

Core:

- The heart of the ADC10 is a 10-bit, switched-capacitor, SAR core.
- It is guaranteed monotonic with no missing codes.
- The ADC10ON bit enables the core and a flag
- ADC10BUSY is set while sampling and conversion is in progress.
- The result is written to ADC10MEM

Clock:

- This can be taken from MCLK, SMCLK, ACLK, or the module's internal oscillator ADC10OSC, selected with the ADC10SSELx bits.
- The internal oscillator runs nominally at 5MHz
- It is automatically enabled when needed and disabled when conversions have finished. This makes it the most convenient source for most applications.
- The frequency of the clock can be divided by 2*, 3, . . . , 7, 8 by configuring the* ADC10DIVx bits.
- The output of the divider is labeled ADC10CLK and feeds both the SAR core and sample-and-hold blocks.

Sample and Hold:

- a **sample and hold** (**S/H**, also "follow-and-hold") circuit is an analog device that samples (captures, grabs) the voltage of a continuously varying analog signal and holds (locks, freezes) its value at a constant level for a specified minimum period of time.

Input Selection

- A multiplexer selects the input from eight external pins A0–A7 (more in larger MSP430s) and four internal connections.
- Two of the internal connections are for optional, The other two internal connections are A10 to a temperature sensor and A11 to *Vmid = ½ (VCC+VSS), which is provided to monitor the supply voltage.*

Conversion Trigger

- A conversion can be triggered in two ways provided that the ENC bit is set.
- The first is by setting the ADC10SC bit from software (it clears again automatically).
- The second OUTn has a rising edge when you want to start a new conversion.

Interrupts

- The interrupt flag ADC10IFG is Set when the result is written to ADC10MEM

Pulse Width Modulation (PWM) Control

- A Pulse Width Modulation (PWM) Signal is a method for generating an analog signal using a digital source.

- A PWM signal consists of two main components that define its behavior: **a duty cycle** and **a frequency**.

- The **duty cycle** describes the amount of time, the signal is in a high (on) state as a percentage of the total time of it takes to complete one cycle.

- The frequency determines how fast the PWM completes a cycle (i.e. 1000 Hz would be 1000 cycles per second)

PWM Control Cont..

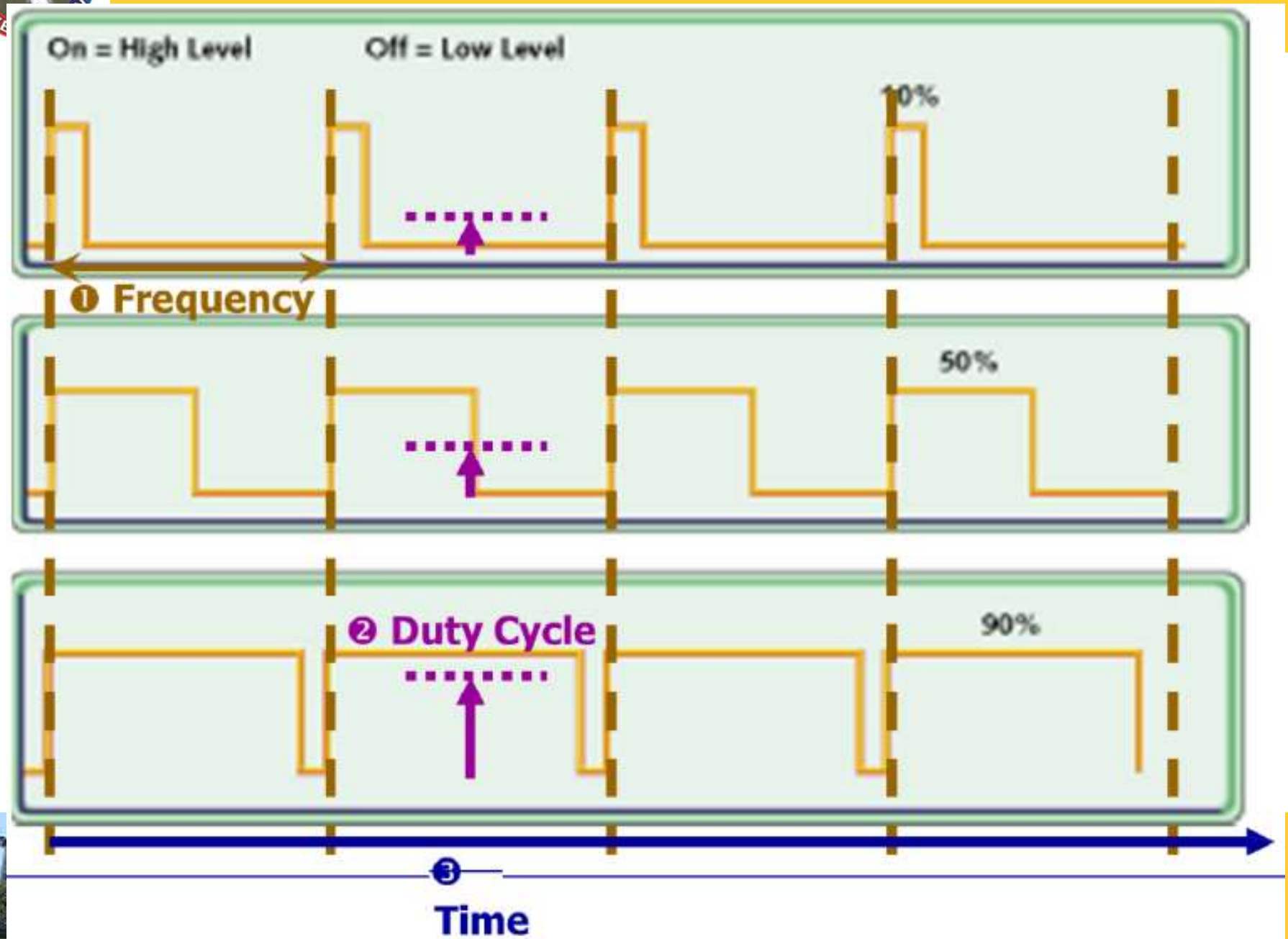
- The duration of the period where the signal is high is known as "**Duty Cycle**".

Uses of PWM Control:

- Pulse Width Modulation is used to control analog circuits with digital outputs.

- PWM signals are used for a wide variety of control applications. Their main use is for controlling DC motors, Servo motors, LEDs and also to control valves, pumps, hydraulics, and other mechanical parts.

PWM Frequency/Duty Cycle Cont..



Data Transfer Using Direct Memory Access (DMA):

- The DMA controller transfers data from memory to peripheral and vice versa. For example, the DMA controller can move data from the ADC conversion memory to RAM.

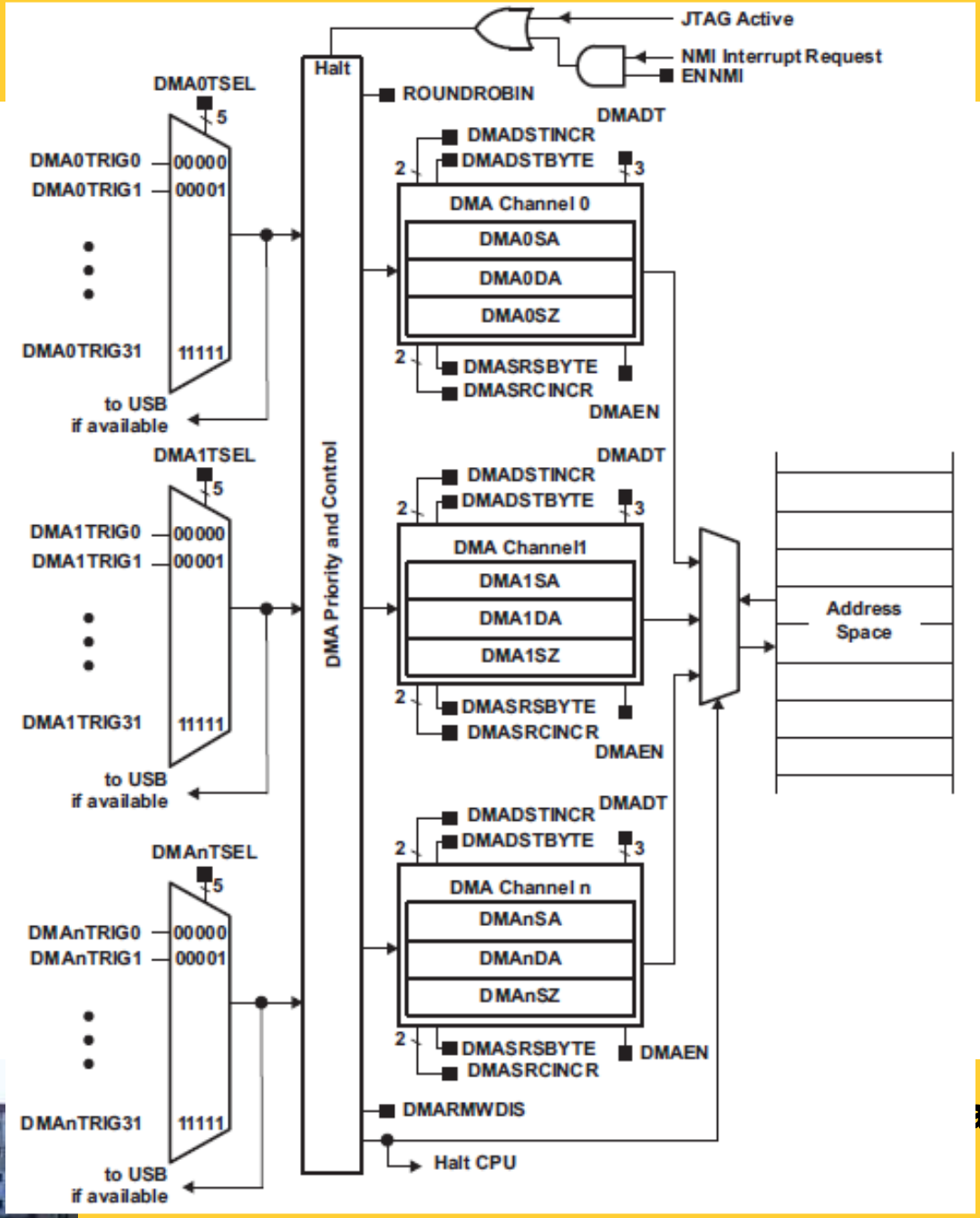
- Using the DMA controller can increase the throughput of peripheral modules.

- It can also reduce system power consumption by allowing the CPU to remain in a low-power mode, without having to awaken to move data to or from a peripheral.

DMA Controller Features :

* Up to eight independent transfer channels
* Configurable DMA channel priorities
* Requires only two MCLK clock cycles per transfer
* Byte or word and mixed byte and word transfer capability
* Block sizes up to 65535 bytes or words
* Configurable transfer trigger selections
* Selectable-edge or level-triggered transfer
* Four addressing modes
* Single, block, or burst-block transfer modes

DMA Controller Block Diagram

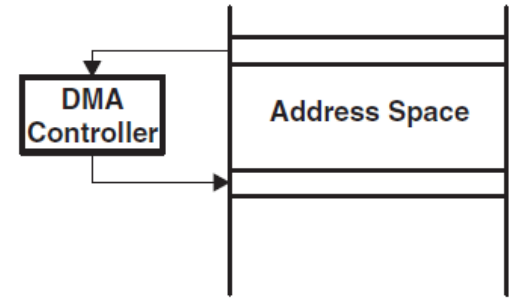


DMA Addressing Modes:

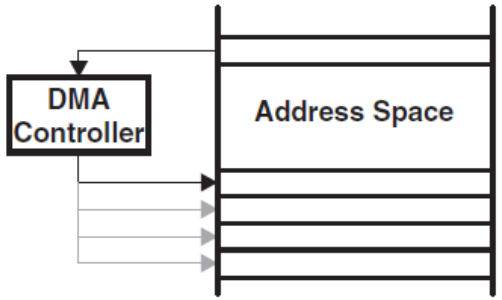
The DMA controller has four addressing modes. The addressing mode for each DMA channel is independently configurable. For example, channel 0 may transfer between two fixed addresses, while channel 1 transfers between two blocks of addresses.

The addressing modes are:

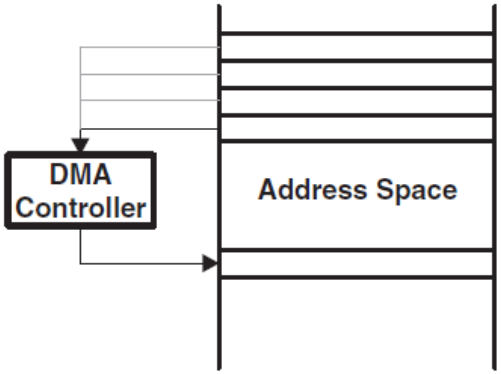
- Fixed address to fixed address
- Fixed address to block of addresses
- Block of addresses to fixed address
- Block of addresses to block of addresses



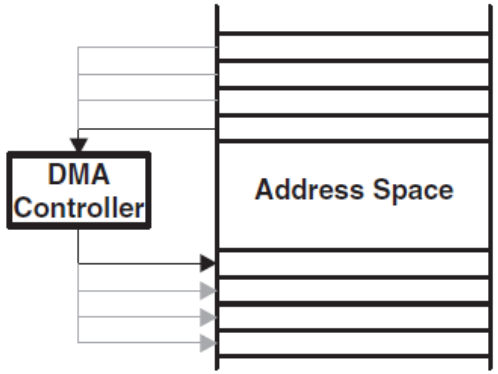
Fixed Address To Fixed Address



Fixed Address To Block Of Addresses



Block Of Addresses To Fixed Address



Block Of Addresses To Block Of Addresses

DMA Transfer Modes:

The DMA controller has six transfer modes selected by the DMADT bits

Table: DMA Transfer Modes

DMADT	Transfer Mode	Description
000	Single transfer	Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.
001	Block transfer	A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.
010, 011	Burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.
100	Repeated single transfer	Each transfer requires a trigger. DMAEN
101	Repeated block transfer	A complete block is transferred with one trigger.
110, 111	Repeated burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN remains enabled.