Lecture 4: Programming with Interrupts

Jan Vitek

ECE568/CS590/ECE495/CS490
Spring 2011

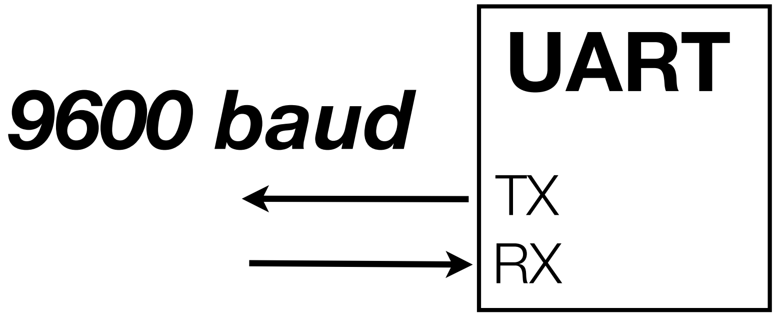
Reading List

- Mandatory Reading
  - Chapter 4 of ESP textbook

- Optional Reading
  - N/A

Event processing with interrupts

- *Consider a program that sends 9600 characters per second on a universal asynchronous receiver/transmitter*



- With polling: 100% CPU load

```
for(;;) {//Polling UART Receive
 while (!(IFG2&URXIFG0));
 TXBUF0 = RXBUF0;
}
```

- With interrupts: 0.1% CPU load

```
//UART Receive Interrupt
#pragma vector=UART_VECTOR
__interrupt void rx() {
 TXBUF0 = RXBUF0;
}
```

Interrupts

- Embedded system must respond to external events in a timely fashion
  - *Example: data received, timeout*

- Interrupts provide a way to react to happenings flagged by the hardware
  - *Interrupts are often used as building blocks for higher-level abstractions*
  - *Interrupts obviate the need for polling*

- Programming with interrupts is tricky and error prone
  - *They introduce most of the dangers of concurrent programming in a sequential context*

Interrupt lifecycle

- The processor detects a signal on Interrupt Request (IRQ) pin.
  - Typically multiple pins attached to hardware components such as serial ports and network interfaces
- Save context.
  - The processor stops what it was doing and saves enough information to be able to return to the task at hand after the interrupt has been handled
- Locate and jump to an Interrupt Service Routine (ISR).
  - There can be multiple ISRs and multiple pending interrupts. The processor will select the interrupt with the highest priority, and identify the ISR corresponding to the IRQ
- Restore context.
  - Upon a RETURN from an ISR, the processor will recover the saved context and resume execution

Contexts

- When an interrupt is detected and the corresponding ISR is executed, the state of the processor will be changed as a side effect of executing the ISR
- In particular, the program counter, the stack pointer, and all the other registers can possibly be modified
- *Saving the context:*
  - the process of pushing original values of registers on the stack before modifying them
- *Restoring the context:*
  - the process of popping values from the stack into registers to restore the state of the system

Disabling Interrupts

- Most microprocessors support disabling *all* interrupts in one atomic step as well as disabling selected interrupt signals

- *nonmaskable interrupt:*
  - an interrupt pin which can not be disabled

- Some microprocessor support disabling interrupt priority ranges

Sharing Data

- ISRs often must communicate with the rest of the system
- This is achieved by sharing mutable memory location between the ISR and the rest of the system
- Such sharing can endanger consistency of the data if proper care is not taken when manipulating it
  - Consider the following example:
    - What is the invariant?
    - How can it be broken?

```
static int T[2];

void interrupt i(){
    T[0] = …
    T[1] = -T[0];
}
```

```
void main() { int i,j;
   while(1){
     i=T[0]; j=T[1];
     if(i+j) ERROR();
   }
}
```

Sharing Data

- Is this a fix?

```
static int T[2];

void interrupt i(){
    T[0] = …
    T[1] = -T[0];
}
```

```
void main() {
 while(1)
   if(T[0]+T[1])
    ERROR();
}
```

Critical sections

- A critical section is a sequence of code that must appear to execute atomically
- It may be preempted if there is no way for the program to observe that it was preempted

```
static int secs,mins,hrs;
void interrupt time(){
  if(++secs>=60) {
    secs=0;
    if(++mins>=60) {
      mins=0;
      if(++hrs>=24) hrs=0;
} } }
```

```
long secFromMidnight() {
 return((hrs*60)+mins)*60+secs;
}
```

Critical sections

- A correct solution must preserve interrupt state

```
long secFromMidnight() {
  long retVal;
  unsigned state=__disable_interrupt();
  retVal =((hrs*60)+mins)*60+secs;
  if(state) __enable_interrupt();
  return retVal;
}
```

volatile

- Compilers try to generate efficient code, for this they recognize certain patterns and replace them with more equivalent (under certain assumptions) code

```
static int secs,mins,hrs;
void interrupt time(){
  if(++secs>=60) {
    secs=0;
    if(++mins>=60) {
      mins=0;
      if(++hrs>=24) hrs=0;
} } }
```

```
long notZero() {
 int retVal=secs;
 while(!retVal)retVal=secs;
 return retVal;
}
```

volatile

- Are the following two programs equivalent?

```
x=1;
x=1;
```

```
x=1;
```

`volatile`

- Are the following two programs equivalent?

```
x=y;
x=y;
```

```
x=y;
```

volatile

- Are the following two programs equivalent?

```
long notZero() {
 int retVal=secs;
 while(!retVal)retVal=secs;
 return retVal;
}
```

```
long notZero() {
 int retVal=secs;
 while(!retVal);
 return retVal;
}
```

Interrupt Latency

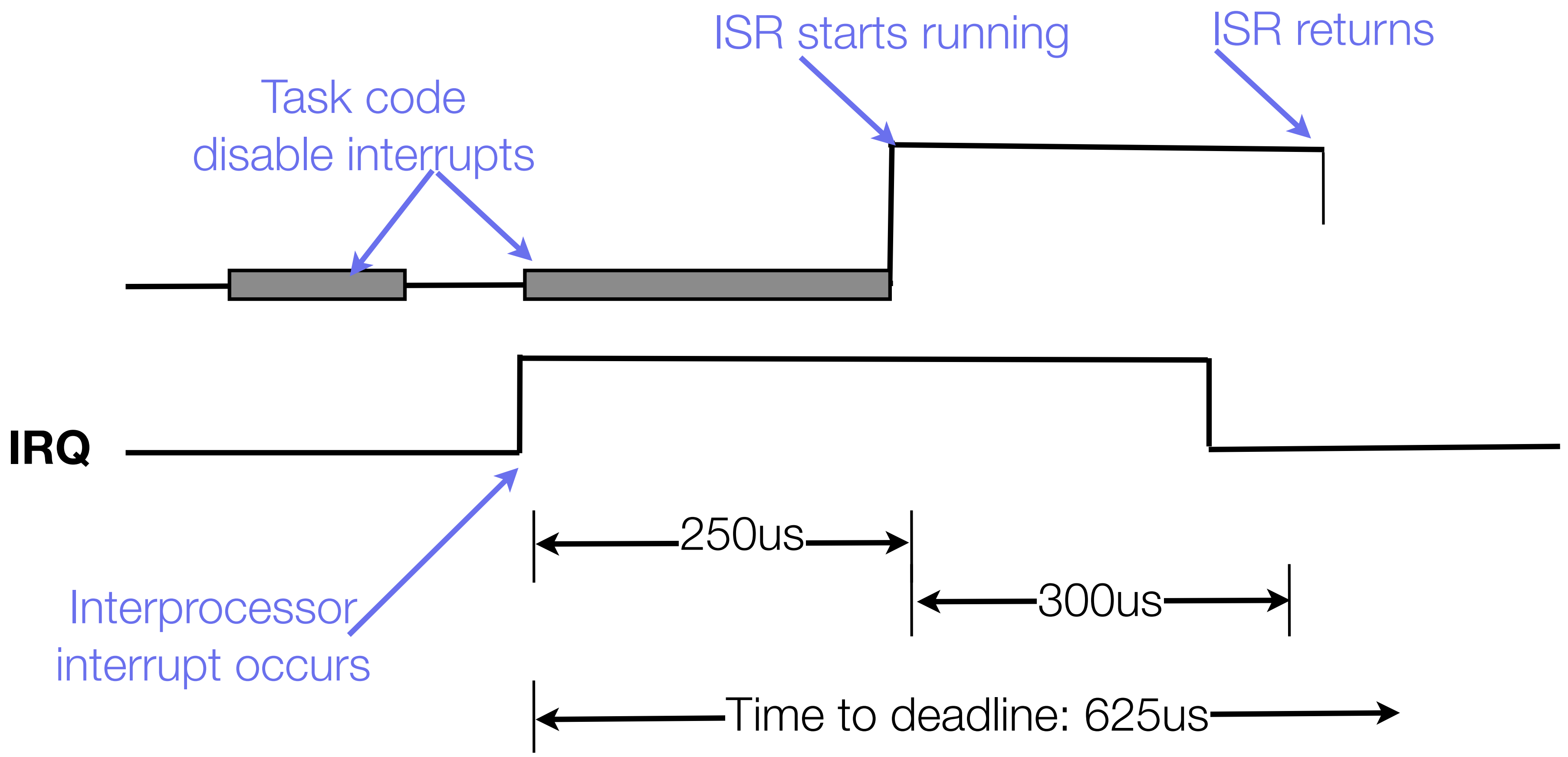
- The latency of an interrupt is measured as the time between an event E being signaled and the corresponding ISR returning

- Interrupt latency is a function of:

1. the longest time interrupts can be disabled
2. the time it takes to execute all ISRs of higher priority than E
3. the time it takes stop executing, save the context, and start executing an ISR
4. the time it take to execute the ISR corresponding to E

Interrupt Latency

- Consider a task that must:
  - disable interrupts for 125us to read temperature
  - disable interrupts for 250us to update the time
  - respond to an interrupt within 625us
  - the ISR takes 300us



MSP430

- A quick overview of some features of the MSP430 will help making some of the discussion more concrete

MSP430 Memory organization

- The general layout of the address space:
  - 0x0000–0x0007
    - Processor special function registers (interrupt control regs)
  - 0x0008–0x00FF
    - 8-bit peripherals. Accessed using 8-bit loads and stores.
  - 0x0100–0x01FF
    - 16-bit peripherals. Acessed using 16-bit loads and stores.
  - 0x0200–0x09FF
    - Up to 2048 bytes of RAM.
  - 0x0C00–0x0FFF
    - 1024 bytes of bootstrap loader ROM
  - 0x1000–0x10FF
    - 256 bytes of data flash ROM
  - 0x1100–0x38FF
    - Extended RAM on models with more than 2048 bytes of RAM
  - 0x1100–0xFFFF
    - Up to 60 kilobytes of ROM. Smaller ROMs start at higher addresses. ***The last 16 or 32 bytes are interrupt vectors.***

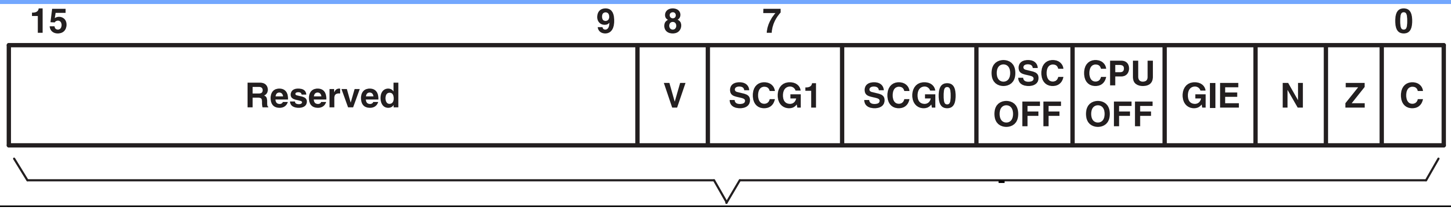
		MSP430F5438 **MSP430F5437**
Memory (flash) Main: interrupt vector Main: code memory	Total Size Flash Flash	256 KB 00FFFFh–00FF80h 045BFFh–005C00h
Main: code memory	Bank 3	64 KB 03FFFFh–030000h
	Bank 2	64 KB 02FFFFh–020000h
	Bank 1	64 KB 01FFFFh–010000h
	Bank 0	64 KB 045BFFh–040000h 00FFFFh–005C00h
RAM	Size	16 KB
	Sector 3	4 KB 005BFFh–004C00h
	Sector 2	4 KB 004BFFh–003C00h
	Sector 1	4 KB 003BFFh–002C00h
	Sector 0	4 KB 002BFFh–001C00h
Information memory (flash)	Info A	128 B 0019FFh–001980h
	Info B	128 B 00197Fh–001900h
	Info C	128 B 0018FFh–001880h
	Info D	128 B 00187Fh–001800h
Bootstrap loader (BSL)(1) memory (flash)	BSL 3	512 B 0017FFh–001600h
	BSL 2	512 B 0015FFh–001400h
	BSL 1	512 B 0013FFh–001200h
	BSL 0	512 B 0011FFh–001000h
Peripherals	Size	4KB 000FFFh–000000h

MSP430 Memory organization

- 16-bit RISC CPU
- Single-cycle register file
  - 4 special purpose registers
    - R0 program counter
    - R1 stack pointer
    - R2 status register
    - R3 constant generator (-1,0,1,2,4,8)
  - 12 general purpose registers

- R4-R10 Expression register	*Callee saved*
- R11 Expression register	Caller saved
- R12 Expression register, argument pointer, return register	Caller saved
- R13 Expression register, argument pointer, return register	Caller saved
- R14 Expression register, argument pointer	Caller saved
- R15 Expression register, argument pointer,	Caller saved

Status Register



Bit	Description	
Reserved	Reserved	
V	Overflow. This bit is set when the result of an arithmetic operation overflows the signed-variable range.	
	`ADD(.B), ADDX(.B,.A), ADDC(.B), ADDCX(.B.A), ADDA`	Set when: positive + positive = negative negative + negative = positive otherwise reset
	`SUB(.B), SUBX(.B,.A), SUBC(.B),SUBCX(.B,.A), SUBA, CMP(.B), CMPX(.B,.A), CMPA`	Set when: positive – negative = negative negative – positive = positive otherwise reset
SCG1	System clock generator 1. This bit, when set, turns off the DCO dc generator if DCOCLK is not used for MCLK or SMCLK.	
SCG0	System clock generator 0. This bit, when set, turns off the FLL+ loop control.	
OSCOFF	Oscillator off. This bit, when set, turns off the LFXT1 crystal oscillator when LFXT1CLK is not used for MCLK or SMCLK.	
CPUOFF	CPU off. This bit, when set, turns off the CPU.	
GIE	General interrupt enable. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.	
N	Negative. This bit is set when the result of an operation is negative and cleared when the result is positive.	
Z	Zero. This bit is set when the result of an operation is 0 and cleared when the result is not 0.	
C	Carry. This bit is set when the result of an operation produced a carry and cleared when no carry occurred.	

Calling conventions

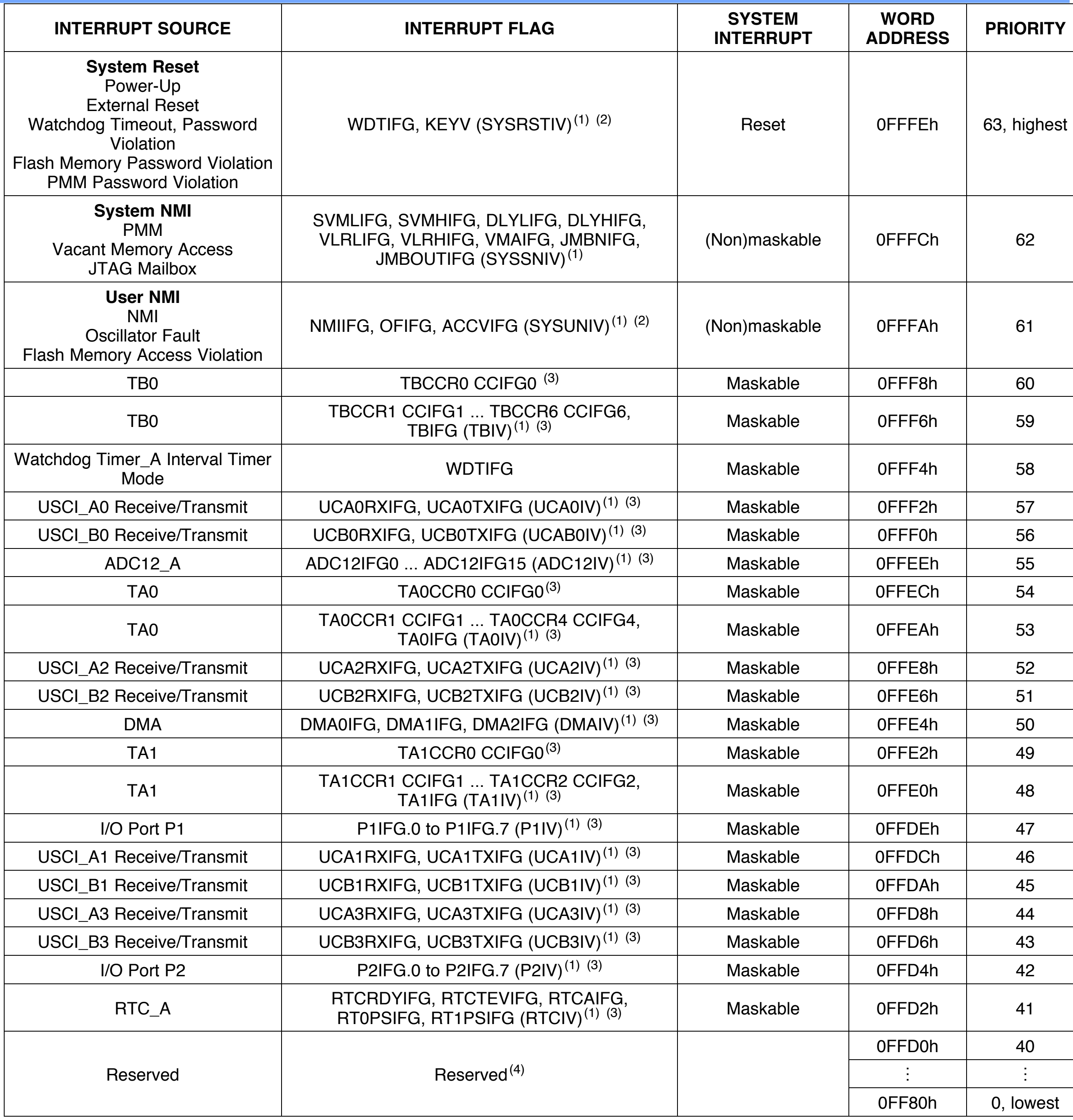
```
func:                   ; Called function entry point
   PUSH.W r10
   PUSH.W r9            ; Save SOE registers
   SUB.W #2,SP          ; Allocate the frame

                        ; Body of function

   ADD.W #2, SP         ; Deallocate the frame
   POP r9
   POPr10               ; Restore SOE registers
   RET                  ; Return
```

Interrupt Vector

- The interrupt vectors are located in the address range 0FFFFh to 0FF80h.
- The vector contains the 16-bit address of the appropriate interrupt-handler instruction sequences.



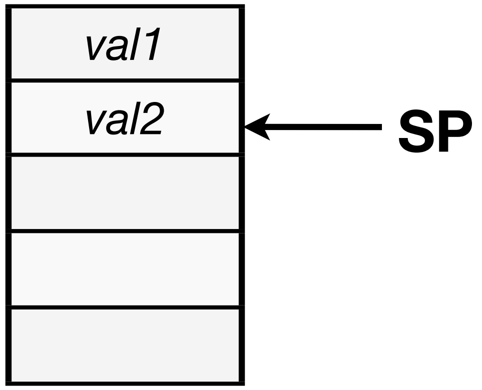
INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
System Reset Power-Up External Reset Watchdog Timeout, Password Violation Flash Memory Password Violation PMM Password Violation	WDTIFG, KEYV (SYSRSTIV)(1) (2)	Reset	0FFFEh	63, highest
System NMI PMM Vacant Memory Access JTAG Mailbox	SVMLIFG, SVMHIFG, DLYLIFG, DLYHIFG, VLRLIFG, VLRHIFG, VMAIFG, JMBNIFG, JMBOUTIFG (SYSSNIV)(1)	(Non)maskable	0FFFCh	62
User NMI NMI Oscillator Fault Flash Memory Access Violation	NMIIFG, OFIFG, ACCVIFG (SYSUNIV)(1) (2)	(Non)maskable	0FFFAh	61
TB0	TBCCR0 CCIFG0 (3)	Maskable	0FFF8h	60
TB0	TBCCR1 CCIFG1 ... TBCCR6 CCIFG6, TBIFG (TBIV)(1) (3)	Maskable	0FFF6h	59
Watchdog Timer_A Interval Timer Mode	WDTIFG	Maskable	0FFF4h	58
USCI_A0 Receive/Transmit	UCA0RXIFG, UCA0TXIFG (UCA0IV)(1) (3)	Maskable	0FFF2h	57
USCI_B0 Receive/Transmit	UCB0RXIFG, UCB0TXIFG (UCAB0IV)(1) (3)	Maskable	0FFF0h	56
ADC12_A	ADC12IFG0 ... ADC12IFG15 (ADC12IV)(1) (3)	Maskable	0FFEEh	55
TA0	TA0CCR0 CCIFG0(3)	Maskable	0FFECh	54
TA0	TA0CCR1 CCIFG1 ... TA0CCR4 CCIFG4, TA0IFG (TA0IV)(1) (3)	Maskable	0FFEAh	53
USCI_A2 Receive/Transmit	UCA2RXIFG, UCA2TXIFG (UCA2IV)(1) (3)	Maskable	0FFE8h	52
USCI_B2 Receive/Transmit	UCB2RXIFG, UCB2TXIFG (UCB2IV)(1) (3)	Maskable	0FFE6h	51
DMA	DMA0IFG, DMA1IFG, DMA2IFG (DMAIV)(1) (3)	Maskable	0FFE4h	50
TA1	TA1CCR0 CCIFG0(3)	Maskable	0FFE2h	49
TA1	TA1CCR1 CCIFG1 ... TA1CCR2 CCIFG2, TA1IFG (TA1IV)(1) (3)	Maskable	0FFE0h	48
I/O Port P1	P1IFG.0 to P1IFG.7 (P1IV)(1) (3)	Maskable	0FFDEh	47
USCI_A1 Receive/Transmit	UCA1RXIFG, UCA1TXIFG (UCA1IV)(1) (3)	Maskable	0FFDCh	46
USCI_B1 Receive/Transmit	UCB1RXIFG, UCB1TXIFG (UCB1IV)(1) (3)	Maskable	0FFDAh	45
USCI_A3 Receive/Transmit	UCA3RXIFG, UCA3TXIFG (UCA3IV)(1) (3)	Maskable	0FFD8h	44
USCI_B3 Receive/Transmit	UCB3RXIFG, UCB3TXIFG (UCB3IV)(1) (3)	Maskable	0FFD6h	43
I/O Port P2	P2IFG.0 to P2IFG.7 (P2IV)(1) (3)	Maskable	0FFD4h	42
RTC_A	RTCRDYIFG, RTCTEVIFG, RTCAIFG, RT0PSIFG, RT1PSIFG (RTCIV)(1) (3)	Maskable	0FFD2h	41
Reserved	Reserved(4)		0FFD0h	40
			⋮	⋮
			0FF80h	0, lowest

Disabling / Enabling Interrupts

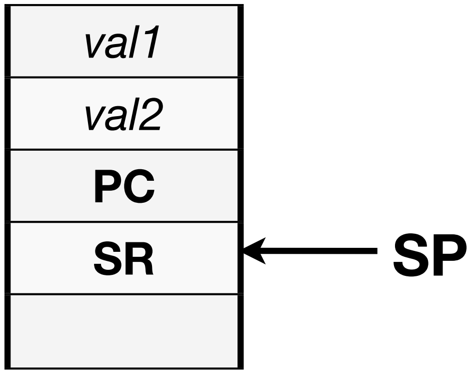
- Intrinsic functions
  - unsigned _EINT()
    - enables global interrupts by setting the GIE bit in the status register.
  - unsigned _DINT()   or   __disable_interrupt()
    - disables global interrupts by clearing the GIE bit in the status register.
    - returns the value of the status register before the GIE bit is cleared
  - __bis_SR_register(GIE)
    - enable all the interrupts by setting the GIE (Global Interrupt Enable) bit in the Status Register
  - …

Interrupt Processing

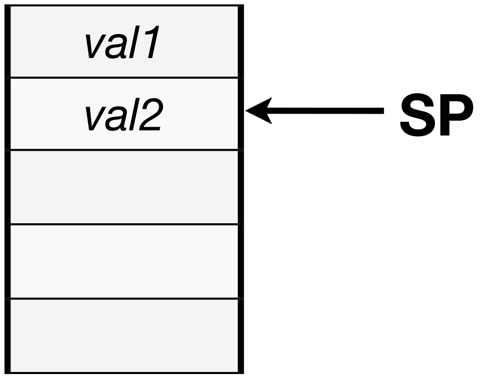
- **Prior to Interrupt Service Routine** (ISR)



- **ISR hardware** (automatically)
  - PC pushed
  - SR pushed
  - Interrupt vector moved to PC
  - GIE (general interrupt enable), CPUOFF, OSCOFF and SCG1 cleared, IFG flag cleared on single source flags



- **reti** (automatically)
  - SR popped
  - PC popped



Summary

- interrupts are used to respond to events
- interrupts can be disabled
- data sharing must be done carefully
- volatile variables are needed to prevent optimizations
- interrupt latency can minimized by careful design of the software