

Design with Microprocessors

Lecture 3

Year 3 CS

Academic year 2023/2024

1st Semester

Lecturer: Radu Dănescu

Stack operations

- Stack pointer (16 bit) – indicates the top of the stack
- Can be accessed using its 8-bit halves, *SPL* and *SPH*, using I/O instructions
- Must be initialized at the beginning of each program that uses stack operations, procedure calls, or interrupts
- Must point to a location in the SRAM
- Because SP is decremented when pushing data on the stack, it is recommended to initialize it with the highest available SRAM address – *RAMEND*

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
0x3D (0x5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	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	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- Example for initializing SP
ldi R16, high(RAMEND)
out SPH, R16
ldi R16, low(RAMEND)
out SPL, R16

The default value is not suitable for use!

Stack operations

- Stack related instructions

push Rx

$\text{Mem}(\text{SP}) = \text{Rx}$
 $\text{SP} = \text{SP} - 1$

pop Rx

$\text{SP} = \text{SP} + 1$
 $\text{Rx} = \text{Mem}(\text{SP})$

rcall adresa

$\text{Mem}(\text{SP}:\text{SP}-1:\text{SP}-2) = \text{PC} + 1$ address of the next instruction, 17 bit

$\text{SP} = \text{SP} - 3$

$\text{PC} = \text{address}^*$

*in fact, the PC is modified by adding an offset value relative to the current address

ret

$\text{SP} = \text{SP} + 3$

$\text{PC} = \text{Mem}(\text{SP}:\text{SP}-1:\text{SP}-2)$

Interrupts

- The interrupts mechanism allows the microcontroller to respond to external events, or events caused by the integrated peripherals.
- In the absence of such events, the processor can execute the main program, or enter a sleep state to save power.
- The interrupt system is activated or de-activated using bit 7 of the status register SREG

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Instructions
 - SEI** – activates the interrupt system ($SREG(7) = 1$)
 - CLI** – de-activates the interrupt system ($SREG(7)=0$)

Interrupts

- Handling an interrupt

1. The peripheral device generates an interrupt request
2. The current finished its execution
3. PC is saved on the stack

TOS = PC

SP = SP - 3

4. The specific interrupt vector is accessed
5. A jump to the *Interrupt Service Routine (ISR)* is executed
6. The interrupt activation flag is cleared (**CLI**)
7. The ISR is executed
8. Return from ISR (**reti**)

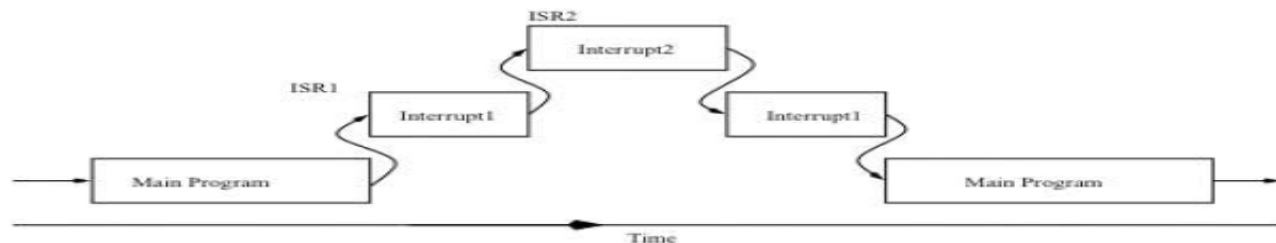
SP = SP + 3

PC = TOS

The interrupt flag is activated (**SEI**)

reti is equivalent to **sei + ret**

If inside an ISR the interrupt system is activated using SEI, nested interrupts can be serviced.



Interrupts

- **Interrupt sources are presented in the Interrupt Vector table**
- Interrupts can be accepted after the finish of the currently executed instruction
- Response time: $\geq 4 \dots 5$ cycles

PC (2/3 bytes) saved on the stack (push)

Stack Pointer \leftarrow Stack Pointer - 2/3;

Jump according to the interrupt vector table

Interrupt system is blocked: bit I, SREG(7) \leftarrow 0

- After 4..5 cycles begins the execution of the ISR (optimal case)
- If the interrupt is used for waking up from sleep mode, the response time is increased with 4..5 cycles
- Returning from ISR (RETI): 4..5 cycles

PC \leftarrow PC saved (pop from stack)

Stack Pointer \leftarrow Stack Pointer + 2/3;

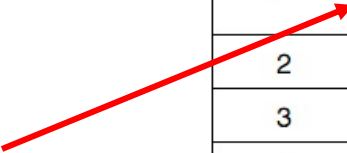
Interrupt system is enabled: bit I, SREG(7) \leftarrow 1

- Bit I can be set/reset directly through instructions SEI & CLI
- **Priority**: decreases with the increasing of the interrupt number
- **Maximum priority** : **Reset**

Interrupts

- Sources of interrupts and their vectors (1)

**Absolute addresses
in the program
(flash) memory**



Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	INT2	External Interrupt Request 2
5	0x0008	INT3	External Interrupt Request 3
6	0x000A	INT4	External Interrupt Request 4
7	0x000C	INT5	External Interrupt Request 5
8	0x000E	INT6	External Interrupt Request 6
9	0x0010	INT7	External Interrupt Request 7
10	0x0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	0x0014	TIMER2 OVF	Timer/Counter2 Overflow
12	0x0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	0x0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	0x001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	0x001C	TIMER1 OVF	Timer/Counter1 Overflow
16	0x001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete

Interrupts

- Sources of interrupts and their vectors(2)

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
19	0x0024	USART0, RX	USART0, Rx Complete
20	0x0026	USART0, UDRE	USART0 Data Register Empty
21	0x0028	USART0, TX	USART0, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030 ⁽³⁾	TIMER1 COMPC	Timer/Counter1 Compare Match C
26	0x0032 ⁽³⁾	TIMER3 CAPT	Timer/Counter3 Capture Event
27	0x0034 ⁽³⁾	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	0x0036 ⁽³⁾	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	0x0038 ⁽³⁾	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	0x003A ⁽³⁾	TIMER3 OVF	Timer/Counter3 Overflow
31	0x003C ⁽³⁾	USART1, RX	USART1, Rx Complete
32	0x003E ⁽³⁾	USART1, UDRE	USART1 Data Register Empty
33	0x0040 ⁽³⁾	USART1, TX	USART1, Tx Complete
34	0x0042 ⁽³⁾	TWI	Two-wire Serial Interface
35	0x0044 ⁽³⁾	SPM READY	Store Program Memory Ready

Interrupts

- External interrupts – caused by activity on the external pins INT7...INT0
- INT7:INT0 are pins of ports **D** and **E** – if the ports are configured as output, software interrupts can be generated by writing these pins.
- Configuring the mode of sensing external interrupts – registers **EICRA** and **EICRB** – a total of 16 bits, 2 bits / interrupt

Bit	7	6	5	4	3	2	1	0									
(0x6A)	<table><tr><td>ISC31</td><td>ISC30</td><td>ISC21</td><td>ISC20</td><td>ISC11</td><td>ISC10</td><td>ISC01</td><td>ISC00</td></tr></table>								ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00										
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W									
Initial Value	0	0	0	0	0	0	0	0									

Bit	7	6	5	4	3	2	1	0									
0x3A (0x5A)	<table><tr><td>ISC71</td><td>ISC70</td><td>ISC61</td><td>ISC60</td><td>ISC51</td><td>ISC50</td><td>ISC41</td><td>ISC40</td></tr></table>								ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40										
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W									
Initial Value	0	0	0	0	0	0	0	0									

ISCn1	ISCn0
0	0
0	1
1	0
1	1

Logic '0' on INTn generates an interrupt request

Any change generates an interrupt request

Falling edge on INTn generates an interrupt request

Rising edge on INTn generates an interrupt request

Interrupts

- Example – incrementing a counter by pressing a button, using the external interrupts mechanism
 - The button is connected to INT0

```
.org 0x0000                                ; vector for the reset interrupt
    rjmp main
.org 0x0002                                ; vector for the external interrupt INT0
    rjmp isr_INT0

main:
    ldi r16, high(RAMEND)                  ; stack pointer initialization – required with interrupts!
    out SPH, R16
    ldi r16, low(RAMEND)
    out SPL, R16

    ldi r16, 0b00000011                   ; handling INT0 – rising edge triggered
    sts EICRA, r16
    ldi r16, 0b00000001                   ; activating INT0
    out EIMSK, r16

    ldi r16, 0xFF                          ; set port E as output – to display the counter
    out DDRE, r16

    ldi r17, 0                             ; initial value of the counter is zero
    sei                                    ; global activation of the interrupt system
```

Interrupts

- Example – incrementing a counter by pressing a button, using the external interrupts mechanism - continuation

loop:

```
    out PORTE, r17      ; the main program – writes the counter to port E
                        ; (assume LEDs are connected here)
```

rjmp loop

isr_INT0:

```
    inc r17             ; beginning of the INT0 service routine
                        ; just increment the counter
    reti               ; return from interrupt
```

- Exercise: how can you solve the problem of counting button presses, without using the interrupt system?
- Exercise 2: Modify the previous example in order to use two buttons connected to INT1 and INT2. The INT1 should increment the counter by 2, while the INT2 should decrement the counter by one.
- Exercise 3: Can you solve the problem of counting button presses, without knowing what type of edge the button causes? (Rising or falling)

Interrupts with Arduino

- Detecting events on the pins, without permanently checking their state by `digitalRead`
- Depending on the Arduino board, the number of external interrupts is variable:

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

- For handling an interrupt, an Interrupt Service Routine (ISR) must be attached. This is done by using the function **`attachInterrupt()`**, with the syntax:

`attachInterrupt(interrupt, ISR, mode)`

interrupt – number of the external interrupt (0, 1, 2, ...)

ISR – name of the Interrupt Service Routine (a function of your program)

mode – triggering mode:

LOW – trigger on level '0'

CHANGE – trigger on pin level change

RISING – trigger on rising edge of the input signal

FALLING – trigger on falling edge of the input signal

Interrupts with Arduino

- De-activating the interrupt handling process is done by calling the function **detachInterrupt()**, with the syntax:

```
detachInterrupt(interrupt)  
    interrupt - interrupt number
```

- If a temporary de-activation of all interrupts is desired, call the function **noInterrupts()**, without parameters. For re-activating the interrupts, call the function **interrupts()**.
- The interrupt system is implicitly active! Deactivation must be done for short periods of time only, otherwise the Arduino functions may be impaired.

Interrupts with Arduino

- **Example:** measuring the width of pulses of a signal (for example, if the signal is from an IR remote receiver, the width of a signal signals whether the pulse is a '0' or a '1').

```
const int irReceiverPin = 2;           // Pin 2, connected to external interrupt 0
const int numberOfEntries = 64;        // Number of transitions that we'll analyze

volatile unsigned long microseconds; // Variable for keeping the number of microseconds since the program started
volatile byte index = 0;               // Position in the transition array
volatile unsigned long results[numberOfEntries]; // Interval time array – the result

void setup()
{
  pinMode(irReceiverPin, INPUT);        // Set the interrupt pin as input
  Serial.begin(9600);                   // USB Serial communication for result display
  attachInterrupt(0, analyze, CHANGE);  // Attach the ISR to interrupt 0, triggered when the signal changes levels
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)           // Check if the maximum number of transitions has been reached
  {
    Serial.println("Durations in Microseconds are:"); // If yes, display the measured intervals
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    index = 0; // After display, re-set the transitions counter and start again
  }
  delay(1000);
}
```

Interrupts with Arduino

- **Example:** measuring the width of pulses of a signal (for example, if the signal is from an IR remote receiver, the width of a signal signals whether the pulse is a '0' or a '1').
- Continued:

```
void analyze()           // The interrupt service routine
{
    if(index < numberOfEntries )    // If we have not reached the end of the array
    {
        if(index > 0)              // But is also not the first detected transition
        {
            results[index] = micros() - microseconds; // Measure the time passed since the last transition
        }
        index = index + 1;          // Increment the index in the transition array
    }
    microseconds = micros();        // Keep the current time, to be used as reference for the next transition
}
```

- The function **micros()** returns the number of microseconds since the program was started.
- For measuring bigger intervals, but with lower precision, you can use **millis()**, which returns the number of milliseconds since the program was started.

Interrupts with Arduino

Attention:

- All global variables that can be modified inside an ISR function must be declared as “**volatile**”. This way, the compiler will know they can change at any moment, and will not try to optimize them by assigning them to registers, or by assuming them constant. They will always be mapped as a location in the RAM.
- Only one ISR function can run at any given time. All other interrupts are, during this time, disabled.
- Since **delay()** and **millis()** rely on the interrupt system, they will not work properly during the execution of an ISR.
- For short delays inside an ISR, one can use the function **delayMicroseconds()**, which does not use interrupts.
- It is not recommended to use the Serial interface inside an ISR.

The interrupt number confusion

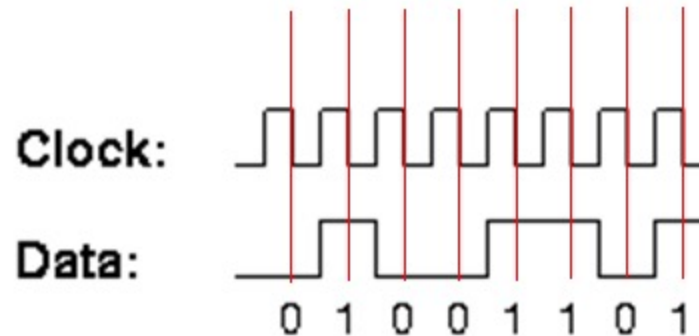
- The number specified as parameter to `attachInterrupt()` is not the same number as the external interrupt number of the AVR microcontroller:
- It is also not the digital pin number.

<code>attachInterrupt</code>	Name	Pin on chip (TQFP)	Pin on board
0	INT4	6	D2
1	INT5	7	D3
2	INT0	43	D21
3	INT1	44	D20
4	INT2	45	D19
5	INT3	46	D18

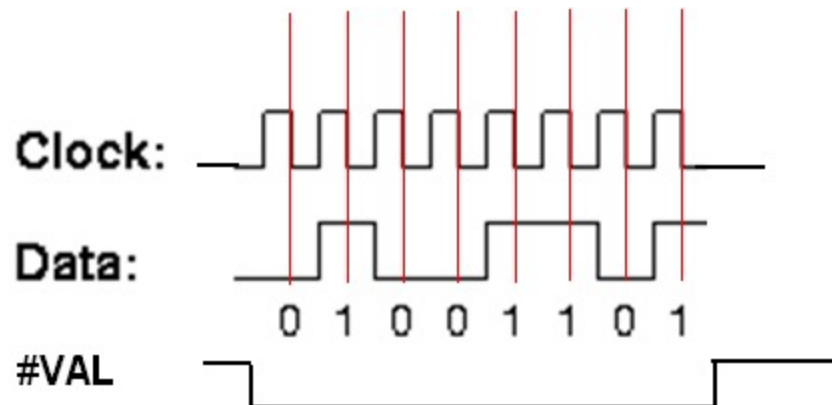
- Solution: use of the **`digitalPinToInterrupt(pin)`** function
 - Example:
`attachInterrupt(digitalPinToInterrupt(21) , isr, FALLING)` will attach to **Arduino interrupt 2**, which is the **AVR interrupt INT0**, connected to **digital pin 21**, the service routine `isr`, which will be triggered when the pin's logic level will fall from '1' to '0'.
- If a digital pin has no interrupt attached to it, the function **`digitalPinToInterrupt`** will return the value **-1** .

Exercises

- Display, using the serial interface, the number of the pins that can be used with external interrupts.
- Write a program capable of receiving serial synchronous data, as shown in the figure below:



- Change the program of the previous exercise, to use an additional signal which marks the beginning and the end of the byte:



Using the AVR interrupts with Arduino

```
// Include the header for the avr interrupt system
#include "avr/interrupt.h";
```

Bit	7	6	5	4	3	2	1	0	
0x39 (0x59)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
volatile int buttonVariable; //public variable that can be modified by the ISR
```

```
void setup(void)
{
    buttonVariable = 0;           // Init the variable shared between the ISR and the main program
    pinMode(21, INPUT);          // Set pin 21 as input (the pin corresponding to INT0)
    EIMSK |= (1 << INT0);        // Activate INT0
    EICRA |= (1 << ISC01);        // Specify INT0 triggering behavior: falling edge
    sei();                       // Global interrupt system activation
    Serial.begin (9600);
}
```

```
void loop()
{
    Serial.println(buttonVariable);
    delay(1000);
}
```

ISCn1	ISCn0
0	0
0	1
1	0
1	1

Falling edge generates request

```
// ISR for INT0, "INT0_vect" is a predefined
ISR(INT0_vect)
{
    buttonVariable ++;
}
```

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	INT2	External Interrupt Request 2
5	0x0008	INT3	External Interrupt Request 3