

ASSEMBLY & C

WEEK 2-2

AGENDA








week	onderwerp	week	week
1	de structuur van AVR-assembly AVR instructies AVR registers en I/O ATmega memory map Atmel Studio AVR expressies en directives AVR addressing modes	3	de structuur van C-programma's ATMEL studio en AVR libc typen, constanten en operatoren AVR register access in C control statements functies & stackframe visibility scope arrays & strings struct & enum
2	flow of control spring instructies, control structuren Arduino UNO AVR studio stack & subroutines interrupts timer/counters switch bounce	4	interrupts in C TM1638 led&key UART PWM & ADC using a TTC-scheduler state diagram

AGENDA

- **AVR Studio**
- breadboard & UNO
- stack & subroutines
- interrupts

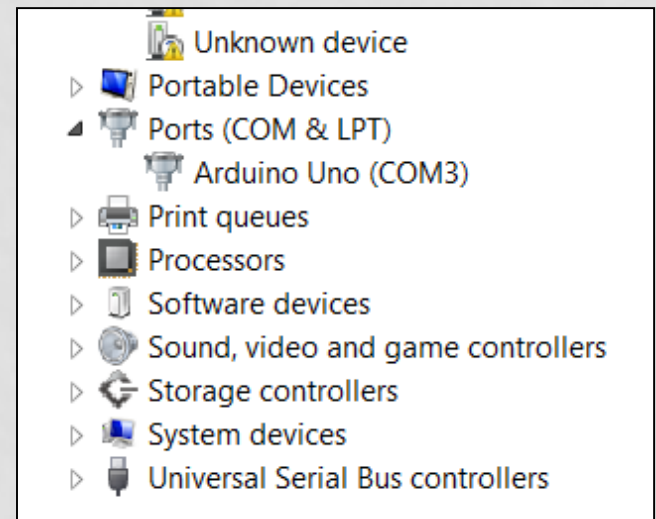
AVRDUDE

- a program for flashing programs to the memories of AVR MCU's
 - installers op BB
 - <http://www.ladyada.net/learn/avr/avrdude.html>
 - <http://download.savannah.gnu.org/releases/avrdude/>
 - Windows : mingw32
- a command line tool
- supports Arduino boards

 avrdude-6.2-mingw32.zip	20-Nov-2015 22:42 218K
 avrdude-6.2-mingw32.zip.sig	20-Nov-2015 22:42 72
 avrdude-6.2.tar.gz	16-Nov-2015 23:12 919K
 avrdude-6.2.tar.gz.sig	16-Nov-2015 23:12 72
 avrdude-6.3-mingw32.zip	17-Feb-2016 10:03 218K
 avrdude-6.3-mingw32.zip.sig	17-Feb-2016 10:03 72
 avrdude-6.3.tar.gz	16-Feb-2016 22:03 888K

AVRDUDE

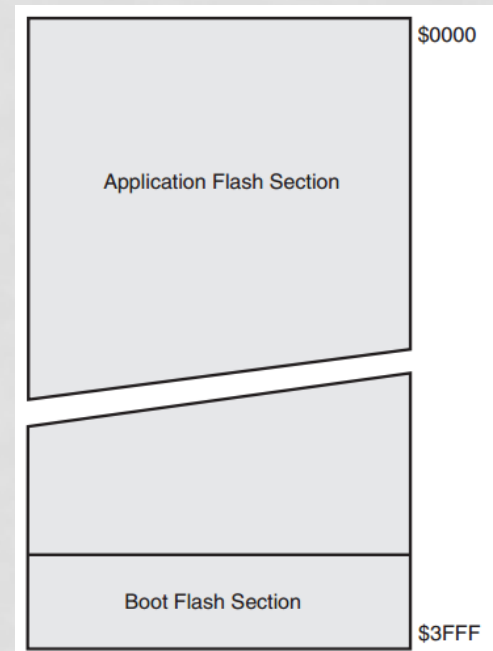
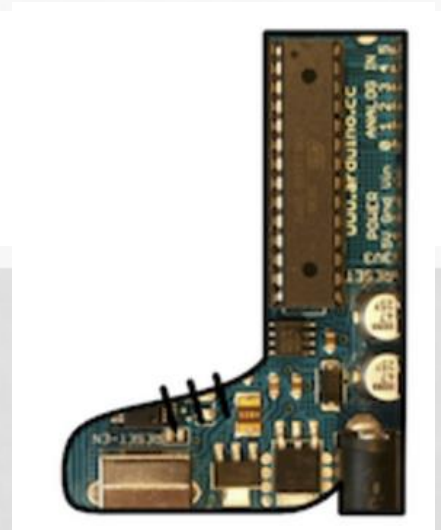
- install AVRdude, e.g. in c:\avrdude
- add c:\avrdude to %PATH%
- connect Arduino and check COM-port in Device Manager
- locate the hex file
- flash the hex file :

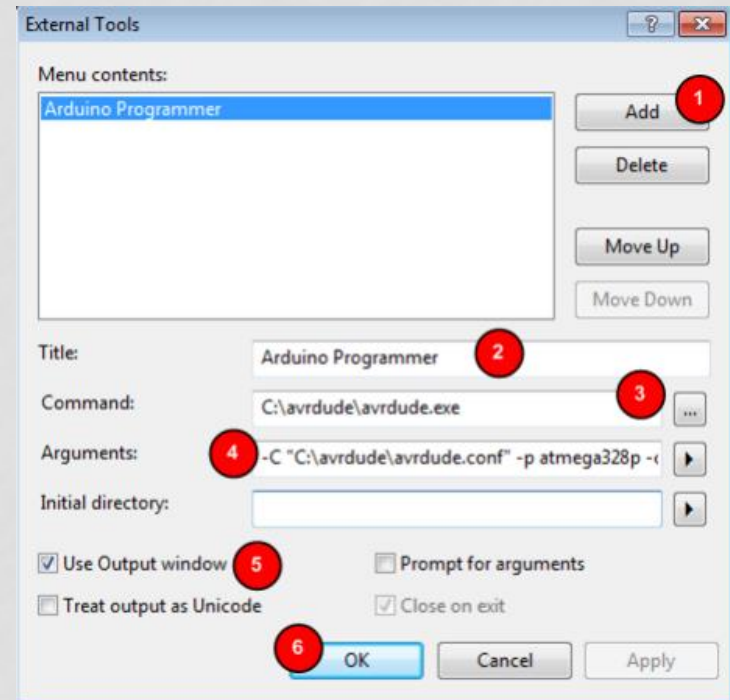
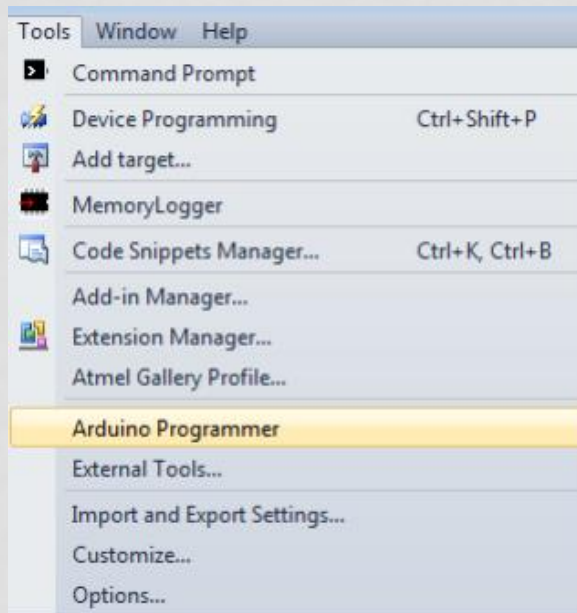
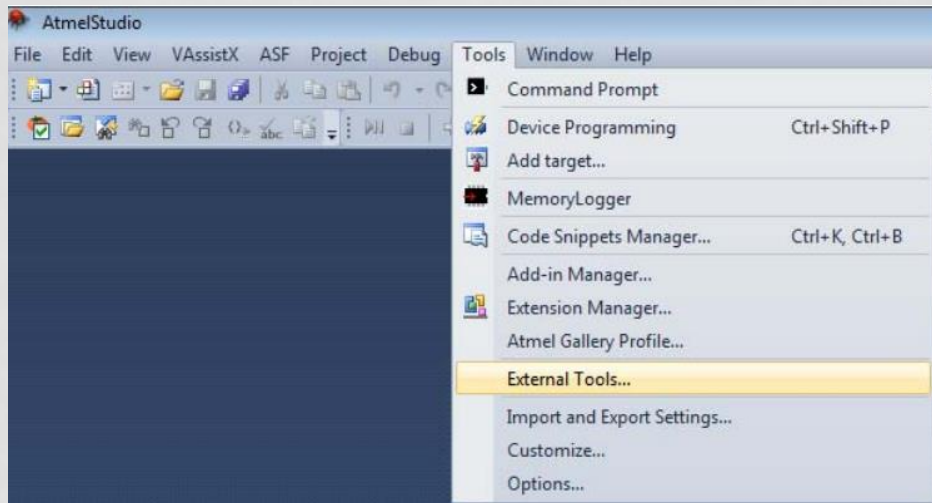


```
> avrdude -v -p atmega328p -c arduino -P COM3  
-U flash:w:test_uno.hex:i
```

BOOTLOADER

- two ways to program :
 - in-system through an SPI serial interface
 - by an On-chip Boot program running on the AVR core
- Arduino : 256 words bootloader called **Optiboot** already installed
- on reset, Optiboot starts and reads the reset reason from MCUSR; in case of external reset Optiboot attempts to download the program
- "start LED" is flashed to indicate that Optiboot is running
- Optiboot will receive commands from AVRdude

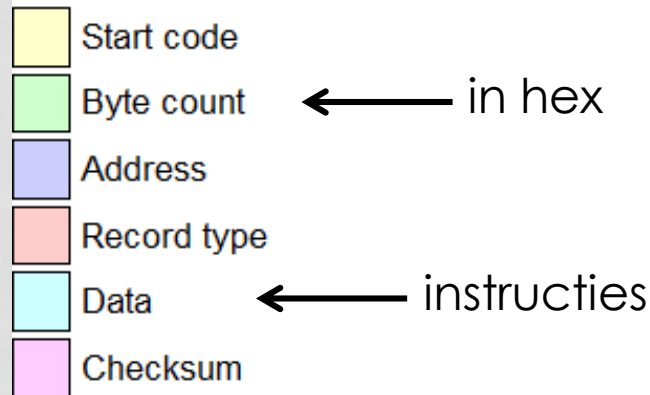




HEX FILE

- is een tekst bestand
- bevat de geheugen adressen en instructies
- bytecount 0x10 = 16 bytes data

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```



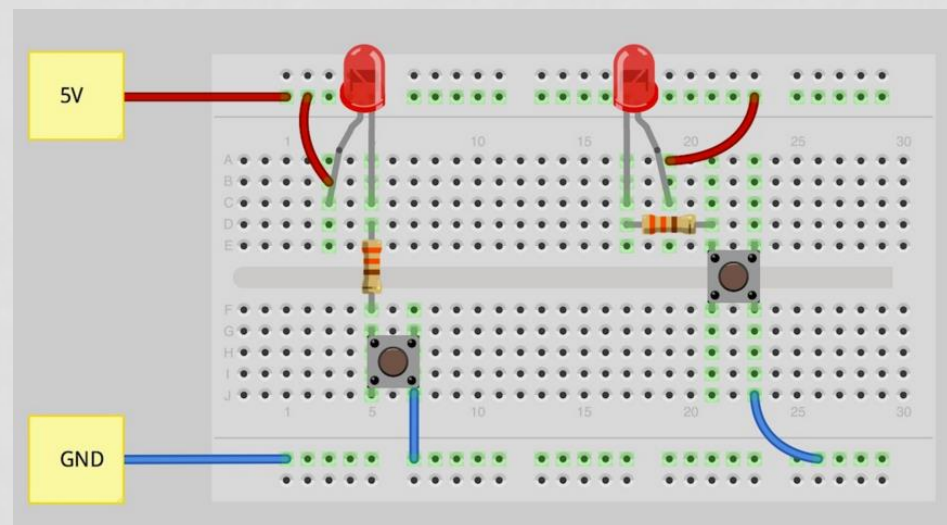
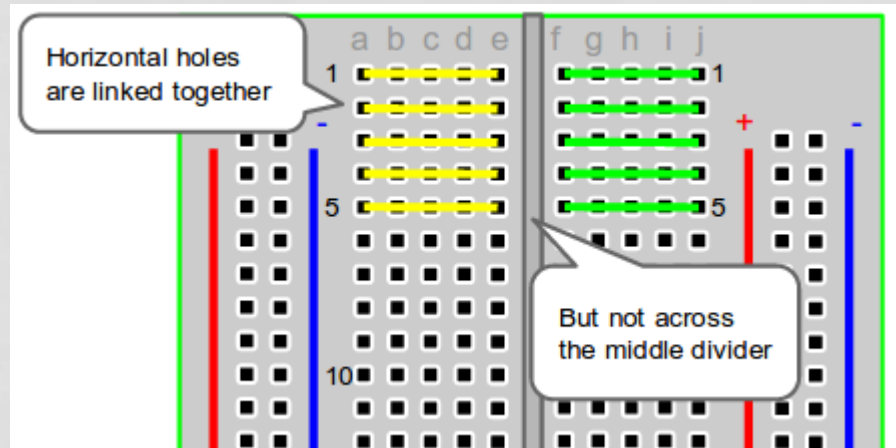
HOE WERK DE ASSEMBLER ?

- forward referentie probleem
 - `jmp label`
 - welk adres invullen voor label ?
- pass 1
 - controleren of syntax errors
 - **symbol table** maken van alle definities en labels, rekening houden met `.org`
- pass 2
 - gebruik symbol table om waarden in te vullen en evalueer assembly functies
 - genereer object code

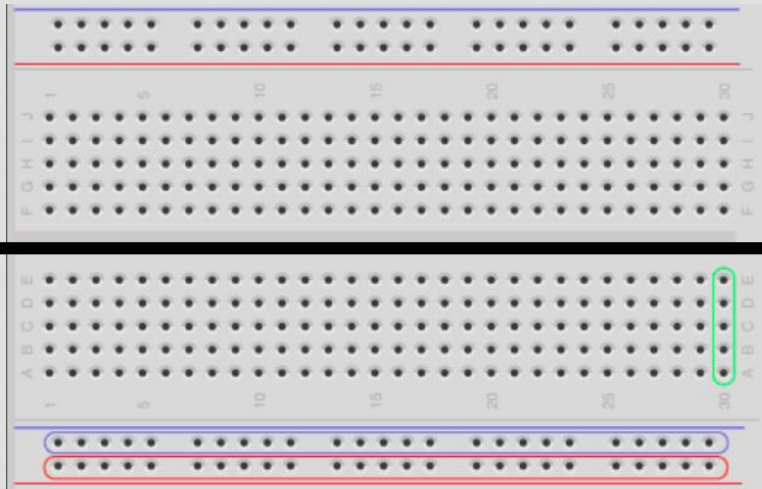
AGENDA

- AVR Studio
- **breadboard & UNO**
- stack & subroutines
- interrupts

BREADBOARD

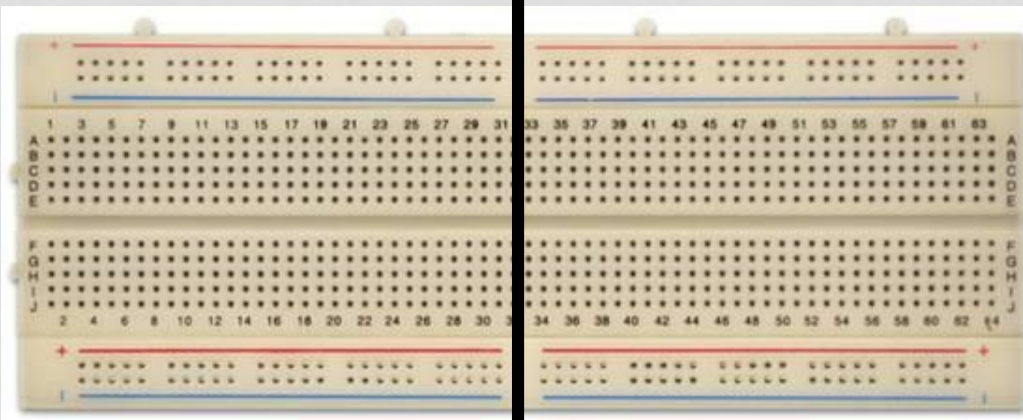


BREADBOARD



$V_{cc} = 5V$
voedingsspanning
ground = 0V

let op : 2 helften zijn
niet verbonden!







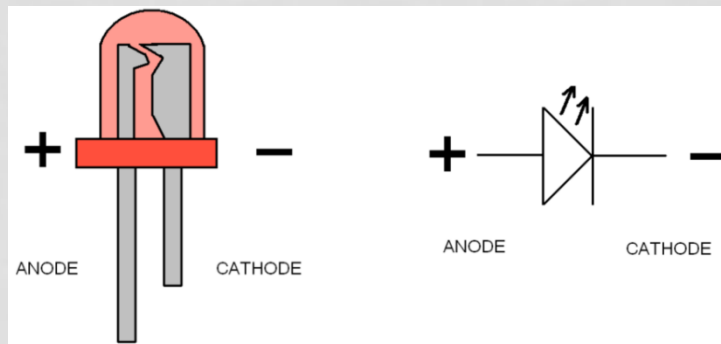
- Domtoren
Utrecht 112m
- Martinitoren
97m
- spanning en
hoogte zijn
relatief t.o.v.
'ground'

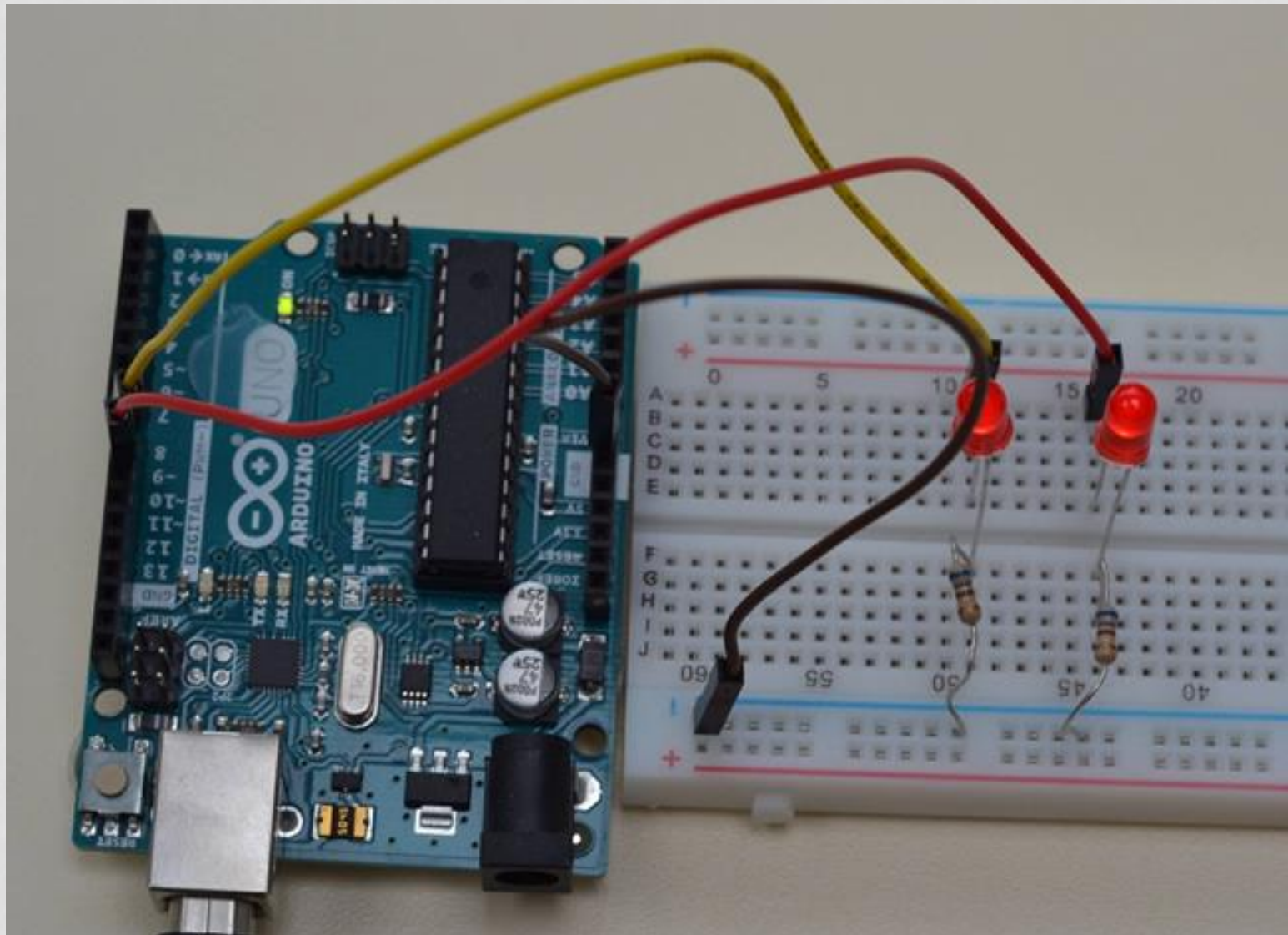


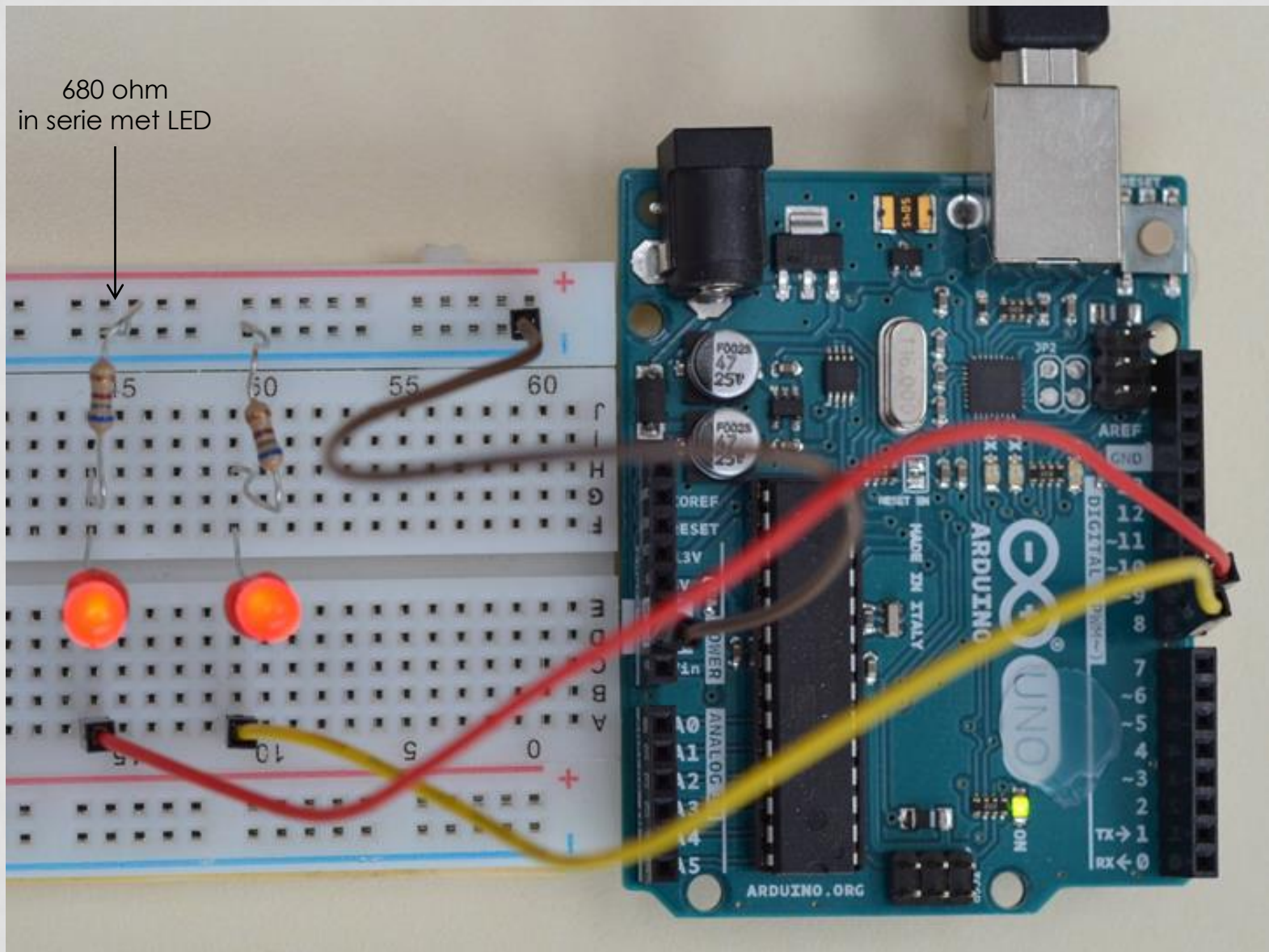
TEST

op Blackboard

Name	Date modified	Type	Size
 UNO_test.hex	15-3-2016 15:17	HEX File	1 KB
 UNO_test_16Mhz.c	15-3-2016 15:17	C File	1 KB





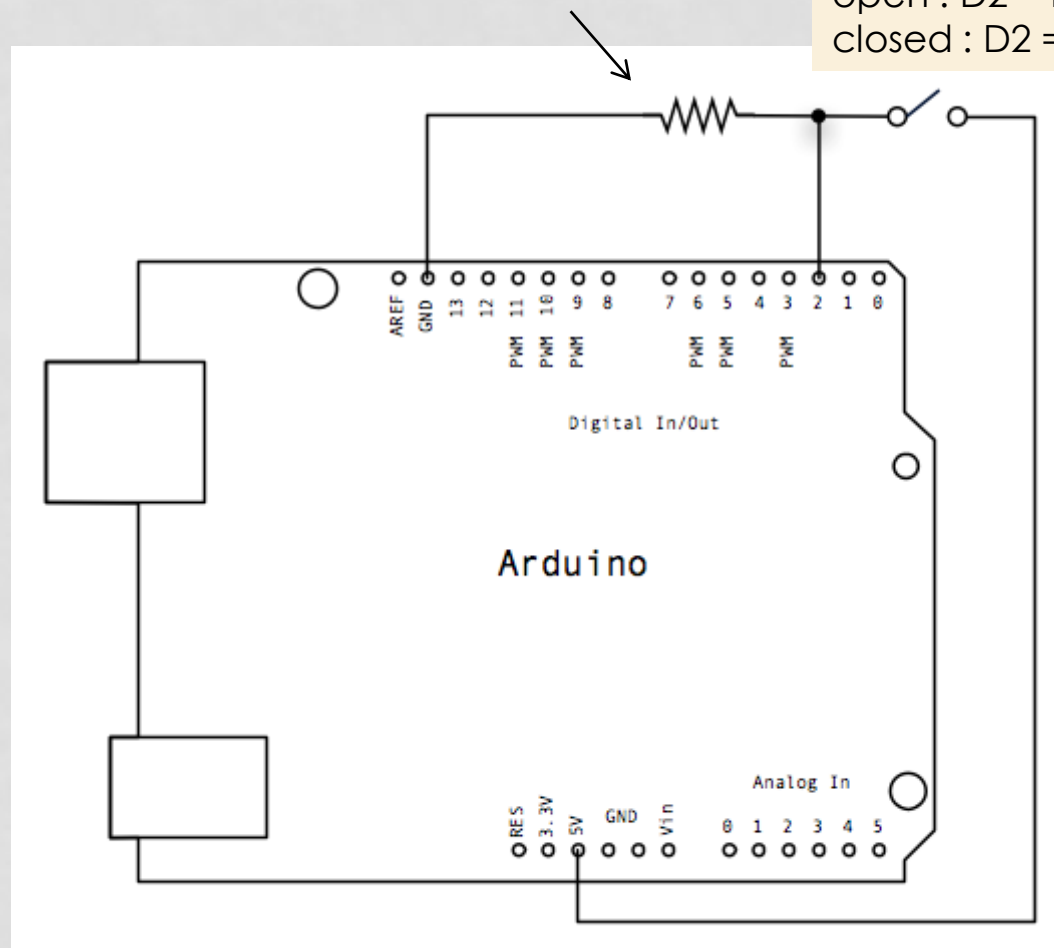


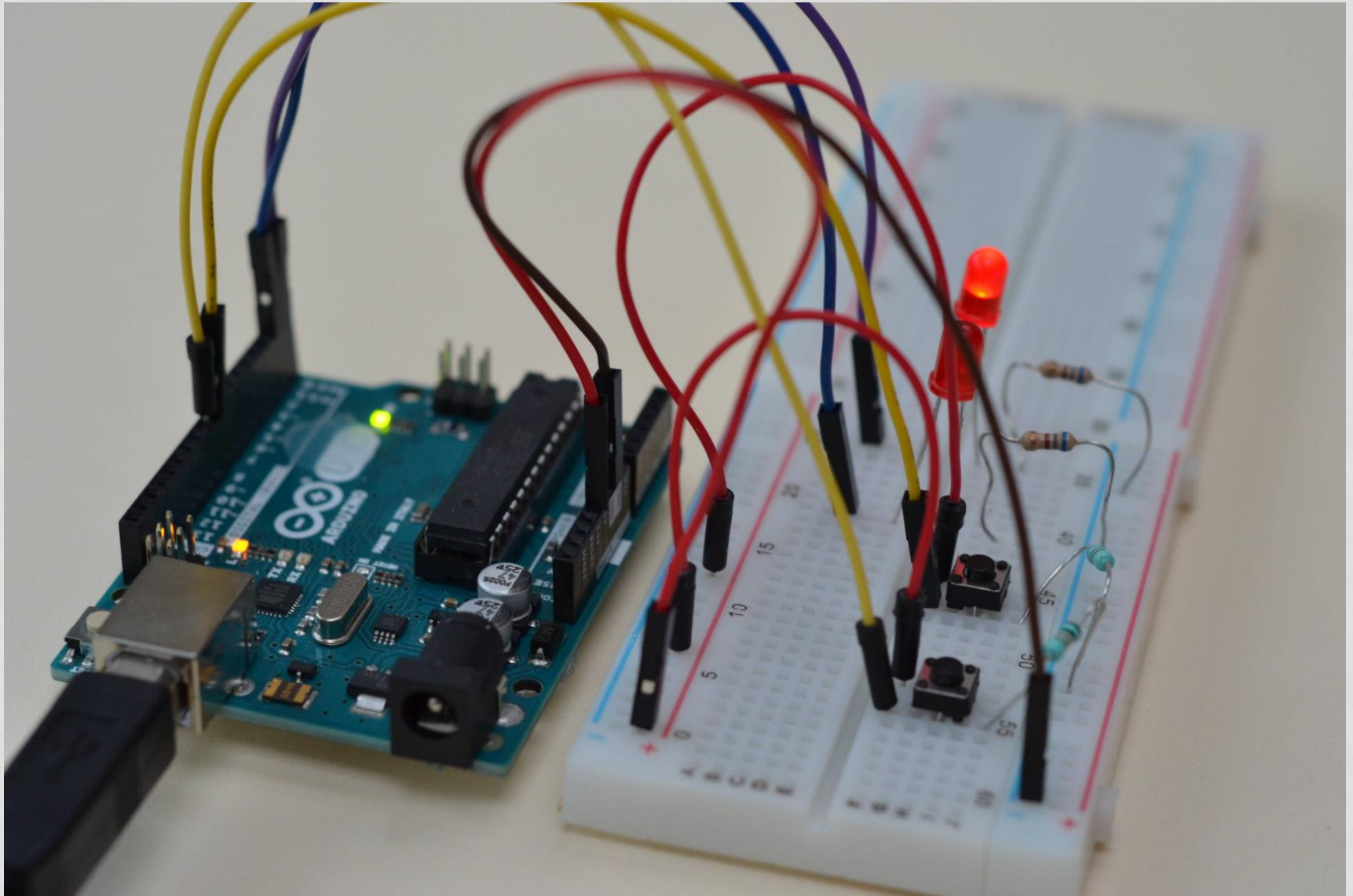
CONNECT A PUSH BUTTON

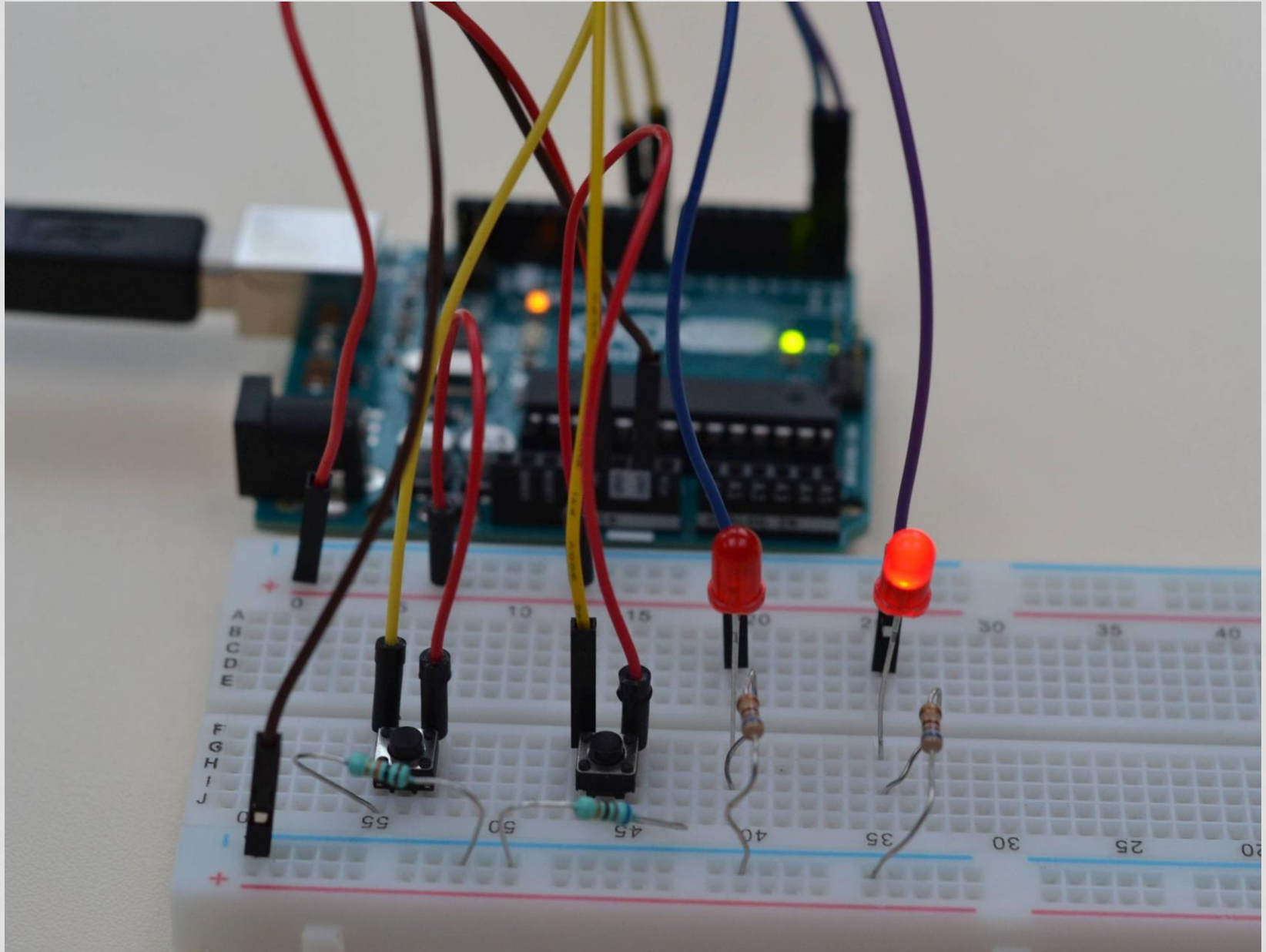
*waarom is een pull-down
weerstand nodig ?*

10 k ohm
pull-down weerstand

open : D2 = low
closed : D2 = high

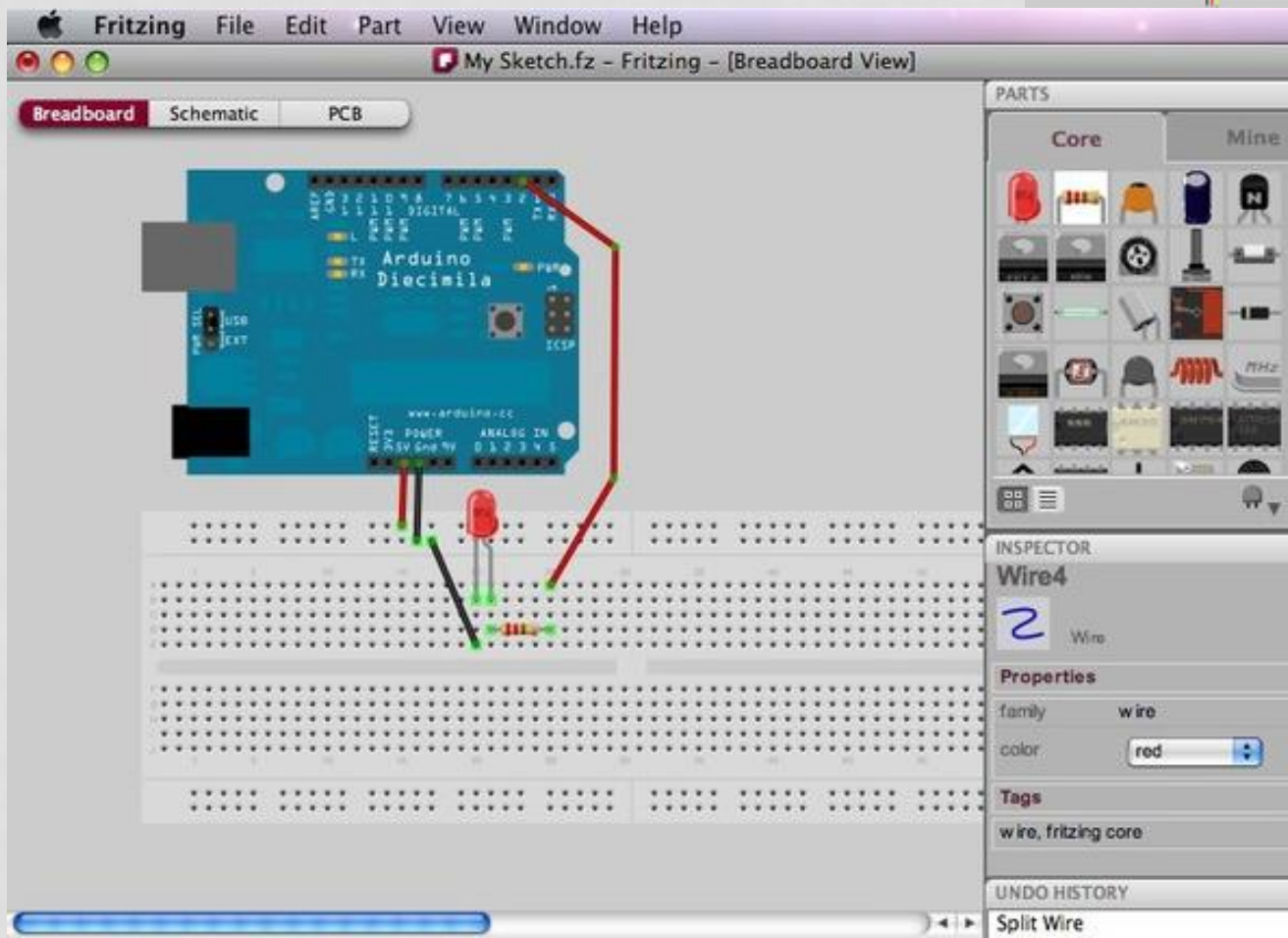
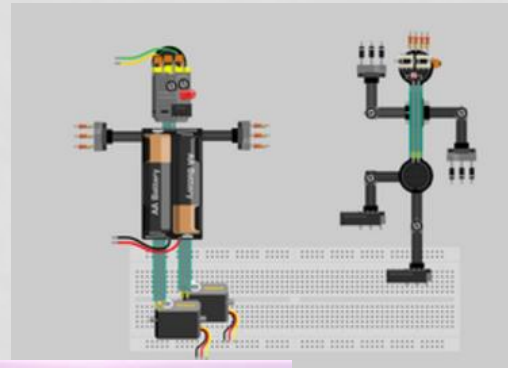






fritzing

electronics
made easy

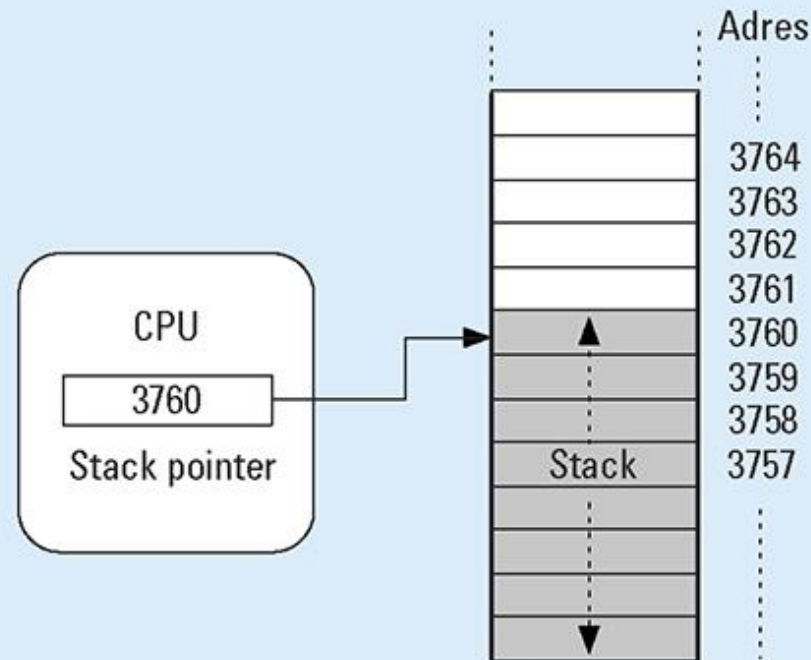


AGENDA

- AVR Studio & UNO
- breadboard & UNO
- **stack & subroutines**
- interrupts

STACK

Figuur 3.18 **Stack**

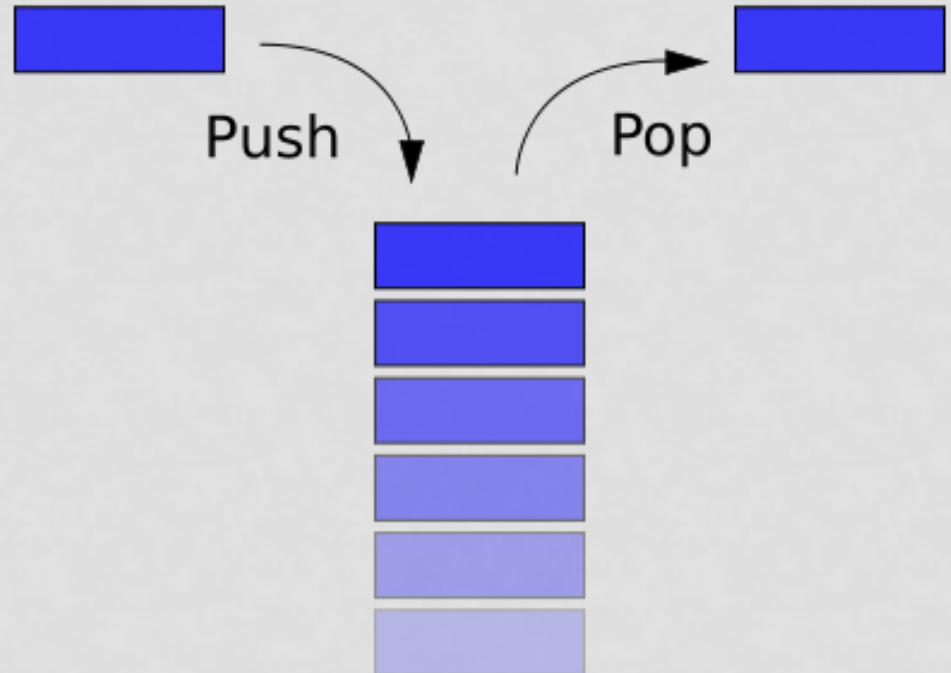


SP wijst altijd naar **top** van de stapel

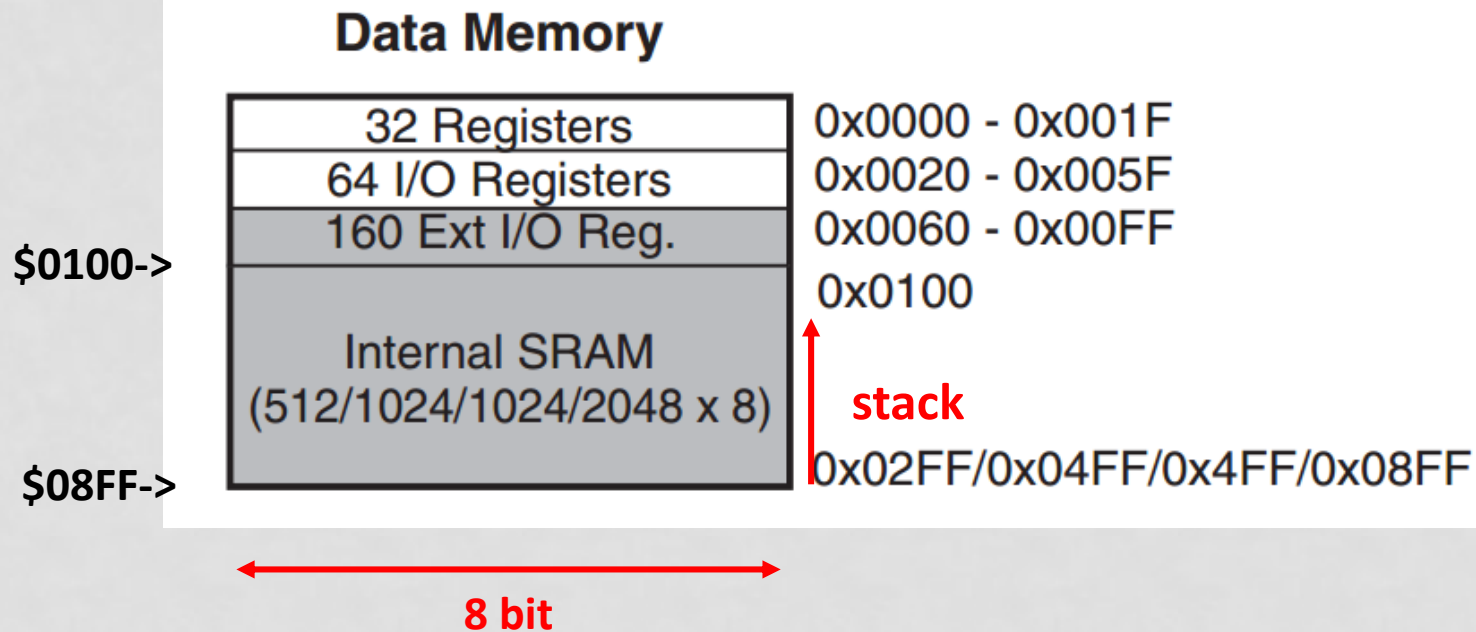
AVR : stack is omgekeerd (*van hoog naar laag adres*)

STACK

- stack = stapel
- PUSH R1
 - inhoud R1 naar adres SP
 - SP++
- POP R1
 - inhoud adres SP naar R1
 - SP--



STACK



SP = 16 bit (SPH + SPL)

IN & OUT WITH ATMEGA328P

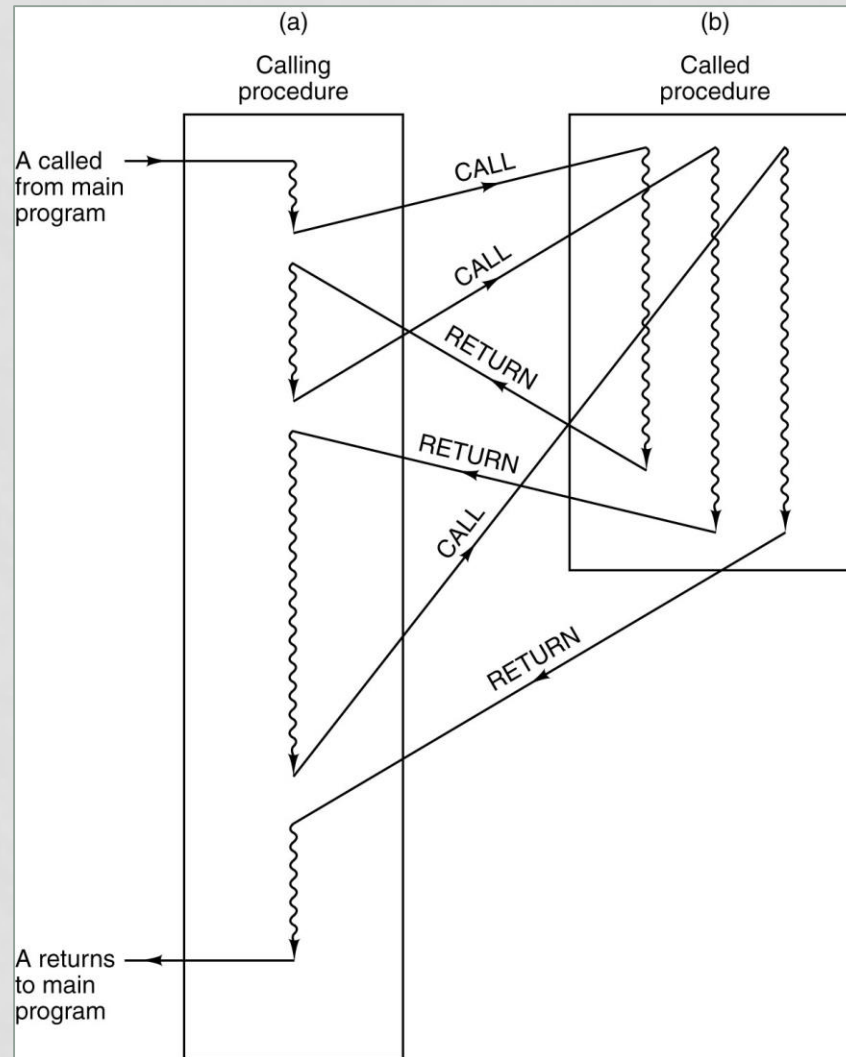
- you can use **LDS & STS** for *all* i/o registers
- the **IN/OUT** instructions can only address I/O registers from 0x00 to 0x3F
- see chapter "register summary" in the data sheet

STACK

- Stack Pointer SP
 - 2x8 bit register in i/o space: SPH+SPL
 - stapel is omgekeerd : van hoog naar laag adres
- initialisatie stack
 - LOW/HIGH zijn functies
 - RAMEND=0x08FF is gedefinieerd in .inc file

```
.include "m328Pdef.inc"
LDI R16, LOW(RAMEND) ;low byte end SRAM
OUT SPL, R16         ;SPL <- R16
LDI R16, HIGH(RAMEND) ;high byte end SRAM
OUT SPH, R16         ;SPH <- R16
```

SUBROUTINES



SUBROUTINES

- waarom subroutines ?
 - maakt modulair programmeren mogelijk
- statements die logisch samenhangen groeperen in subroutine :
 - vermijdt duplicatie van code
 - maakt hergebruik van code eenvoudig

SUBROUTINES

- startadres subroutine = *label*
 - adres om naar toe te springen
- rcall instructie : $PC \leftarrow PC + k + 1$
 - rcall k, met $-2048 \leq k < 2048$
 - k= label : assembler berekent voor jou destination address
- return adres ?
 - door *RCALL* wordt PC+1 opgeslagen *op stack* (2x PUSH)
 - door *RET* wordt waarde return adres *van stack* gehaald (2x POP)

PARAMETER PASSING

- via registers
 - eenvoudig en snel
 - beperkte hoeveelheid data
 - conflict als registers niet uniek zijn toegewezen aan subroutine
- via de stack
 - PUSH en POP
- via datageheugen (SRAM)
 - 2048 registers

OPGAVE

```
int cube(int n)
{
    return n * n * n;
}

main()
{
    int number = 5;
    int result = cube (number);
    return 0;
}
```

```

.include "m328Pdef.inc"
.def number = r16    ; defineer number
.def n = r17         ; defineer n
.def result = r18    ; defineer result
.def tmp = r19       ; defineer tmp
.org 0x0000          ; volgende instructie op de int vector van RESET

```

init:

```

    ; opzetten stack pointer (nodig voor rcall)
    ldi tmp, LOW(RAMEND)
    out SPL, tmp
    ldi tmp, HIGH(RAMEND)
    out SPH, tmp

```

main:

```

    ldi number,5      ; number=5
    rcall cube        ; return value in result (aannamen  $n^3 < 256$ )

```

lus:

```

    rjmp lus

```

cube :

```

    mov n, number     ; n = number
    mul n, n          ; r0 = n*n
    mul n, r0         ; r0 = n*r0
    mov result,r0     ; result = r0
    ret              ; return

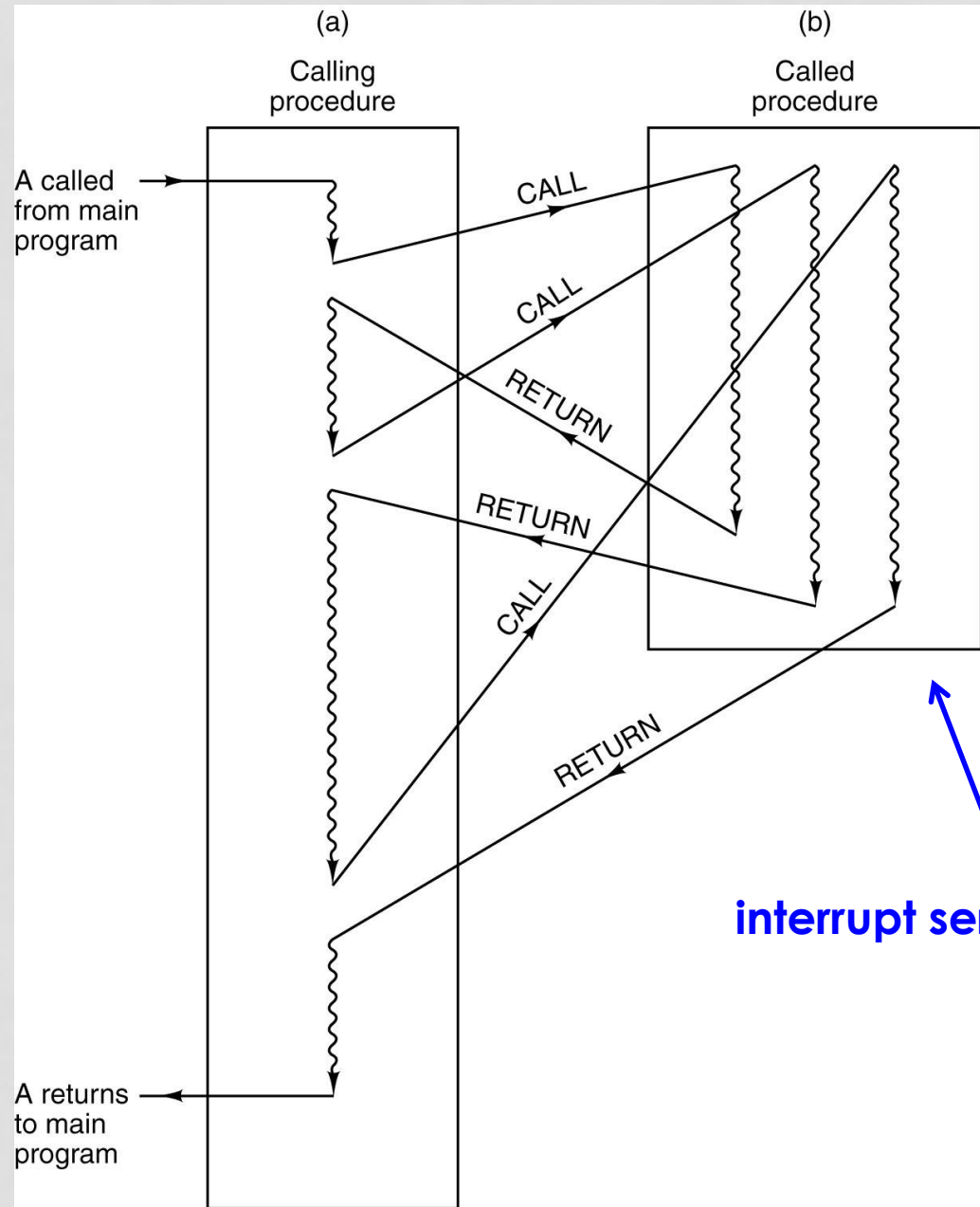
```


AGENDA

- AVR Studio
- breadboard & UNO
- stack & subroutines
- **interrupts**

INTERRUPT

- vergelijkbaar met call subroutine
- maar initiatief komt niet van CPU maar van (CPU-) externe bron
- wat gebeurt er ?
 - CPU onderbreekt het huidige programma
 - CPU 'springt' naar **ISR** (= Interrupt Service Routine)
 - return : CPU hervat onderbroken programma



INTERRUPT BRONNEN

- typen :
 - intern of extern MCU
 - software of hardware
- interne interrupt : ge-triggered door on-chip component
 - timer 0, 1, 2
 - USART
- externe interrupt : ge-triggered door extern device
 - RESET (pin 9)
 - INT0..2 (pin 16, 17, 3)

INTERRUPT BRONNEN

- traps
 - software interrupts die worden ge-triggered door een machine instructie
 - afgehandeld door trap-handler
 - bekendste voorbeeld : afhandelen van system calls
 - ander voorbeeld : delen door 0

INTERRUPTS ATMEGA238P

*reset and Interrupt
vectors placement
in code segment*

- reset
- externe interrupts
- timer interrupts
- serielle interfaces

Table 12-1. Reset and Interrupt Vectors in ATmega48A and ATmega48PA

Vector No.	Program Address	Source	Interrupt Definition
1	0x000	RESET	External Pin, Power-on Reset, Brown-out F
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	PCINT0	Pin Change Interrupt Request 0
5	0x004	PCINT1	Pin Change Interrupt Request 1
6	0x005	PCINT2	Pin Change Interrupt Request 2
7	0x006	WDT	Watchdog Time-out Interrupt
8	0x007	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x008	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x009	TIMER2 OVF	Timer/Counter2 Overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x00B	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x00C	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x00D	TIMER1 OVF	Timer/Counter1 Overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x010	TIMER0 OVF	Timer/Counter0 Overflow
18	0x011	SPI, STC	SPI Serial Transfer Complete
19	0x012	USART, RX	USART Rx Complete

INTERRUPTS ATMEGA238P

- zie hoofdstuk "interrupts" in datasheet
- worden geen interrupts gebruikt, dan kan programma code worden geplaatst op deze adressen
 - anders code plaatsen vanaf 0x030

INT0 EN INT1

(PCINT14/ $\overline{\text{RESET}}$) PC6	<input type="checkbox"/>	1	ATmega328P	28	<input type="checkbox"/>	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	<input type="checkbox"/>	2		27	<input type="checkbox"/>	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	<input type="checkbox"/>	3		26	<input type="checkbox"/>	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	<input type="checkbox"/>	4		25	<input type="checkbox"/>	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	<input type="checkbox"/>	5		24	<input type="checkbox"/>	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	<input type="checkbox"/>	6		23	<input type="checkbox"/>	PC0 (ADC0/PCINT8)
VCC	<input type="checkbox"/>	7		22	<input type="checkbox"/>	GND
GND	<input type="checkbox"/>	8		21	<input type="checkbox"/>	AREF
(PCINT6/XTAL1/TOSC1) PB6	<input type="checkbox"/>	9		20	<input type="checkbox"/>	AVCC
(PCINT7/XTAL2/TOSC2) PB7	<input type="checkbox"/>	10		19	<input type="checkbox"/>	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	<input type="checkbox"/>	11		18	<input type="checkbox"/>	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	<input type="checkbox"/>	12		17	<input type="checkbox"/>	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	<input type="checkbox"/>	13		16	<input type="checkbox"/>	PB2 ($\overline{\text{SS}}$ /OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	<input type="checkbox"/>	14		15	<input type="checkbox"/>	PB1 (OC1A/PCINT1)

INTERRUPT PRIORITEIT

- de plaats in de vector tabel bepaalt de prioriteit :
 - RESET heeft hoogste prioriteit
 - daarna INT0, enz.

AVR INTERRUPTS

- op de adressen uit de vector tabel staan jump instructies : `jmp 'startadres ISR'`
- een interrupt die niets doet kan eenvoudige een RETI instructie hebben op zijn adres (in de vector tabel)

AVR INTERRUPTS

- bij elke interrupt hoort een eigen adres in de vector tabel; dit adres wordt geladen in de PC als de interrupt optreedt
 - adres \$0000 is de RESET interrupt
 - adres \$0002 is external interrupt 0
 - enz.
- zie m328Pdef.inc voor adreslabels :

```
; ***** INTERRUPT VECTORS *****  
.equ  INT0addr = 0x0002 ; External Interrupt Request 0  
.equ  INT1addr = 0x0004 ; External Interrupt Request 1
```

ENABLE EN DISABLE

- elke interrupt bron kan individueel worden uitgezet
 - **behalve** de RESET interrupt
- in status register is een Global Interrupt Enable bit voor alle interrupts
 - maar RESET kan *niet* worden uitgezet

WAT GEBEURT ER BIJ EEN INTERRUPT ?

- maak huidige instructie af
- zet GIE flag in SREG = 0
 - (geen andere interrupts afhandelen)
- PC+1 op stack saven (2 x PUSH)
- PC wordt interrupt vector
- hier staat (meestal) jump naar ISR

WAT GEBEURT ER BIJ EEN INTERRUPT ?

- ISR :
 - registers (inclusief Status Register) op stack saven ; zelf doen !
 - afhandelen interrupt : zo kort mogelijk
 - registers van stack halen : zelf doen !
 - RETI :
 - PC van stack halen (2 x POP)
 - zet GIE flag in SREG = 1
- weer doorgaan met onderbroken programma

VRAGEN

- waarom moet de SP worden opgezet ?
- waarom moet je in ISR het Status Register saven ?
- hoe kun je dit doen ?

is interrupt nesting mogelijk ?

- nee, de hardware zet de GIE flag = 0 voor laden van interrupt vector
- RETI instructie maakt GIE flag = 1
- (maar .. we kunnen wel in ISR naar SREG schrijven)

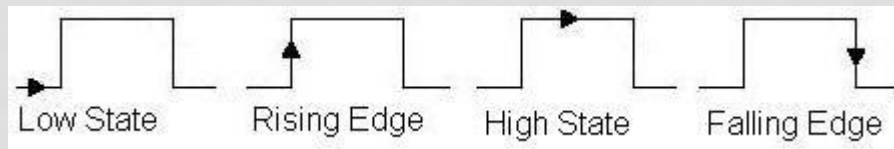
VRAGEN

- wat gebeurt er als meerdere interrupts tegelijk plaatsvinden ?
 - prioriteit o.b.v. plaats in interrupt vector tabel
 - direct na RETI wordt de volgende interrupt met *hoogste* prioriteit afgehandeld

Algemeen : hou ISR zo kort mogelijk !
zet vlag in ISR en handel deze in 'main-lus' af

EXTERNE INTERRUPT

- External Interrupt Control Register (EICRA)
 - externe interrupt ge-triggered door niveau of flank ?
- External Interrupt Mask Register (EIMSK)
 - enable / disable INT0 of INT1
- Global Interrupt Enable
 - = bit in Status Register
 - enable / disable alle interrupts
 - hiervoor speciale instructies : SEI en CLI



```

;*
;* aanslagen tellen via interrupt op INT1=PD3
;* verbind de knopjes met port D en de LEDs met port B
;*

.include "m328Pdef.inc"
.def counter=r16      ; variabele Counter
.def saveSR=r17       ; om SREG tijdelijk te bewaren
.def tmp=r18          ; hulp register
.cseg
.org 0x0000

rjmp init             ; bij opstarten & reset naar init

.org INT1addr         ; zet rjmp op adres INT1addr
rjmp handle_INT_1     ; adres ISR

init:
ldi tmp, LOW(RAMEND)  ; init stack pointer
out SPL, tmp

```

```

ldi tmp, HIGH(RAMEND)
out SPH, tmp
; init ports
ser tmp
out DDRB, tmp      ; Port B is output port (via LEDs)
out PORTB, tmp     ; LEDs uitzetten
clr tmp
out DDRD, tmp      ; Port D is input port (via switches)

clr Counter        ; Counter = 0
; init interrupt
ldi tmp, (1<<ISC11)|(1<<ISC10)
                    ; rising edge of INT1 generates interrupt

sts EICRA , tmp
ldi tmp, (1<<INT1)  ; enable INT1
out EIMSK, tmp
sei                ; enable interrupts (SREG)

main:
rjmp main

```

```
; interrupt service routine
```

```
handle_INT_1:
```

```
    in saveSR, SREG      ; save SREG  
    inc counter          ; verhoog teller  
    out PORTB, counter    ; schrijf counter naar Leds  
    out SREG, saveSR      ; restore SREG  
    reti                 ; return from interrupt
```