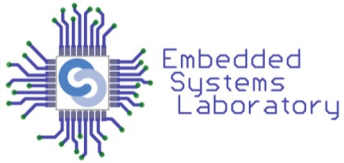


PROIECTAREA CU MICROPROCESOARE

Cursul 3
Întreruperi

Facultatea de Automatică și Calculatoare
Politehnica București


mov rax, rbx
add rax, rcx
HELLO IT'S ME THE KEYBOARD
I HAVE AN IMPORTANT MESSAGE
E
FALSEKNEES.COM © 2016

Întreruperi

- Un eveniment extern procesului care este în execuție. Evenimentul provoacă o schimbare în fluxul normal al execuției instrucțiunilor; de obicei, generat de dispozitive hardware externe CPU-ului.
- Punctul cheie este că întreruperile sunt asincrone față de procesul curent.
- De obicei, indică faptul că un anumit dispozitiv necesită atenție.

De ce avem nevoie de ele?

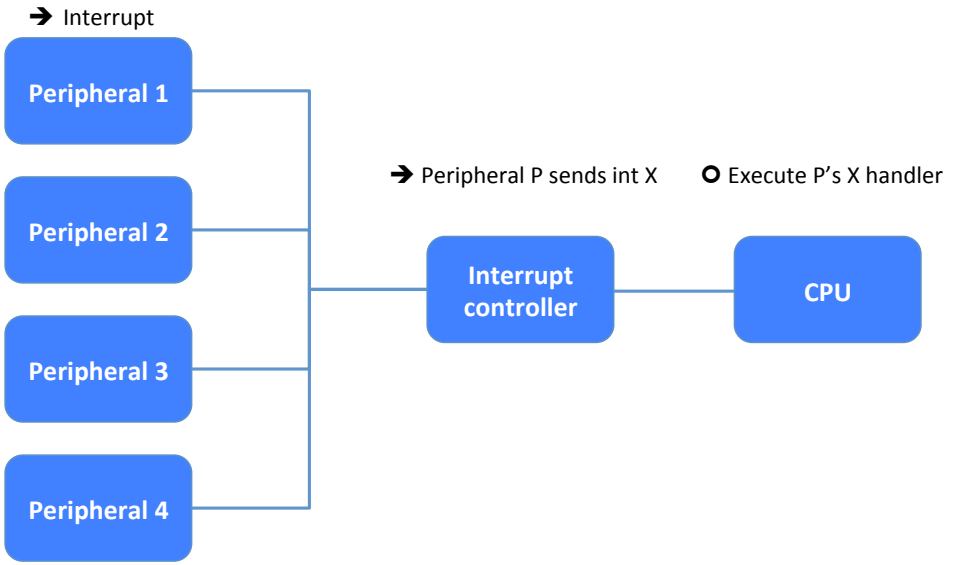
- Sistemele de calcul au multe periferice externe
  - Tastatură, mouse, ecran, unități memorie, scanner, imprimantă, placă de sunet, cameră web etc.
  - Toate aceste dispozitive au nevoie ocazional de interacțiune cu procesorul
    - Dar nu putem să știm când
  - Vrem să ținem CPU ocupat (sau în sleep) între evenimente
  - Trebuie să existe un mecanism prin care CPU determină care periferic are nevoie de atenție

Soluție posibilă: polling

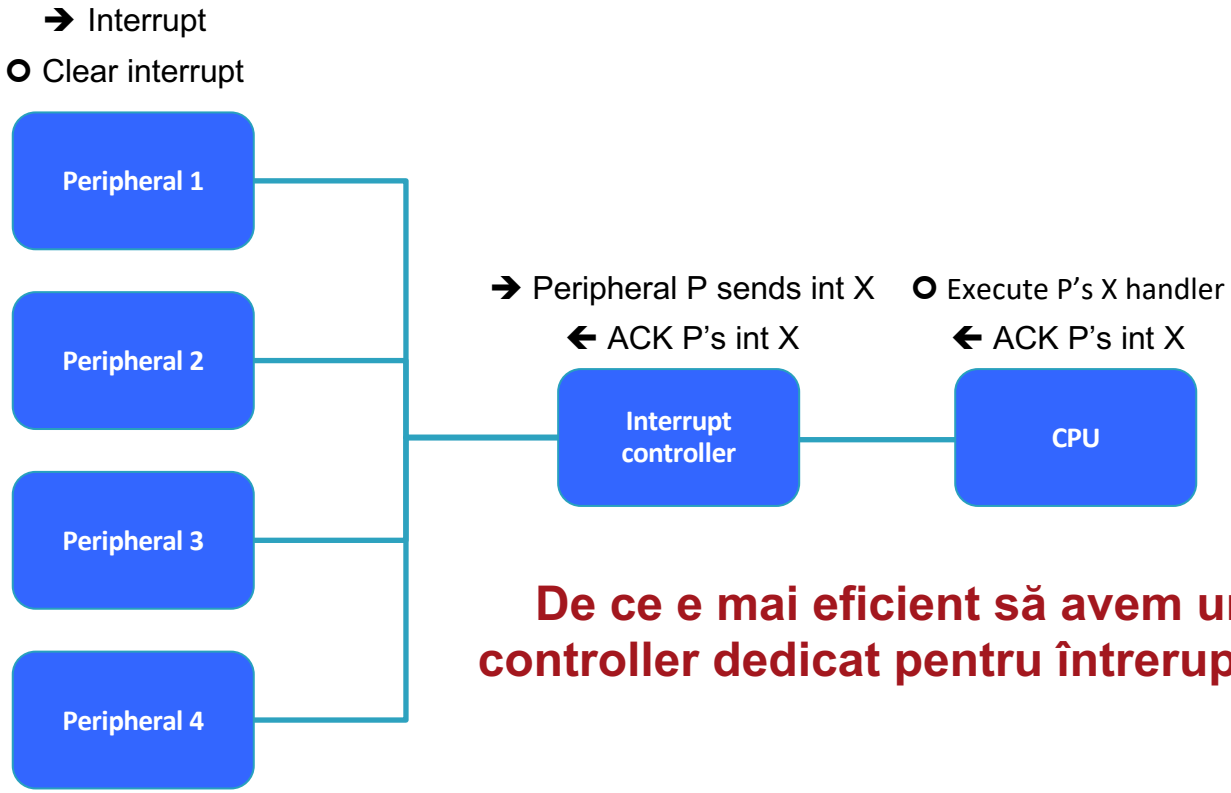
- CPU verifică periodic dacă un periferic are nevoie de atenție
  - Polling este ca și cum te-ai uita la telefon în fiecare secundă ca să verifici dacă cineva te sună
  - Pro: poate fi eficient dacă evenimentele vin în succesiune rapidă
  - Con: ține procesorul ocupat chiar și atunci când nu se petrece nici un eveniment

Alternativă: Întreruperi

- Fiecare periferic are o linie prin care poate să semnalizeze procesorului că are nevoie de atenție (IRQ)
- Când un IRQ sosește la procesor, se lansează o rutină de tratare a întreruperii (interrupt handler)
- Nu există nici un ciclu pierdut atunci când nu există cereri de întrerupere



Cum funcționează întreruperile?

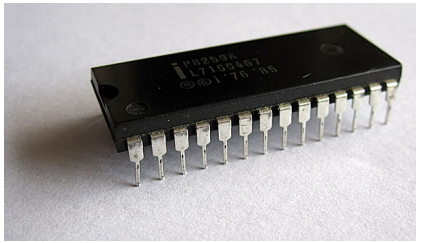


De ce e mai eficient să avem un controller dedicat pentru întreruperi?

Controllerul de întreruperi

- **Tratează întreruperi simultane**
  - Recepționează cereri de întrerupere în timp ce procesorul tratează
- **Setează flag-uri de întrerupere**
  - CPU poate să facă poll pe aceste flag-uri în loc să sară imediat într-un interrupt handler
- **Multiplexează multe linii de IRQ la puține linii**
  - CPU nu are nevoie de o linie dedicată de IRQ la fiecare periferic

Fun fact: Controllerele de întreruperi au fost circuite integrate separate!



Intel 8259A IRQ chip

Cum să lucrezi cu întreruperi

1. Procesorul inițializează perifericul și activează întreruperile pe care dorește să le primească de la acel periferic
2. Procesorul inițializează controlerul de întreruperi cu o listă de priorități pentru fiecare întrerupere activată
3. Codează rutina de tratare a întreruperii și plaseaz-o în memoria de program a.î. procesorul s-o execute atunci când este nevoie
4. După rularea rutineri de tratare a întreruperii, revino la execuția normală a codului

Execuția rutinelor de tratare a întreruperilor

ÎNTRERUPERE

1. Așteaptă ca instrucțiunea curentă să se execute
2. Salvează PC în stivă
3. Salvează toate registrele active în stivă
4. Salt la adresa rutinei de tratare a întreruperii (RTI) specificată în tabela de vectori de tratare a întreruperilor
5. La întoarcerea din RTI, reface registrele active și PC salvate pe stivă

Întreruperi la ATmega324P

- 32 întreruperi specificate în ordinea priorității
- Toate sunt mascabile
  - Mai puțin RESET
- Un periferic poate avea unul sau mai mulți vectori de întrerupere

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	PCINT0	Pin Change Interrupt Request 0
6	$000A	PCINT1	Pin Change Interrupt Request 1
7	$000C	PCINT2	Pin Change Interrupt Request 2
8	$000E	PCINT3	Pin Change Interrupt Request 3
9	$0010	WDT	Watchdog Time-out Interrupt
10	$0012	TIMER2_COMPA	Timer/Counter2 Compare Match A
11	$0014	TIMER2_COMPB	Timer/Counter2 Compare Match B
12	$0016	TIMER2_OVF	Timer/Counter2 Overflow
13	$0018	TIMER1_CAPT	Timer/Counter1 Capture Event
14	$001A	TIMER1_COMPA	Timer/Counter1 Compare Match A
15	$001C	TIMER1_COMPB	Timer/Counter1 Compare Match B
16	$001E	TIMER1_OVF	Timer/Counter1 Overflow
17	$0020	TIMER0_COMPA	Timer/Counter0 Compare Match A
18	$0022	TIMER0_COMPB	Timer/Counter0 Compare match B
19	$0024	TIMER0_OVF	Timer/Counter0 Overflow
20	$0026	SPI_STC	SPI Serial Transfer Complete
21	$0028	USART0_RX	USART0 Rx Complete

Mecanismul de tratare al întreruperilor la ATmega324P

1. Finalizează instrucțiunea curentă
2. Push la PC + 2 în stivă
3. Push la SREG și orice alte registre generale folosite în bucla principală
4. Salt la adresa din tabela vector întreruperi corespunzătoare tipului întreruperii sosite
5. În tabelă trebuie să fie un jmp la RTI
6. Execuție RTI
7. La retur din RTI, pop la registre generale utilizate, pop la SREG
8. Pop la PC + 2 -> reluarea execuției în bucla main()

Inițializare întreruperi pentru un periferic în avr-gcc

Să zicem că vrem să activăm întreruperile la recepționarea unui caracter pe interfața USART

- Setăm bitul din registrul perifericului (USART) care activează întreruperea (*RXCIE0 din UCSRB0*) – local interrupt enable
- Setăm bitul *I din SREG* – global interrupt enable
- Scriem codul pentru RTI – *ISR(USART0_RX_vect)*

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	

Exemplu: întreruperi externe la ATmega324P

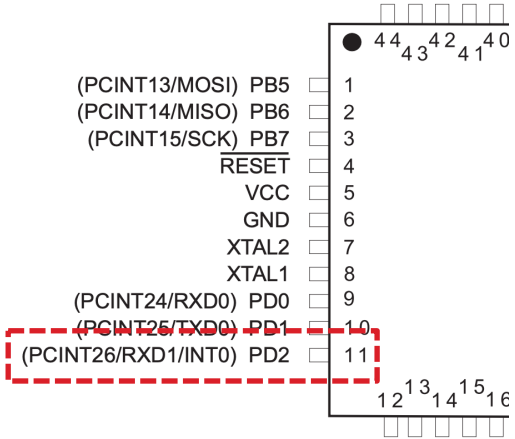
- ATmega324P are 3 linii de întrerupere externă (INT0, INT1 și INT2)
- Orice periferic extern conectat la una din aceste linii poate genera o întrerupere prin schimbarea valorii logice a pinului
  - Chiar și un buton

Exemplu: întreruperi externe la ATmega324P

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any edge of INTn generates asynchronously an interrupt request.
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	INT2	INT1	IINT0	EIMSK
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



- **Bits 2:0 – INT2:0: External Interrupt Request 2 - 0 Enable**

When an INT2:0 bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Register, EICRA, defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

Exemplu: întreruperi externe la ATmega324P

- Probabil cel mai simplu exemplu de cod cu întreruperi
- ISR se execută atunci când pinul INT0 are valoarea 0
- sei() setează bitul I din SREG
  - cli() îl dezactivează
- Bucla principală ciclează la infinit

```
#include <avr/io.h>
#include <avr/interrupt.h>

ISR(INT0_vect)
{
    PORTC ^= (1 << PC0);
}

int main()
{
    DDRC |= (1 << PC0);
    EIMSK |= (1 << INT0);
    sei();

    while(1);

}
```

Rezultatul compilării

```
00000000 <__vectors>:
__vectors():
0: 0c 94 3e 00 jmp 0x7c  ; 0x7c <__ctors_end>
4: 0c 94 48 00 jmp 0x94  ; 0x94 <__vector_1>
8: 0c 94 48 00 jmp 0x90  ; 0x90 <__bad_interrupt>
c: 0c 94 48 00 jmp 0x90  ; 0x90 <__bad_interrupt>
10: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
14: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
18: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
1c: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
20: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
24: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
28: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>

.............................................

60: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
64: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
68: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
6c: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
70: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
74: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
78: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
0000007c <__ctors_end>:
```

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	PCINT0	Pin Change Interrupt Request 0
6	$000A	PCINT1	Pin Change Interrupt Request 1
7	$000C	PCINT2	Pin Change Interrupt Request 2
8	$000E	PCINT3	Pin Change Interrupt Request 3
9	$0010	WDT	Watchdog Time-out Interrupt
10	$0012	TIMER2_COMPA	Timer/Counter2 Compare Match A
11	$0014	TIMER2_COMPB	Timer/Counter2 Compare Match B
12	$0016	TIMER2_OVF	Timer/Counter2 Overflow
13	$0018	TIMER1_CAPT	Timer/Counter1 Capture Event
14	$001A	TIMER1_COMPA	Timer/Counter1 Compare Match A
15	$001C	TIMER1_COMPB	Timer/Counter1 Compare Match B
16	$001E	TIMER1_OVF	Timer/Counter1 Overflow
17	$0020	TIMER0_COMPA	Timer/Counter0 Compare Match A
18	$0022	TIMER0_COMPB	Timer/Counter0 Compare match B

Rezultatul compilării

```
__trampolines_start():
   7c: 11 24           eor r1, r1    ; r1 = 0
   7e: 1f be           out 0x3f, r1  ; SREG = r1
   80: cf ef           ldi r28, 0xFF ; 255
   82: d8 e0           ldi r29, 0x08 ; 8
   84: de bf           out 0x3e, r29 ; SPH = 0x8
   86: cd bf           out 0x3d, r28 ; SPL = 0xFF
   88: 0e 94 4a 00     call 0xb8     ; 0xb8 <main>
   8c: 0c 94 59 00     jmp 0xc0      ; 0xc0 <_exit>

<__bad_interrupt>: __vector_22():
   90: 0c 94 00 00     jmp 0 ; 0x0 <__vectors>
```

0x7c e adresa la care programul sare la RESET

Inițializare stivă. Stack pointer poziționat pe ultima adresă din RAM (0x08FF pentru ATmega324P)

Apel la rutina main()

Orice întrerupere generează un RESET

Unde găsim ce registre sunt la adresele 0x3f, 0x3e, 0x3d etc.? În datasheet:

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	18
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	19
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	19

Rezultatul compilării

```
__vector_1():
#include <avr/io.h>
#include <avr/interrupt.h>
ISR(INT0_vect){
  94: 1f 92 push r1       ; salvare context
  96: 0f 92 push r0
  98: 0f b6 in r0, 0x3f   ; r0 = SREG
  9a: 0f 92 push r0       ; salvare SREG
  9c: 11 24 eor r1, r1
  9e: 8f 93 push r24
  a0: 9f 93 push r25
   PORTC ^= (1<<PC0);
  a2: 88 b1 in r24, 0x08  ; r24 = PORTC
  a4: 91 e0 ldi r25, 0x01 ;
  a6: 89 27 eor r24, r25
  a8: 88 b9 out 0x08, r24 ; PORTC = r24
}
  aa: 9f 91 pop r25       ; refacere context
  ac: 8f 91 pop r24
  ae: 0f 90 pop r0
  b0: 0f be out 0x3f, r0  ; SREG = r0
  b2: 0f 90 pop r0
  b4: 1f 90 pop r1
  b6: 18 95 reti          ; return from interrupt
```

Conținutul SREG este pus pe stivă

Conținutul ISR specificat în codul C (ce vede programatorul)

Reface SREG

Reface PC (pop mascat din stivă)

Rezultatul compilării

```
int main()
{
   DDRC |= (1 << PC0);
b8: 38 9a sbi 0x07, 0    ; set bit 0 DDRC
   EIMSK |= (1 << INT0);
ba: e8 9a sbi 0x1d, 0    ; set bit 0 EIMSK
   sei();
bc: 78 94 sei
be: ff cf rjmp .-2       ; 0xbe <main+0x6>

000000c0 <_exit>:
exit():
c0: f8 94 cli

000000c2 <__stop_program>:
__stop_program():
c2: ff cf rjmp .-2 ; 0xc2 <__stop_program>
```

Set Enable Interrupt (setează bitul I din SREG)

Sare înapoi 2 adrese (la adresa 0xbe) - echivalent cu while(1)

Clear global interrupt flag

ISR Attributes

Sintaxă: `ISR(vector, attributes)`, unde `attributes` poate să fie:

- **ISR_BLOCK** – are același efect ca un call simplu `ISR(vector)` – la intrarea în execuție a codului din ISR se dezactivează orice altă întrerupere
- **ISR_NOBLOCK** – ISR rulează cu întreruperile activate. Permite execuția de nested interrupts (întrerupere în întrerupere)
  - Recomandat doar dacă știți foarte bine ce faceți, execuția nested interrupts poate să conducă la supraîncărcarea stivei și execuție eronată!
- **ISR_NAKED** – nu se mai face automat salvarea și refacerea contextului la intrarea/ieșirea din ISR. Este la latitudinea programatorului să facă acești pași în cod.
- **ISR_ALIASOF** – ISR e legat de alt ISR. Ajută dacă vreți să rulați aceeași rutină de tratare pentru două întreruperi din surse diferite

https://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

Alte aspecte

- Puteți vedea un ISR ca un fir de execuție care rulează independent de main()
- Dacă vrem să comunicăm între un ISR și main() trebuie să apelăm la variabile globale
- *volatile* instruiește compilatorul să nu optimizeze pentru acea variabilă
- Ce se întâmplă dacă nu declaram *var* ca *volatile*?

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>

volatile int var = 0;

ISR(INT0_vect)
{
    var = 1;
}

......

int main()
{
    DDRC |= (1 << PC0);

    EIMSK |= (1 << INT0);

    initUSART();
    stdout = &mystdout;

    sei();

    while(1)
    {
        if(var){
            printf("Button pressed! \n");
        }
    }

}
```

Pin Change INTerrupt (PCINT)

- La ATmega324P, orice pin de GPIO poate genera o întrerupere
- Patru vectori alocați pentru PCINT
  - PCINT0: PCINT7..0
  - PCINT1: PCINT15..8
  - PCINT2: PCINT23..16
  - PCINT3: PCINT31..24

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	PCINT0	Pin Change Interrupt Request 0
6	$000A	PCINT1	Pin Change Interrupt Request 1
7	$000C	PCINT2	Pin Change Interrupt Request 2
8	$000E	PCINT3	Pin Change Interrupt Request 3
9	$0010	WDT	Watchdog Time-out Interrupt
10	$0012	TIMER2_COMPA	Timer/Counter2 Compare Match A
11	$0014	TIMER2_COMPB	Timer/Counter2 Compare Match B
12	$0016	TIMER2_OVF	Timer/Counter2 Overflow
13	$0018	TIMER1_CAPT	Timer/Counter1 Capture Event
14	$001A	TIMER1_COMPA	Timer/Counter1 Compare Match A
15	$001C	TIMER1_COMPB	Timer/Counter1 Compare Match B
16	$001E	TIMER1_OVF	Timer/Counter1 Overflow
17	$0020	TIMER0_COMPA	Timer/Counter0 Compare Match A
18	$0022	TIMER0_COMPB	Timer/Counter0 Compare match B

Activare PCINT

- Fiecare din cei 4 vectori activat din bitul corespunzător din PCICR
- Întreruperile individuale activate din cele patru registre PCMSK

PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	PCIE3	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK3 – Pin Change Mask Register 3

Bit	7	6	5	4	3	2	1	0	
(0x73)	PCINT31	PCINT30	PCINT29	PCINT28	PCINT27	PCINT26	PCINT25	PCINT24	PCMSK3
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	

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

PCINT Code

- Butonul este conectat la PD6 și corespunde PCINT30
- Observați folosirea *volatile* și a lui *ATOMIC_BLOCK*
  - Ultimul probabil nu e necesar aici din cauza pauzei de debounce, dar devine util când pot apărea întreruperi repetate

https://www.nongnu.org/avr-libc/user-manual/group__util__atomic.html

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/atomic.h>
#include <util/delay.h>
#include <stdbool.h>
/*
 * A global flag used to communicate between the Interrupt Service Routine
 * and the main program.  It has to be declared volatile or the compiler
 *  might optimize it out.
 */
volatile bool update = false;

ISR(PCINT0_vect) { /* set update on a high edge */
    if (!(PIND & (1 << PD6))) update = true;
    _delay_ms(300); // Giant friggin' debounce delay
}

int main(void) {

    DDRC = (1 << PC0); /* Using PC0 as LED output*/
    PORTC = 0x00;

    DDRD &= ~(1 << PD6); /* PD6 as input, pull-up activated */
    PORTD |= (1 << PD6);

    PCICR |= _BV(PCIE3); /* Pin Change Interrupt enable on PCINT30 (PD6)*/
    PCMSK3 |= _BV(PCINT30);

    sei();

    while(1){
        ATOMIC_BLOCK(ATOMIC_FORCEON) { /* This turns interrupts off for the code inside it. */
            if (update) {
                PORTC ^= (1 << PC0); /* Toggle LED */
                /*
                 * We reset the update flag to false to indicate that
                 * we are done.  This ensures that this block will not
                 * be executed until update is set to true again, which
                 * is only done by the interrupt service routine.
                 */
                update = false;
            }
        }
    }
}
```


ONE DOES NOT SIMPLY
Use delays in an ISR

Multithreading

- Se poate face multithreading pe AVR?
  - DA! (dar sunt limitări)
- Un exemplu de multithreading cooperativ în codul din dreapta
  - Fiecare thread rulează, apoi cedează execuția altui thread
  - Implementarea este constrânsă de dimensiunea fizică a stivei

https://github.com/phkeese/avr-threads

```
THREAD workerThread(void) {
    while (1) {
        work();
        Threads::yield();
    }
}

THREAD blinkerThread(void) {
    while (1) {
        blinkLED();
        Threads::yield();
    }
}

int main(void) {
    Threads::init(128);
    Threads::createThread(workerThread);
    Threads::createThread(blinkerThread);

    while (1) {
        prepareWork();
        Threads::yield();
        outputWork();
    }
}
```

Multithreading

- La fiecare *yield()* se salvează contextul thread-ului curent (în cazul de față push în stivă pentru toate reg. generale și SREG), apoi se reface contextul pentru threadul următor

```
void yield() {
        SM_SAVE_CONTEXT()

        // Save stack of current thread
        currentThread->stackptr = SP;

        // Switch threads
        currentThread = currentThread->next;

        // Restore stack of currentThread
        // As this is a critical 16 bit value
        // we cannot let interrupts occur
        asm("cli");
        SP = currentThread->stackptr;

        SM_RESTORE_CONTEXT()
    }
```