Al crear-se una interrupció (flag activat), el flag del registre d'estats desactiva les interrupcions globals, per tant no serán podran activar més fins que no surti de la subrutina

VDCO buit estarà activat quan no estàs escrivint res

Quan fas veti -> fas que al sortir de la subrutina d'interrupció -> estarà activar el flag d'interr. global

si al mig d'execució d'una subrutina d'interrupció, paral·lelament es genera una altra, mentre s'executi aquesta la altra no no serà, ja que el flag d'interrupcions està desactivat. Al sortir de la primera subrutina, d'interrupció, començarà la següent interrupció.

PORT_B

in r16 → xx1x xxxx

lac 7

PortB.

PortB → xx1x xxxx

just a l'inici a l'interrupció el flag d'interrupció es desactiva.

SREG -> bit 7 (I) -> les activa o desactiva totes
  └> SEI (set enable interrupt)
  └> CEI (clear " ")

Interrupcions -> la edició de poder treballar en paral·lel

No hi ha call, però el hardware podueix la crida. Aleshores s'interromp el programa i va (jmp) a la subrutina d'interrupció

En aquestes subrutines encara és molt més important la transparència ja que no sabem quan es cridarà

El hard sap l'adreça de la subr. d'interrupció?

Mecanisme del vector d'interrupció

si cliquem reset -> anem a PC=0

Al ser interrupció 18 -> USART RX anem a adreça 0x0024 -> que és on està guardada la subrutina

cada 2 words hi tenim interrupcions: cada interrupció del vector -> és un jMP on està guardada la subrutina d'interrupció 2 words -> 1 codifica que és un jMP, el segon -> 16 bits que és la llargada de la memòria de programa. Fem el salt per sobre

A on posi _vector_ IS → guarda aquella adreça @ del per surti

al activar-se la interrupció va a la posició

0x0024 on hi ha un JMP, @

si a la rutina d'interrupció sen generen 2 més, en aquell moment no s'executen ja que el bit 7 de SREG està desactivat, ~~es~~ al sortir de la subrutina, s'executarà la darrera inserir primer, després l'altre ja que el flag es manté activat sins que no s'entra a la subrutina corresponent

bootrst no es pot enviar amb el cable USB

si no, no podriem anar programant ja que no arribem al boot loader

Si una subrutina fa operacions Aritmitico-lògiques, cal protegir SREG!

rx -> no fi parametres, resultat = r16

fx -> parametre r16, no fa resultat.

Haurem de posar un ret a final de main -> ara ens ho ofrenem com una subrutina

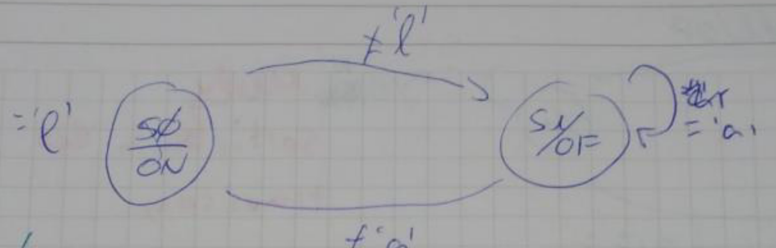
bad_interrupt -> predefinit fent resets

```
eor r1,r1      netejem resistre
out, 0x3F,r1   posem a 0 el resistre d'octet
ldi r28,0xFF   } inicialitza stack pointer
ldi r29,0x08   |   posant 0x08FF
out 0x3d,r28   |      ↓
out 0x3e,r29   }   última posició física de dades

call main
jmp exit
      ↳ cli

rjmp -2
```

és per si el nostre programa main no acaba, llavors acaba aquí



Moore
↓
Sortides als estats!

He de ser subrutines que tinguin sentit per si mateixes.

```
LEDON:
  sbi PORTB0, 5
  ret
```

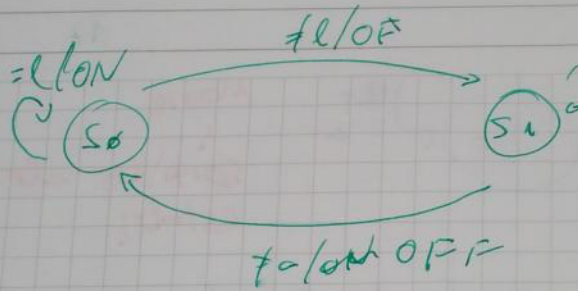
Massa línies per la línia de programa.

Ho faig com una macro → .macro ledon
sbi PORTB, 5
.endm

```
S0:
  ledon
  :
  call rx
  if rx ≠ l
     RJMP S1
else: RJMP S0

S1:
  led off
  call rx
  if rx ≠ a        → cpi r16,'l'
     RJMP S0         brne S1
  RJMP S1
```

estructura → al final de les inicialitzacions (main) poso RJMP A S0. Sino, no ho poso i escric primer S0.



Mealy
Sortides a les transicions

S0:
call rx
is Rx = 'l'
led on
rjmp S0
else:
led off
rjmp S1

S1:
call rx
is rx = 'o'
led on
rjmp S1
else:
led off
rjmp S0

AMB INTERRUPCIONS

- Vector-RS:

LDS r16, VORO copiem el codi de comparació en comptes de ser un rjmp a ell
is estat = 0
RJMP S0 / comprovar estat
is estat = 1
RJMP S1 / comprovar estat

OPERACIONS ARITMÈTIQUES

Resta -> Suma de A + complement A2 de B

$Ca2(Dada) = Ca1(Dada) + 1 = (Dada \oplus 0xFF) + 1$

```
  010   2          010   2
- 101   5        + 011  -5
                 -------
                   101  -3
                 0
                 ↑
                 C
```

```
  110   6          110   6
- 011   3    →   + 101  -3
                 -------
                 1 011  3
                 ↑
                 C
```

Negarem el carry per tant -3 -> 1101

3 -> 0011

1 byte té dos nibbles:

1010 : 0110

SREG: T, I | Z, C, N, V, H, S

Z: zero resultat; C: carry; N: negatiu; V: overflow; S: signe; H: half carry

OVERFLOW V → considerem dades amb signe

és el que ens diu si ens hem passat del valor de la taula

SUMA → dos operands (+)
resultat (=)

en Ca2, el carry no té significat

$S = N \oplus V$

el bit de més pes és l'S, no el carry

```
  011  3          010  2
+ 010  2          001  1
-------          -------
  101  5          0011  3
S V              S V
0 1
```

```
  100  -4          111  -1
  010   2         1100  -4
-------          -------
 1 0 1 1 0        11 011
 S V              SV
```

SUMA AMB SIGNED

OVERFLOW → quan la suma de dos positius dóna negatiu i al revés.

FLAG $S = N \oplus V$ (S: signe, N: negatiu, V: overflow)

S'indica el resultat total i correcte de la suma

Ara el resultat té 4 bits

RESTA AMB SIGNED

el mateix que a la suma, però abans fem un ~~overflo~~ canvi a ca2

Si restem dos positius o dos negatius, mai podrà donar overflow

També codifiquem el total amb el resultat i el bit S

$$\begin{array}{r} 010_2 \\ -100 \quad -4 \end{array} \rightarrow \begin{array}{r} 010_2 \\ +100_{,4} \\ \hline 01\,10\ 6 \end{array}$$

S V

S V V

es fa una XOR per trobar S

$$\begin{array}{r} 010_2 \\ -011\ 3 \end{array} = \begin{array}{r} 010_2 \\ +101\ -3 \\ \hline 10\,111\ -1 \end{array}$$

S V

BREQ → si el flag Z = 1, salta k posicions

BRNE → si el flag Z = 0 salta k posicions

si ~~ab~~ la instrucció prèvia no afecta als flags, es mantindrà el flag passat. Exemple: fais operació i després una NOP. el flag no canvia a la NOP.

BRBS → si el resistre d'estat, el bit s està activat, saltaré k posicions

BRBS s, k if SREG(s) = 1 then PC ← PC + k + 1

Un breq té el mateix opcode que brbs però amb s = 001 → el bit ~~s~~ Z és l'1

BRBC → si el bit s del SREG està desactivat

BRBC s, k

BRMI → branch if minus → si N = 1

BRSH → branch if same or higher UNSIGNED

" Es resten els valors i miren el carry

BRCC CP → COMPARE sense actualitzar el resultat.

Un sub ~~fals~~ ons variaria els resultat

Mirem si C = 0

BRLO/BRCS → si és lower. el complementary, aquí

Mirem si C = 1

També

BRLO* → si c+z=0 ja que és estrictament més petit

BRSH* → Rd ≤ Rr → c+z=1

si Rd és més petit activa el carry
si són iguals s'activa el z

Realment no es fan amb aquests dos *. Ja que no es poden accedir a dos flags alhora. Utilitzen els de la pàgina anterior!

Només tenim BRLO i BRSH*, per tant nosaltres hem de fer les combinacions i sigui l'ordre dels registres en funció del que volem fer

SIGNED

BRGE Rd ≥ Rr Greater equal

BRLT Rd < Rr less than zero

Realment en les operacions aritmetico lògiques també actualitzen la 's', tot i que no surti a la taula d'instruccions

El CP també canvia la 'S'