interrupts

connection between CPU and I/O devices

- all i/o devices are different in speed, analog /digital..., and need different tasks: buffering, controlling..

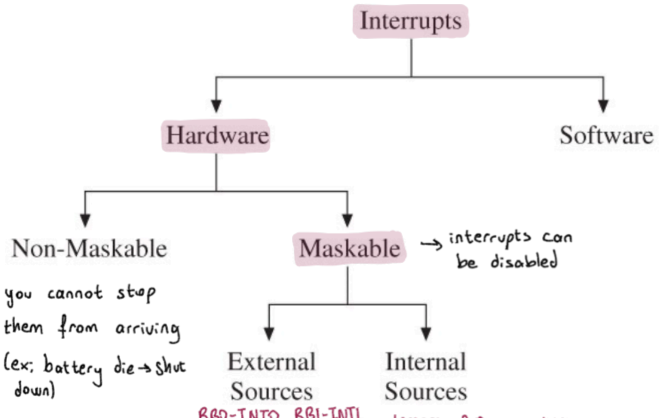
↳ programmed i/o = the status of i/o device is repeatedly checked by the program (polling=seçmek) → wastefull
↳ synchronous

↳ interrupt i/o = the i/o device request the interrupt, then interrupt handling /service rutine

interrupt = temporary break in the flow of execution of a program (asynchronous = CPU does not know when it will arrive)

↳ interrupt service routine (ISR) = deals with them, then continue where it was left off (MPU → micro processor unit in PIC)

↳ they normally handled by OS, but in embedded system, there is no OS → needs to be programmed



→ interrupts can be disabled

you cannot stop them from arriving (ex: battery die → shut down)

RB0-INT0, RB1-INT1, RB2-INT2, [4,7] PORTB ↳ RBIF

timers, A/D converters, serial i/o...

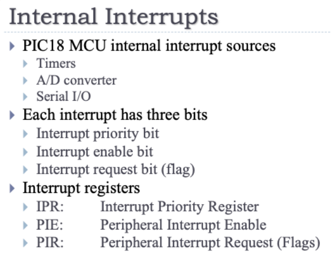
- in PIC all interrupts are hardware, maskable
- intel has some sofware too
- interrupts are disabled by default in PIC

① MPU checks interrupt request flag at the end of each inst
→ set by output device

② if present reset the flag, save return address on the stack (finishes current instruction execution)

③ redirect to interrupt vectors, ISR meets request, MPU returns back

3 interrupt flags in PIC = enable / disable, request (flag), high priority / low priority
→ set by programer
IE
IF → set by output device



Internal Interrupts

- PIC18 MCU internal interrupt sources
  - Timers
  - A/D converter
  - Serial I/O
- Each interrupt has three bits
  - Interrupt priority bit
  - Interrupt enable bit
  - Interrupt request bit (flag)
- Interrupt registers
  - IPR: Interrupt Priority Register
  - PIE: Peripheral Interrupt Enable
  - PIR: Peripheral Interrupt Request (Flags)

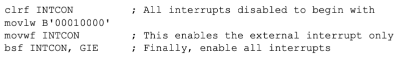
special function registers → INTCON (interrupt control), RCON (priority enable), IPR-PIE-PIR (internal peripheral int)

GIE = global interrupt enable (7th bit of INTCON), if it is 0, no interrupts are enabled

RCON = reset control, if 7th bit 0, no priority, all are high as default

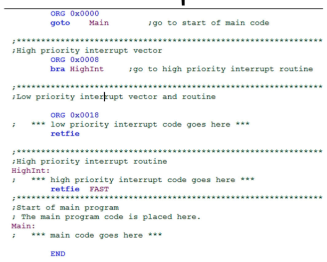
✱ pins must be defined as digital inputs to use as interrupt flag

initialization =



```
clrf INTCON          ; All interrupts disabled to begin with
movlw B'00010000'
movwf INTCON         ; This enables the external interrupt only
bsf INTCON, GIE      ; Finally, enable all interrupts
```

- high priority interrupts are automatically saved into registers, low ones must be saved by the programmer
- enabled interrupts must be checked by programmer, when there is a interrupt, to determine which ones



```
        ORG 0x0000
        goto    Main        ;go to start of main code

;****************************************************************
;High priority interrupt vector
        ORG 0x0008
        bra HighInt     ;go to high priority interrupt routine

;****************************************************************
;Low priority interrupt vector and routine

        ORG 0x0018
;   *** low priority interrupt code goes here ***
        retfie

;****************************************************************
;High priority interrupt routine
HighInt:
;   *** high priority interrupt code goes here ***
        retfie  FAST
;****************************************************************
;Start of main program
; The main program code is placed here.
Main:
;   *** main code goes here ***

        END
```

acknowledging the interrupt = clearing interrupt flag before returning from the ISR

- ISR for one interrupt must be short to not to miss another interrupt
- in basic i/o very short signals can be missed but not in interrupts (you can response late though)
- for different speed communication, buffer is needed

ring buffers = to implement FIFO data, two pointers; one for head one for tail, if full go to beginning