E/S devices resume in MSX88

IMPORTANT REGISTERS

All registers mentioned here are 8 bits each

PIC

20h → EOI (used for telling the PIC an interruption ended)

21h → IMR (¿wich interruptions are allowed? 1 means not allowed, 0 means allowed

24h → INT0 register: corresponds to **F10** key interruption

25h → INT1 register: corresponds to **TIMER** interruption

- **10h →** timer **CONT** register
- **11h →** timer **COMP** register

26h → INT2 register: corresponds to **HANDSHAKE** interruption

Handshake

- **40h →** HANDSHAKE **data** register
- **41h →** HANDSHAKE **state and control** register

27h → INT3 register: corresponds to **CMDA** interruption

PIO

30h → Port A register

32h → Port A configuration register

31h → Port B register

33h → Port B configuration register

USART

60h → DIN input register

61h → DOUT output register

62h → CTRL control register (writing) and state register (reading)

DMAC

50h y 51h → SOURCE direction(high and low) register

52h y 53h → BYTES to transfer

54h y 55h → DESTINATION direction (high and low) register

56h → CONTROL register

57h → START register

PIO CONFIGURATIONS/SUBROUTINES

The *PIO is for comunnication with periferia*

One form of connection has switches (connected to port A) and leds (connected to port B).
The other form has the printer, which uses port A for state and port B for data.

PIO for switches and leds

- We can set all switches as input devices and all leds as output devices.
- This is not a rule but there's not much logic in doing it another way: leds cannot give us any information and we cannot give any information to switches.

```
config_pioKL: mov al, 0FFh ; keys for input
              out 33h, al
              mov al, 00h ; leds for output
              out 32h, al
ret
```

PIO for printer

- Requires manually sending the signal for printing → that is sending a 0 to 1 signal to strobe bit.
- Setting bit 0 of PA as read, and bit 1 as write
- Setting PB as write, for sendig data

```
config_pioPR: in al, 32h ; using 2 masks to avoid altering CA previous state.
              and al, 0FDh
              or al, 01h
              out 32h, al
              mov al, 00h ; CB uses the 8 bits for output.
              out 33h, al
ret
```

Strobes for printer

- Sending to bit 1 in port A a 0 or a 1.

```
; forcing a 0 and avoid altering the rest of bits.
strobe0: in al, 30h
         and al, 0FDh
         out 30h, al
ret
```

```
; forcing a 1 and avoid altering the rest of bits.
strobe1: in al, 30h
         or al, 02h
         out 30h, al
ret
```

Polling with pio

- *Is the only way of using the printer with the PIO device.*

```
; ¿is busy bit 0? if not, ask again
pollPIO: in al, 30h
         and al, 01h
         jnz pollPIO
ret
```

PRINTER WITH HANDSHAKE

With hanshake we have to ways of using the printer: via polling or via interruptions

In the state/control register:

- Bit 1 is strobe, but the handshake sends the signals automatically (basically we don't need it).
  We just need to send data when the printer is not busy (bit 0 of state/control register is 0, similar as using the PIO)
- Bit 7 is the INT bit. A 1 indicates the handshake to produce a interruption when it can receive more data (we also need to allow INT2 in the PIC)

Handshake polling

- Is the same as when we are using the PIO, the only diference is the direction of the register with the busy bit.

```
pollHAND: in al, 41h
          and al, 01h
          jnz pollHAND
ret
```

Handshake without INT

- This setting goes paired with polling, similar as using the PIO.
- The only diference is that the strobe signal is automatically sent when we put data in the 40h register.

```
configHAND: in al, 41h
            and al, 7Fh
            out 41h, al
ret
```

Handshake with INT

- In this case is also needed that we configure the PIC to allow the printer to interrupt the CPU.
- If not, the printer will send the interruptions but they will not be atended.

```
configHAND: in al, 41h
            or al, 80h
            out 41h, al
ret
```

- When we finished using the handshake via interruptions, we can use the NO INT version to stop and also deactivate from the PIC.

PRINTER WITH USART DEVICE

USART device allows the CPU to comunicate with serie devices.

It can generate 2 types of interruptions:

- INT 2 → when there's a character ready to be received.
- INT 3 → when it's ready to send a character

Configuration for DTR protocol

We need to send to the control register (dir 62h) a ***51h → 01010001.***

- Bit 0 indicates asincronic comunication.
- Bit 4 indicates that we are using the DTR protocol.
- Bit 6 indicates error reset.

How to operate

With polling

We need to have in mind:

- Bit 0 → TxRDY, this bit indicates when the USART is ready to transmit (1 means ready)
- Bit 7 → DSR (data set ready), this bit indicates when the printer is ready to receive a caracter.

Example implementation

1. Reads state register.
2. Uses a mask to check if bit 0 = 1.
3. If no, that means that al = 0, TxReady = 0 = USART is not ready.
4. Keeps asking.
5. When the USART is ready, starts to check for the printer
6. Uses a mask to check if bit 7 = 1
7. If no, that means al = 0, DSR = 0 = PRINTER is no ready.

```
USARTisready: in al, 62h
              and al, 01h ; checks TxRDY
              jz USARTisready
```

```
ret

PRINTERisready: in al, 62h
                and al, 80h ; checks DSR
                jz PRINTERisready
ret

SENDdata: mov al, [bx]
          out 61h, al
ret

; we can rewrite all in one subroutine
PRINT: call USARTisready
       call PRINTERisready
       call SENDdata
ret
```

💡 This implementation is clearly incomplete! This are a "base" subroutine. Adapt it to your program (with the parameters/data/control flow of your choice)

Configuration for XON/XOFF protocol

We need to send to the control register (dir 62h) a 4***1h → 01000001.***

- Bit 0 indicates asincronic comunication.
- Bit 4 indicates that we are using the XON/XOFF protocol.
- Bit 6 indicates error reset.

How to operate

With polling

We have to have in mind:

- Bit 0 → TxRDY, this bit indicates when the USART is ready to transmit (1 means ready)
- Bit 1 → RxRDY, this bit indicates we have a caracter to read from the printer.

  → This caracter can be XON = 13h or XOFF = 11h

Example implementation

1. Same as before, we need to first wait until the USART is ready to transmit
2. We can send a caracter.
3. We need to check if we received a caracter from the printer.
    1. Read from state register in dir 62h, from bit 1
    2. If = 1, verify which caracter is
4. If there is a caracter, read it
    1. Read from DIN register in dir 60h
    2. If caracter = XOFF, poll RxReady again until we have another char for reading

5. Repeat until all is printed

```
USARTisready: in al, 62h
              and al, 01h ; checks TxRDY
              jz USARTisready
ret

SENDdata: mov al, [bx]
          out 61h, al
ret

NEEDtoread: in al, 62h
            and al, 02h ; checks RxReady
            jz NEEDtoread
ret

CANsendMore: call NEEDtoread
             in al, 60h
             cmp al, 13h ; if we didnt't received "XON", wait until new char comes
             jnz CANsendMore
ret

PRINT: call USARTisready
       call SENDdata
       call CANsendMore
ret
```

💡 This implementation is clearly incomplete! This are a "base" subroutine. Adapt it to your program (with the parameters/data/control flow of your choice)