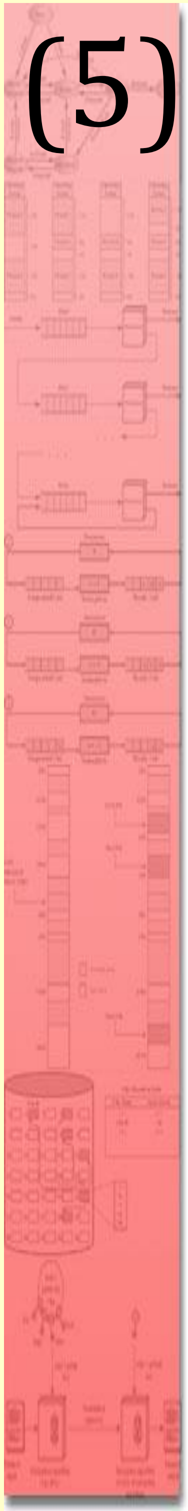


(5)

Escuela Politécnica Superior

UAM
Universidad Autónoma
de Madrid

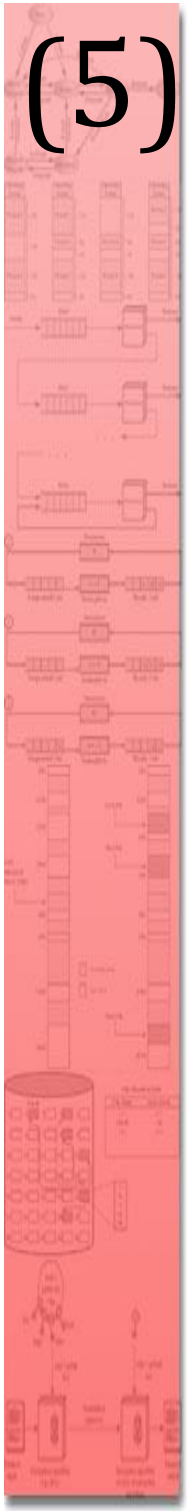
Unit 5
Input / Output

MICROPROCESSOR-BASED SYSTEMS

Degree in Computer Science Engineering
Double Degree in Computer Engineering and Mathematics

EPS - UAM

(5)

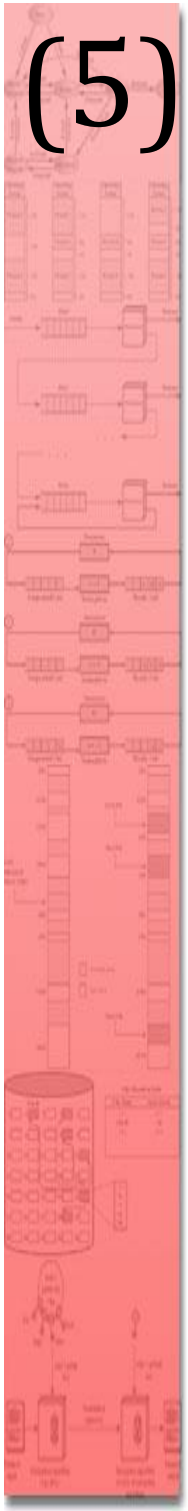


Index

5. Input / Output.

5.1. Input and output programming techniques.
5.2. Polling.
5.3. Interrupts.
5.4. DMA.
5.5. Interrupt management and programming in the 80x86: the 8259A controller.

(5)

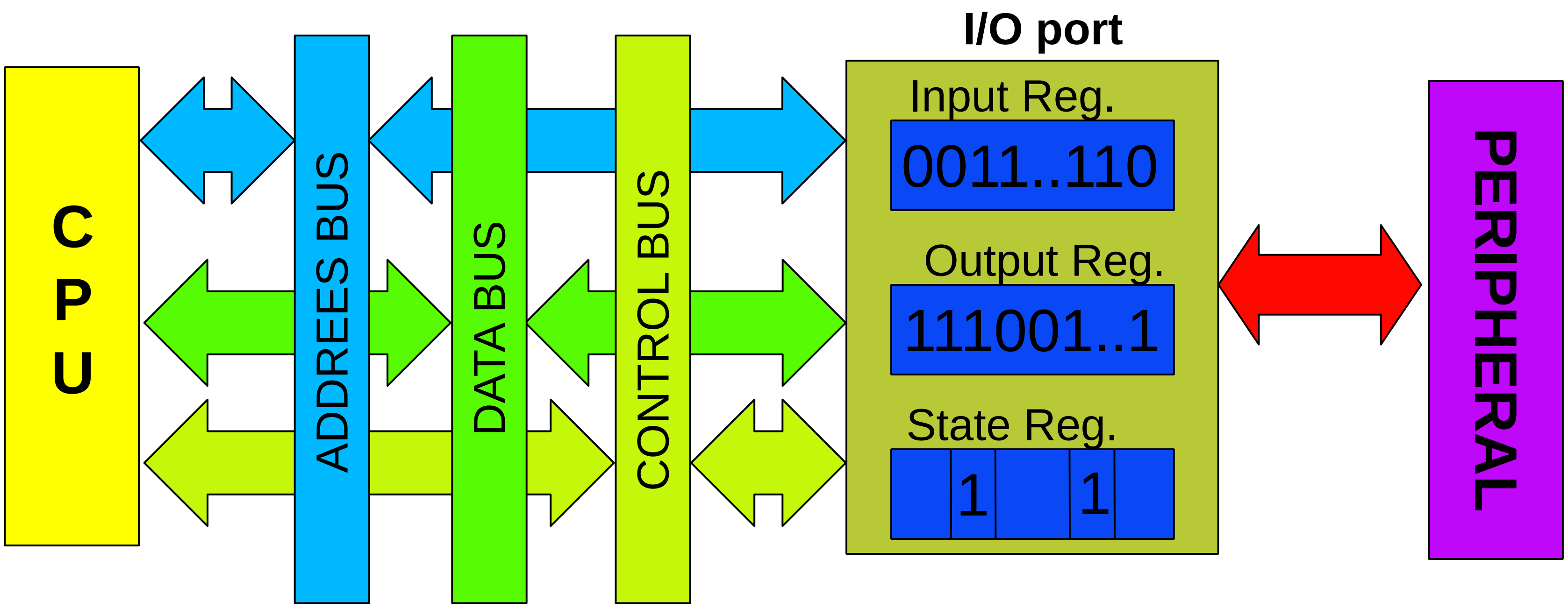


5.1. I/O programming techniques (I)

- A microprocessor-based system must transfer data from (input) and toward (output) external devices.
- Three ways of performing the data transfer:
  - Polling
    - The CPU is responsible for sending and receiving data, and for synchronizing with peripherals (wait for data reception or wait for request of data emission).
  - Interrupts
    - The CPU is responsible for sending and receiving data.
    - Synchronization is performed through hardware interrupts received by the CPU.
  - DMA *(Direct Memory Access)*
    - The CPU configures a DMA controller that sends and receives data.
    - Synchronization is performed by the DMA controller itself through hardware interrupts.
    - Used for transferring data blocks.

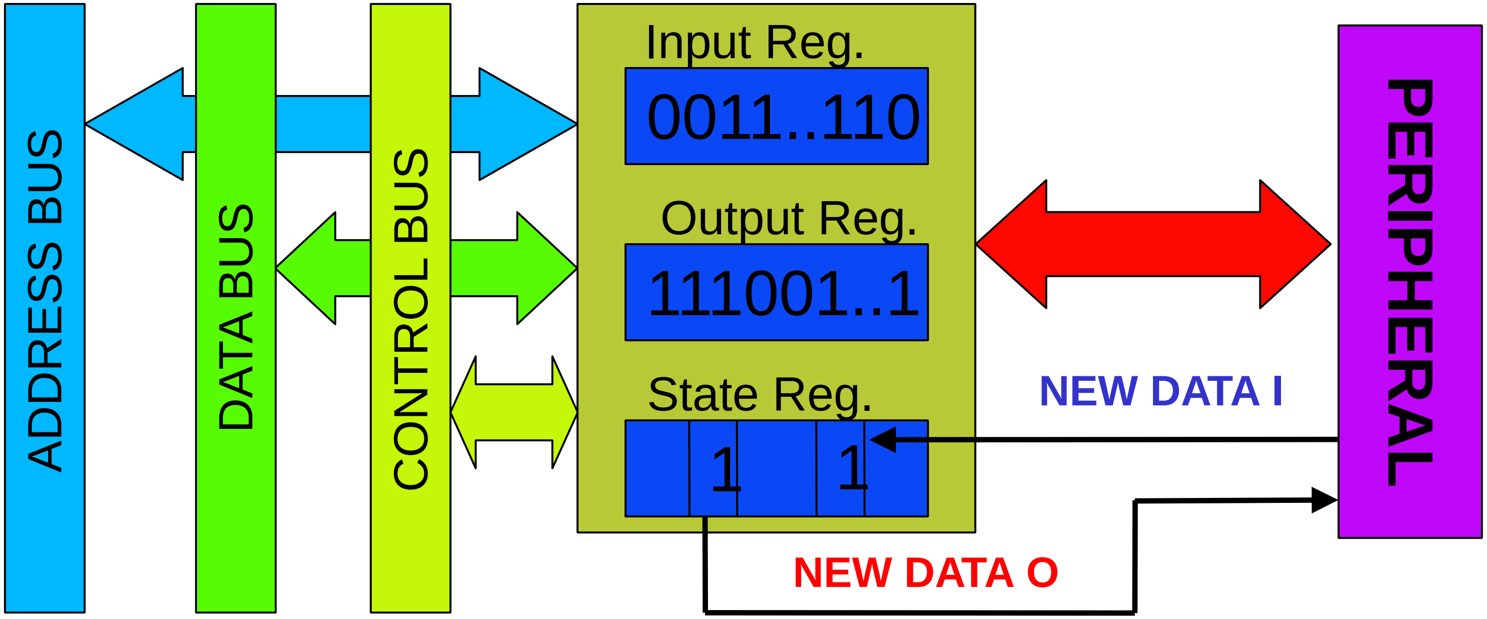
5.1. I/O programming techniques (II)

- Every peripheral is accessed by the CPU through an I/O port (controller), which acts as an interface.
- The CPU usually accesses the ports by reading and writing internal registers (every register has an I/O address):
  - **Input registers**: Data from the peripheral toward the CPU.
  - **Output registers:** Data from the CPU toward the peripheral.
  - **State registers:** Peripheral's current state.



5.2. Polling (I)

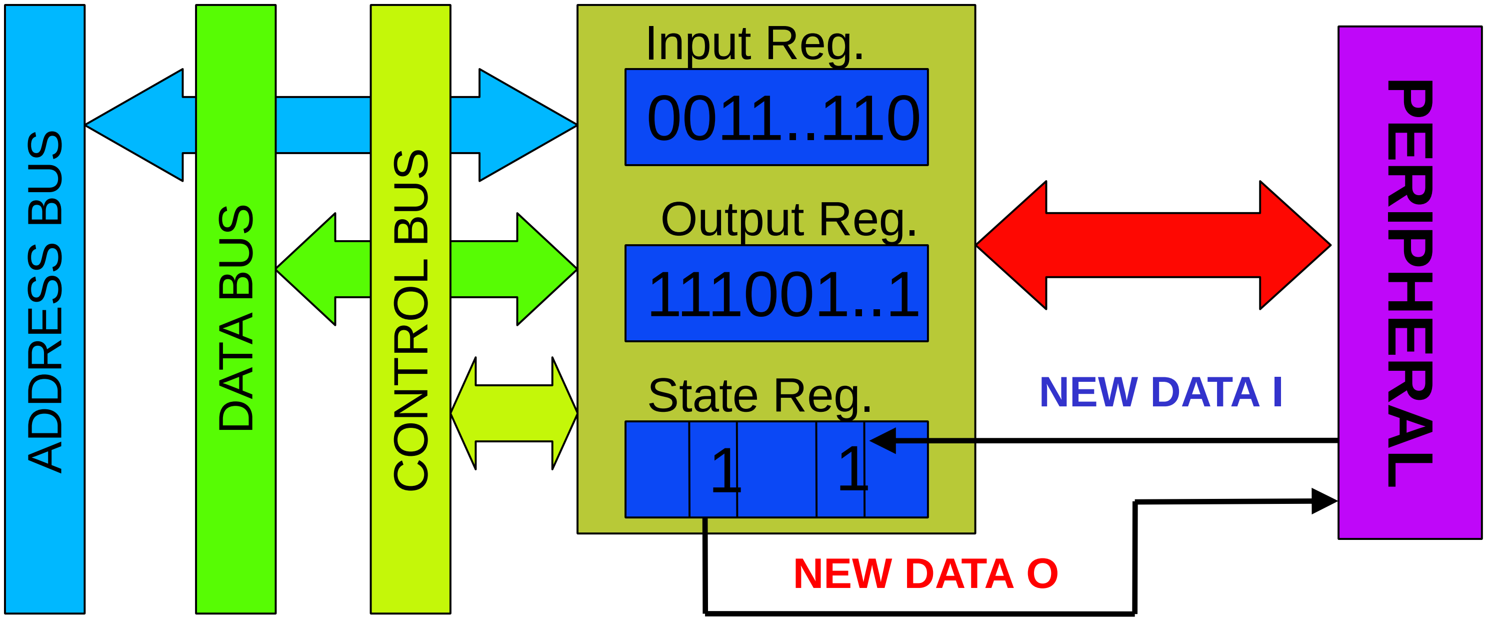
- The CPU must send and receive data, and synchronize with the peripheral.
- Synchronization through active wait: a loop continuously queries the state register (very inefficient).
- Handshaking protocol through two control lines: **NEW DATA I** and **NEW DATA O**.



(5)

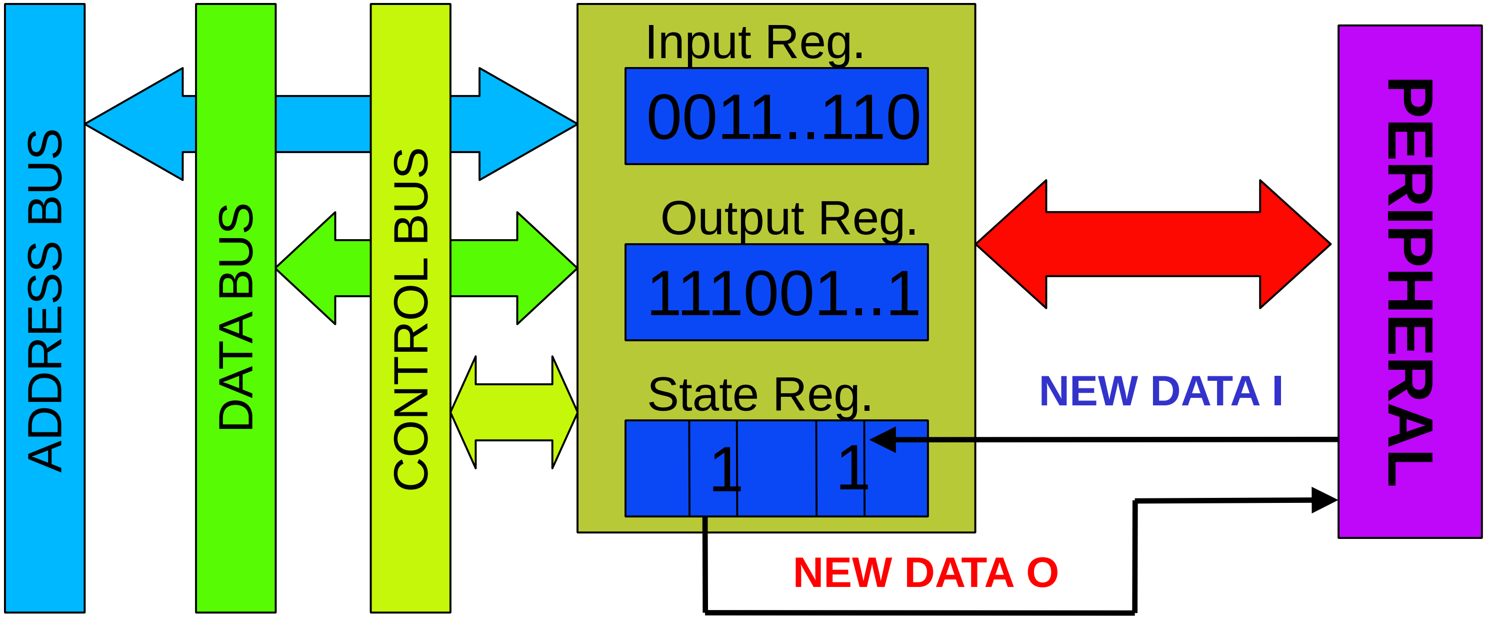
5.2. Polling (II)

- Basic protocol for writing data to the peripheral:
  - CPU writes data to output register.
  - CPU activates signal **NEW DATA O** in state register.
  - CPU waits for activation of signal **NEW DATA I** in state register (active wait).
  - Peripheral receives data from port and activates signal **NEW DATA I** (*Acnowledge, ACK*).

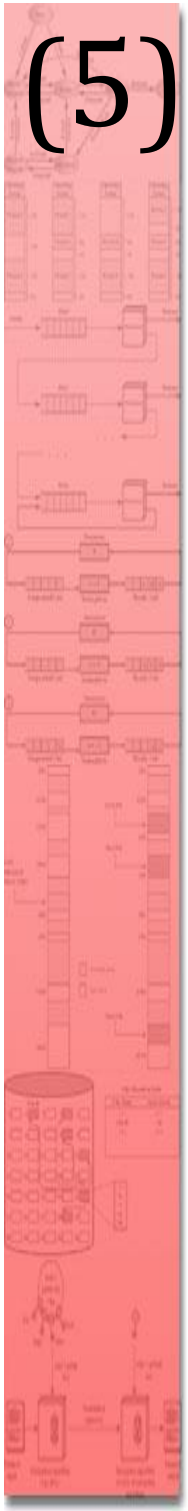


5.2. Polling (III)

- Basic protocol for reading data from peripheral:
  - CPU waits for activation of signal **NEW DATA I** in state register (active wait).
  - Peripheral sends data to port and activates signal **NEW DATA I**.
  - CPU reads data from input register.
  - CPU activates signal **NEW DATA O** in state register.



(5)



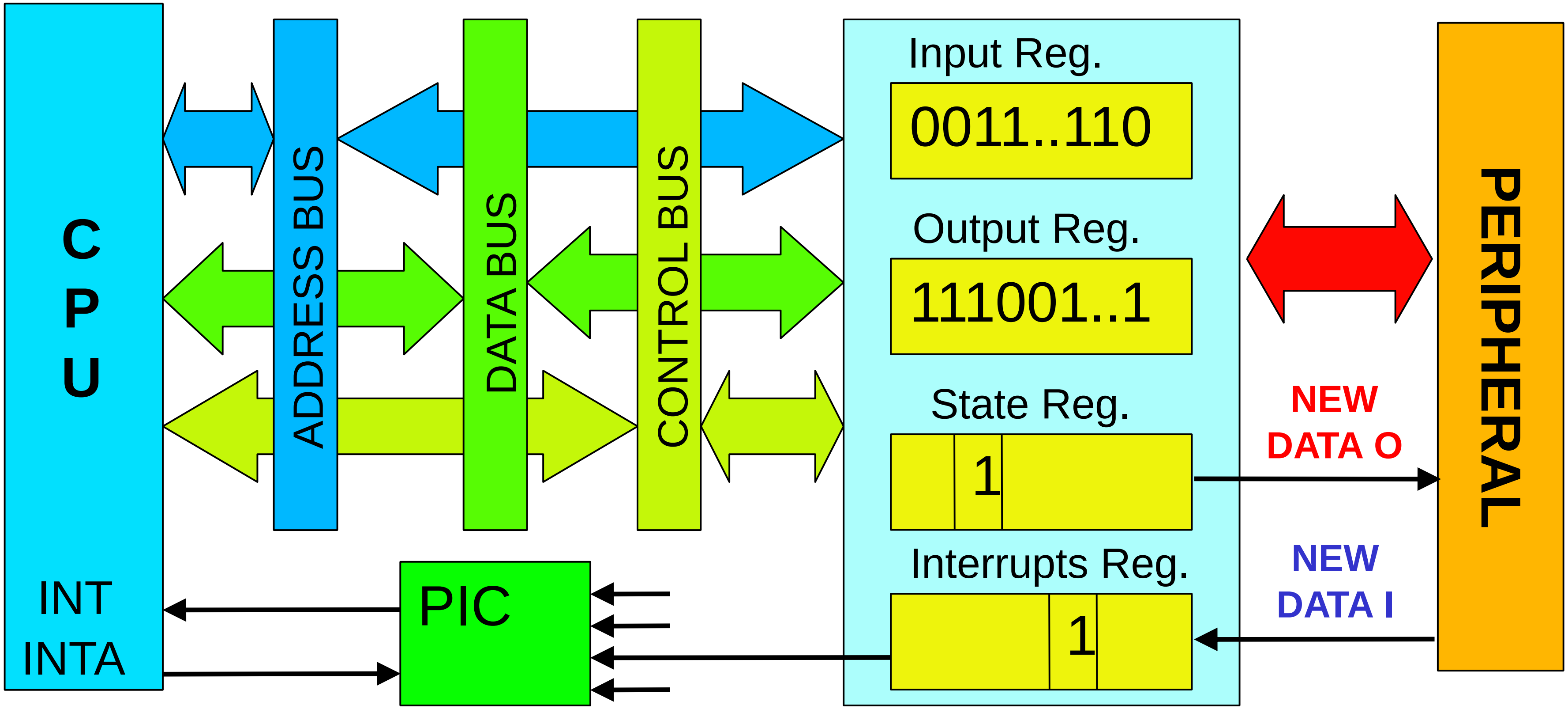
5.2. Polling (IV)

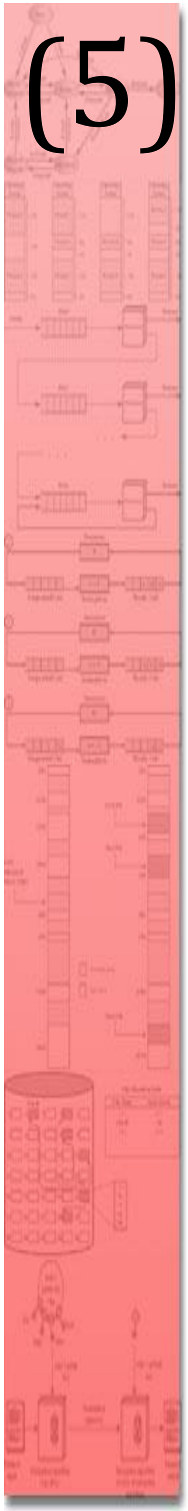
- **Example**: Read data from I/O port, store them into memory buffer and deliver control byte.
- I/O port:
  - 52h ⇒ @Input Reg.
  - 53h ⇒ @Output Reg.
  - 54h ⇒ @State Reg.

```
data    SEGMENT
        buffer  200 DUP (0)
data    ENDS

code    SEGMENT
        .........
        mov ax, data
        mov ds, ax
        mov si, 0

wait:   in al, 54h    ; Active wait
        test al, 00000001b
        jz wait

        in al, 52h        ; Read data
        mov buffer[ si ], al
        inc si
        cmp si, 200
        jne wait

        mov al, 0FFh
        out 53h, al    ; Control byte
```

5.3. Interrupts

- The CPU is interrupted when the peripheral sends data (input) or when it sends a request to receive data (output).
- The CPU executes a service routine that reads the data (input) or sends them through the port’s registers.

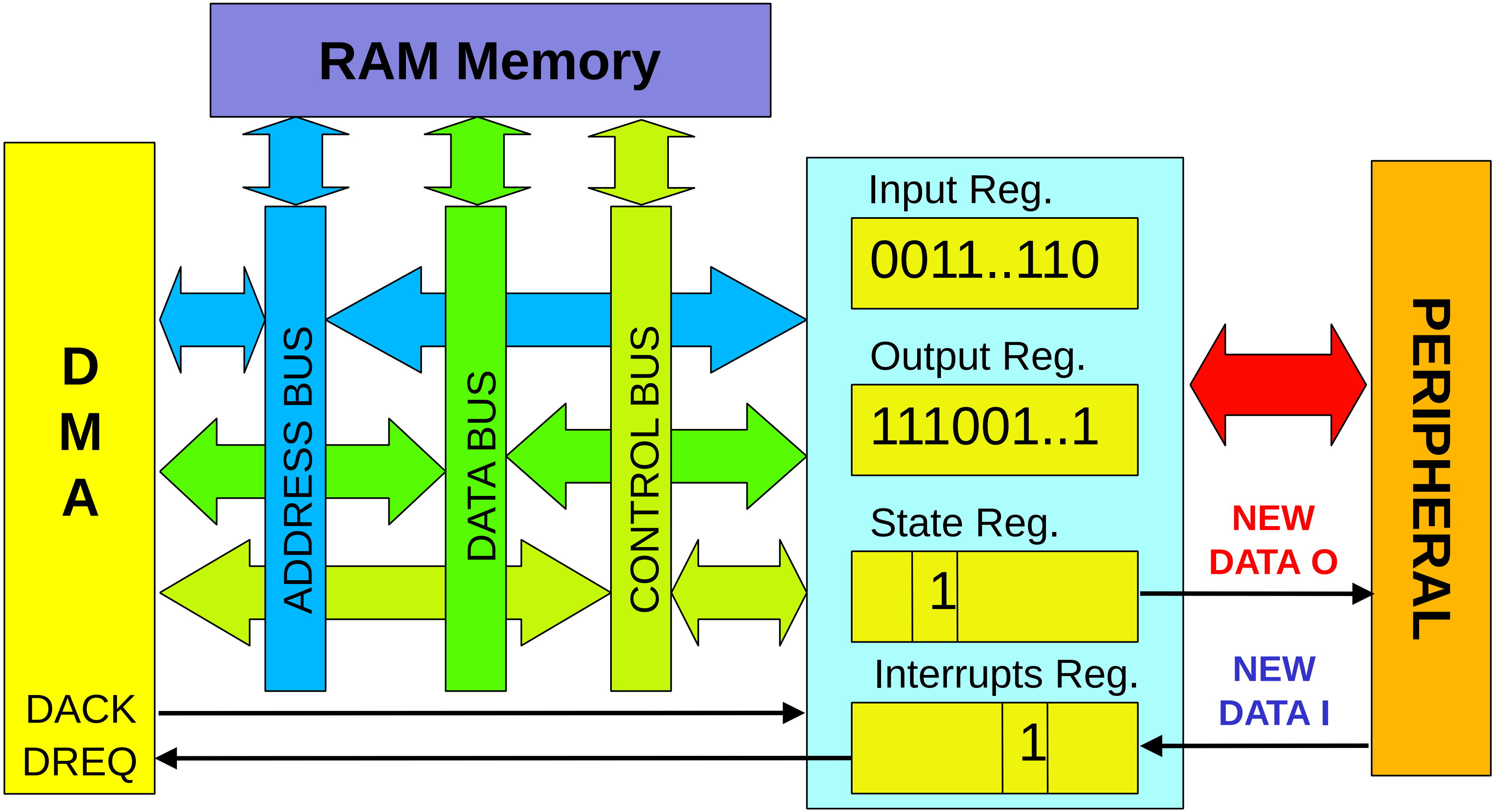




(5)

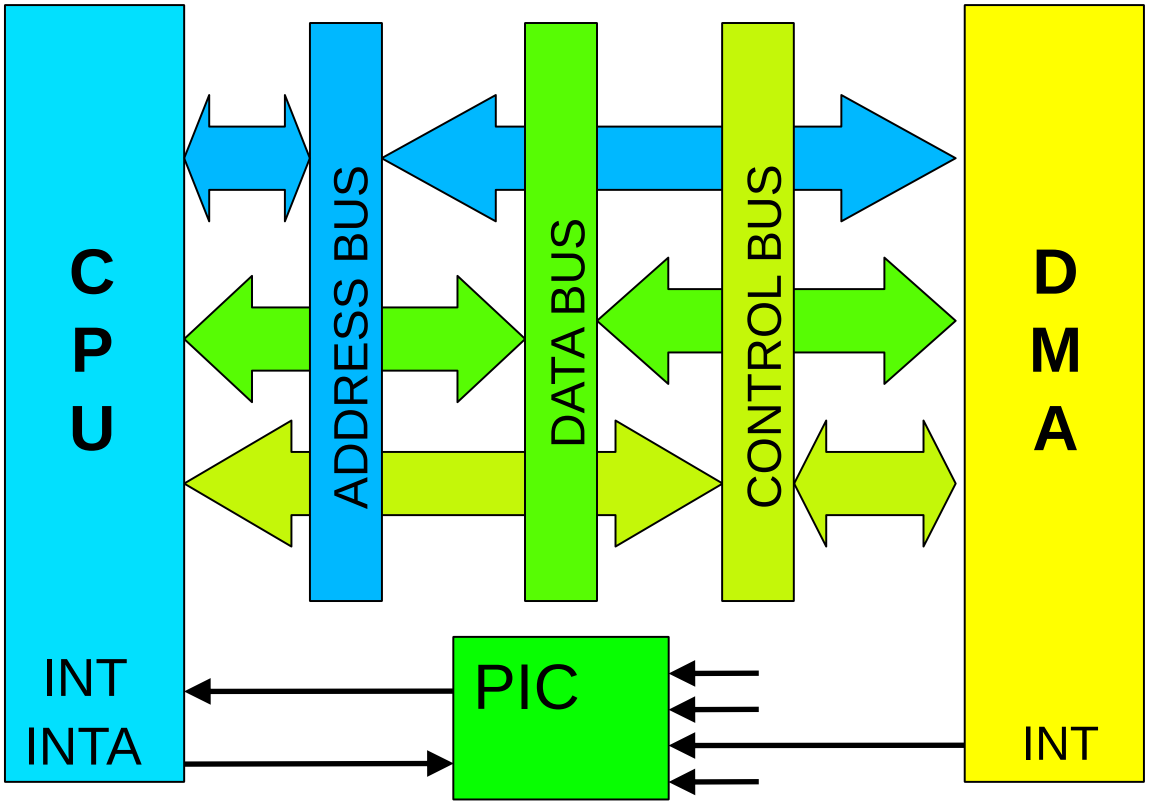
5.4. DMA (I)

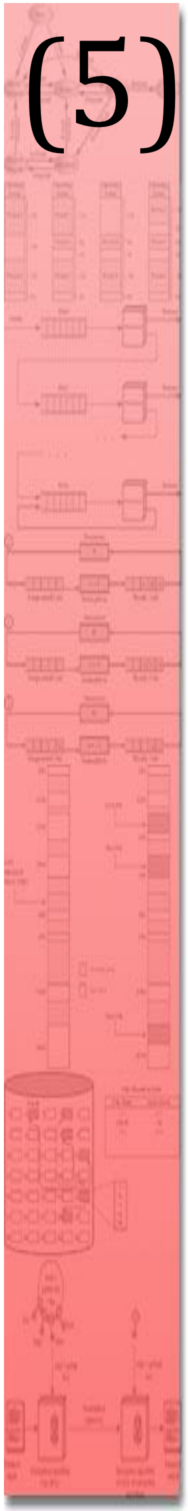
- The CPU programs the DMA controller to transfer a data block from memory to the I/O port (output) or from the port to memory (input).



5.4. DMA (II)

- The DMA controller interrupts the CPU after a complete block has been transferred.
- The CPU executes a service routine that processes the block (input) or generates a new block (output) and then reprograms the DMA controller.

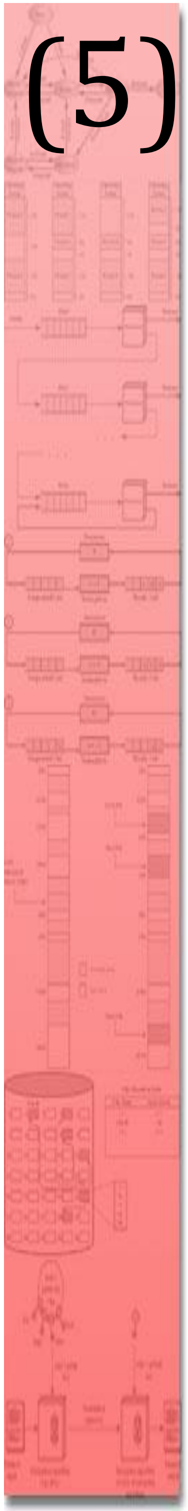




(5)

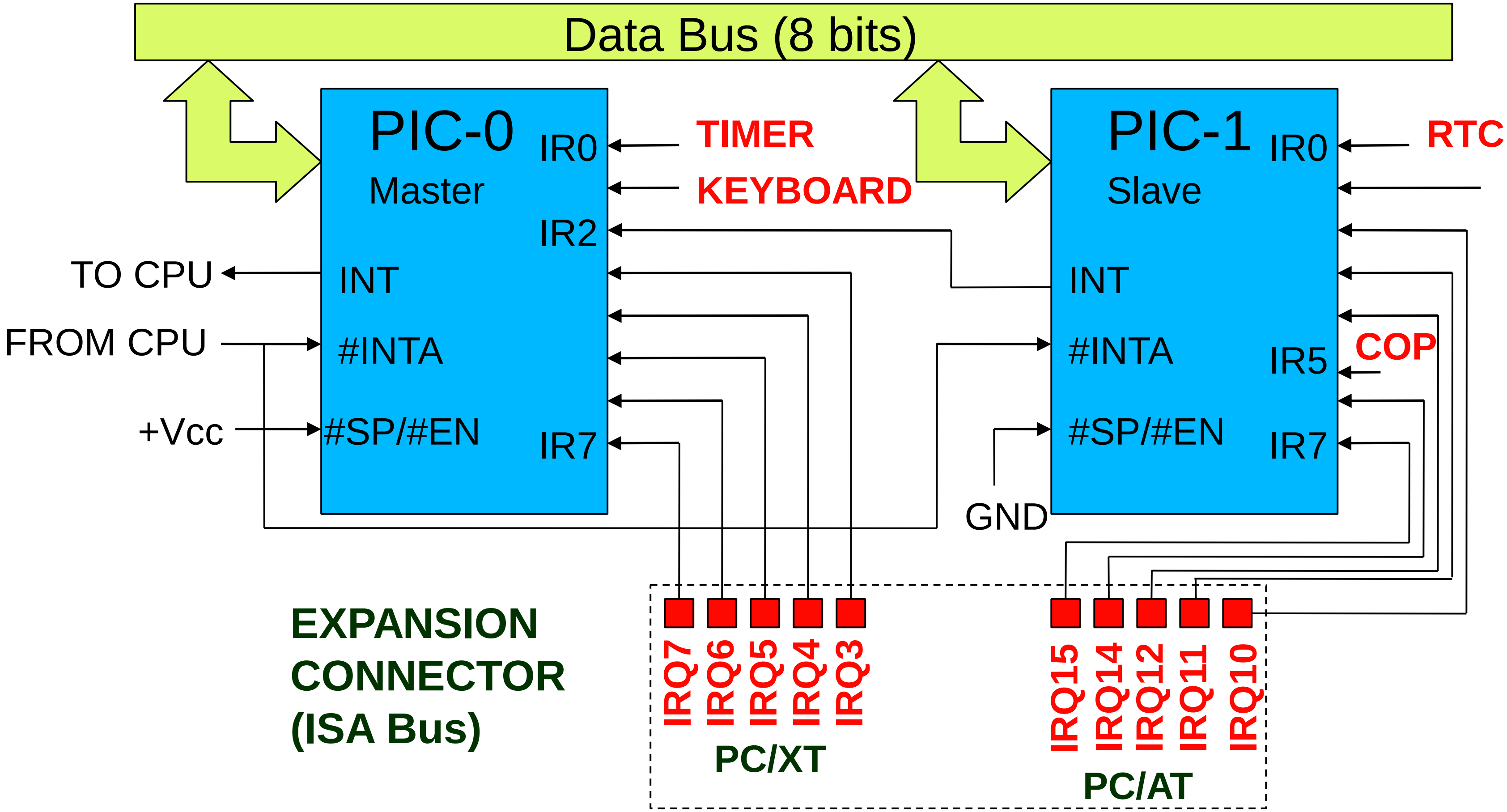
5.5. Interrupt management and programming in the 80x86 (I)

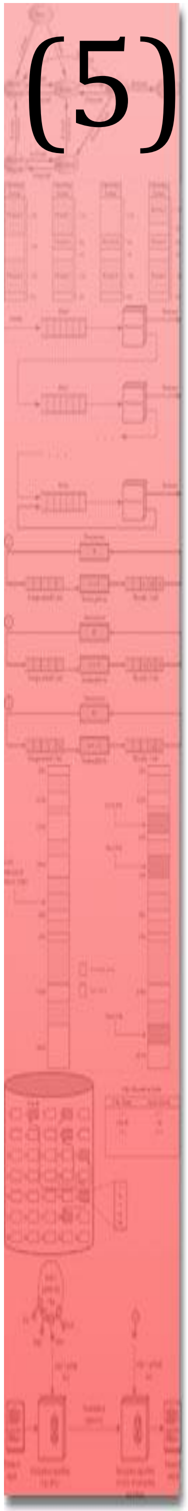
- Maskable hardware interrupts are managed through a programmable interrupt controller (PIC) 8259.
- A PIC has 8 interrupt inputs and one output.
- Multiple maskable interrupts can be managed depending on the number of installed 8259 (1 in PC/XT, 2 in PC/AT and above).
- The CPU receives a single interrupt request from the main PIC (master).
- Every interrupt can be masked independently through the 8259.
- Possible to set different priority schemes of maskable interrupts.



5.5. Interrupt management and programming in the 80x86 (II)

- Addresses PIC-0 : 20h, 21h (PC/XT, PC/AT and above)
- Addresses PIC-1 : A0h, A1h (PC/AT and above)
- Interrupts: PIC-0 ⇒ 08h (IR0) , ... , 0Fh (IR7)
  PIC-1 ⇒ 70h (IR0) , ... , 77h (IR7)

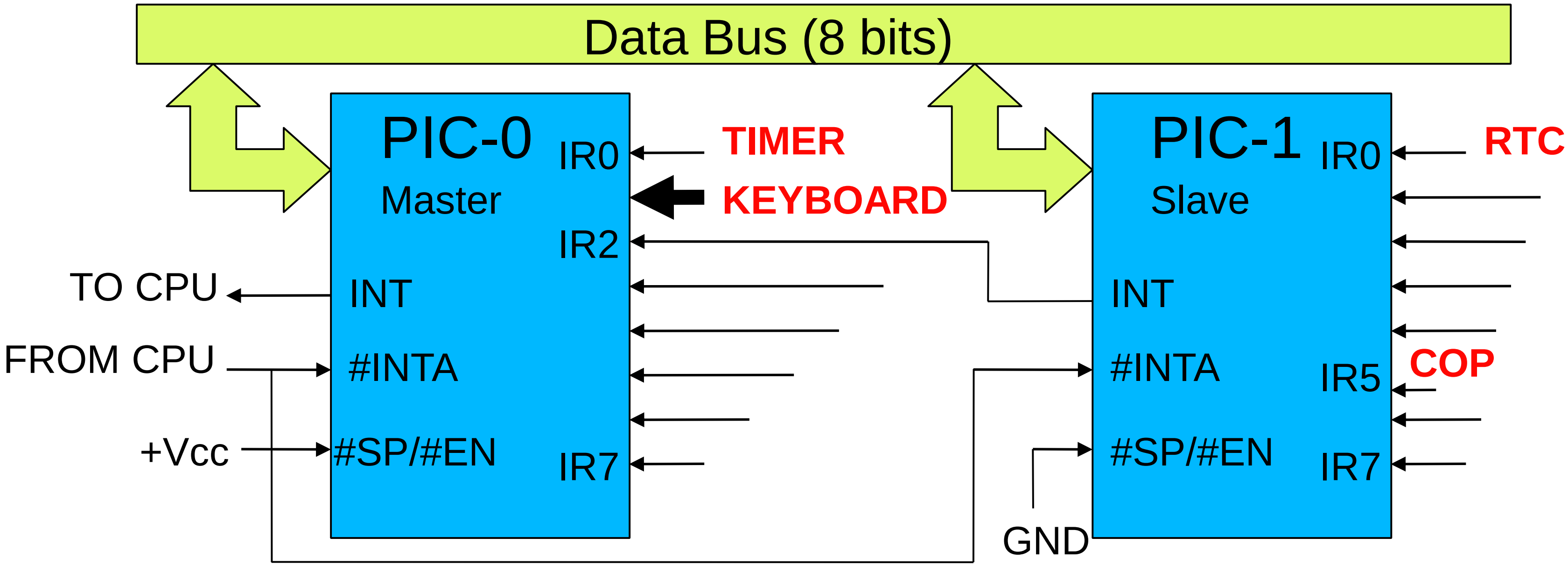




(5)

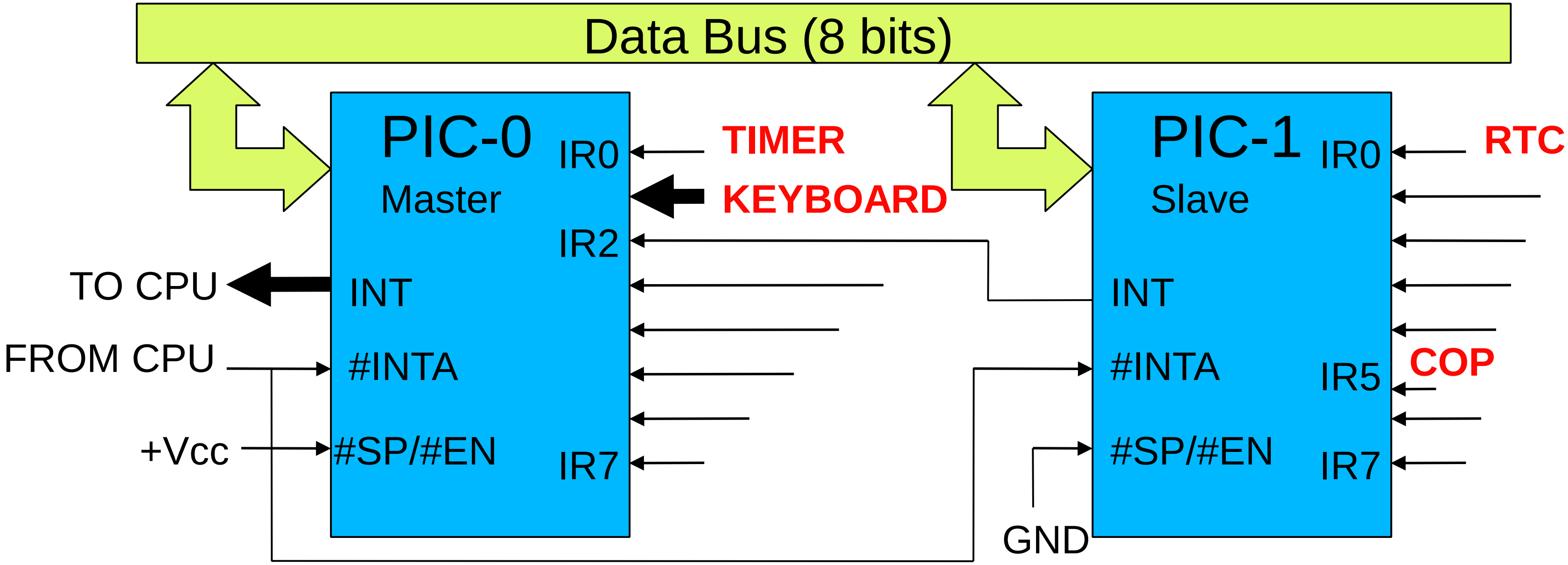
5.5. Interrupt management and programming in the 80x86 (III)

- Interrupt request through the master PIC:
  1. I/O port activates interrupt request of master.



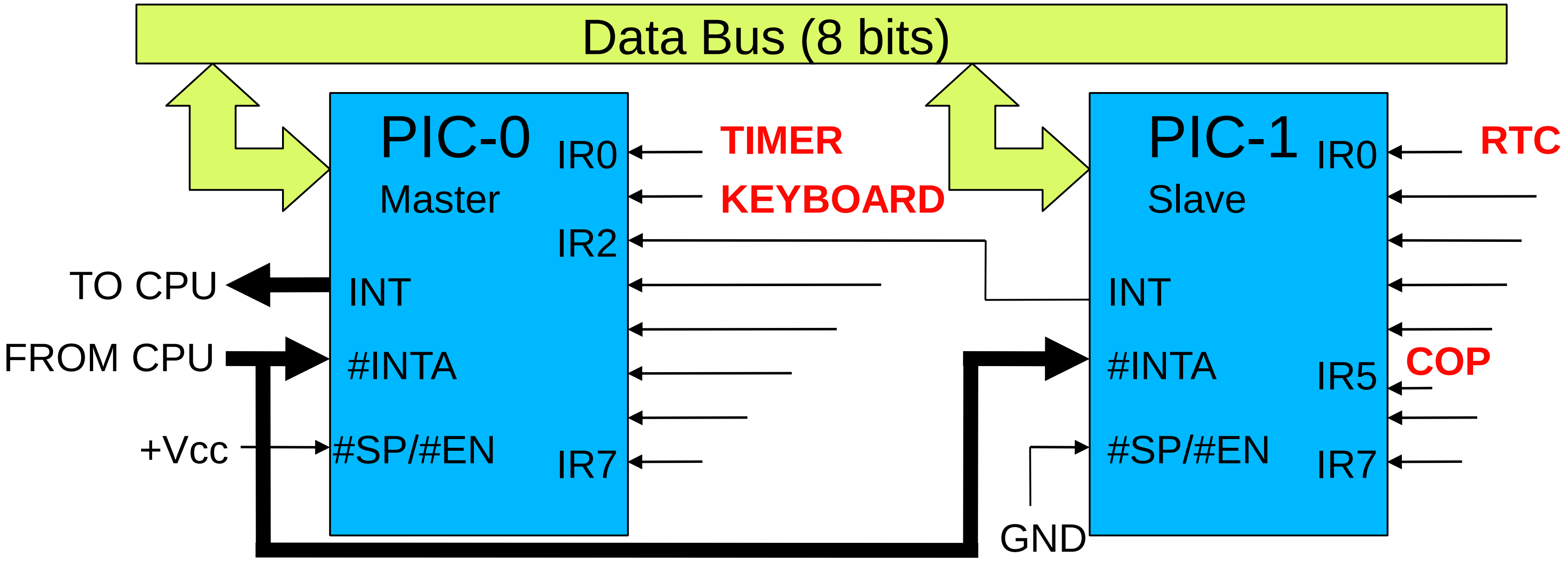
5.5. Interrupt management and programming in the 80x86 (IV)

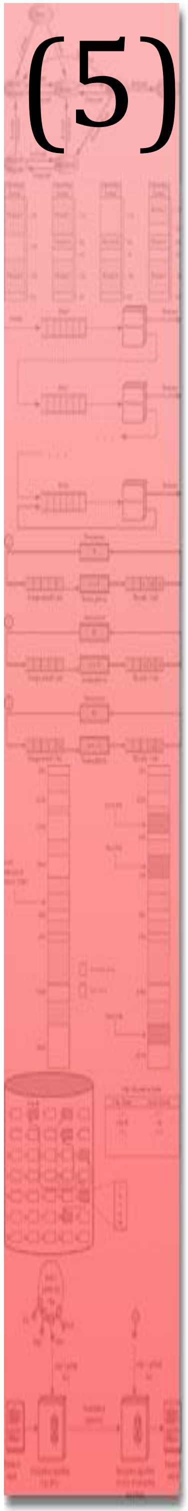
- Interrupt request through the master PIC:
  1. I/O port activates interrupt request of master.
  2. If interrupt has enough priority, master PIC activates interrupt request of CPU (INT signal).



5.5. Interrupt management and programming in the 80x86 (V)

- Interrupt request through the master PIC:
  1. I/O port activates interrupt request of master.
  2. If interrupt has enough priority, master PIC activates interrupt request of CPU (INT signal).
  3. If CPU accepts interrupt, it activates acknowledge signal (#INTA signal) two consecutive times.



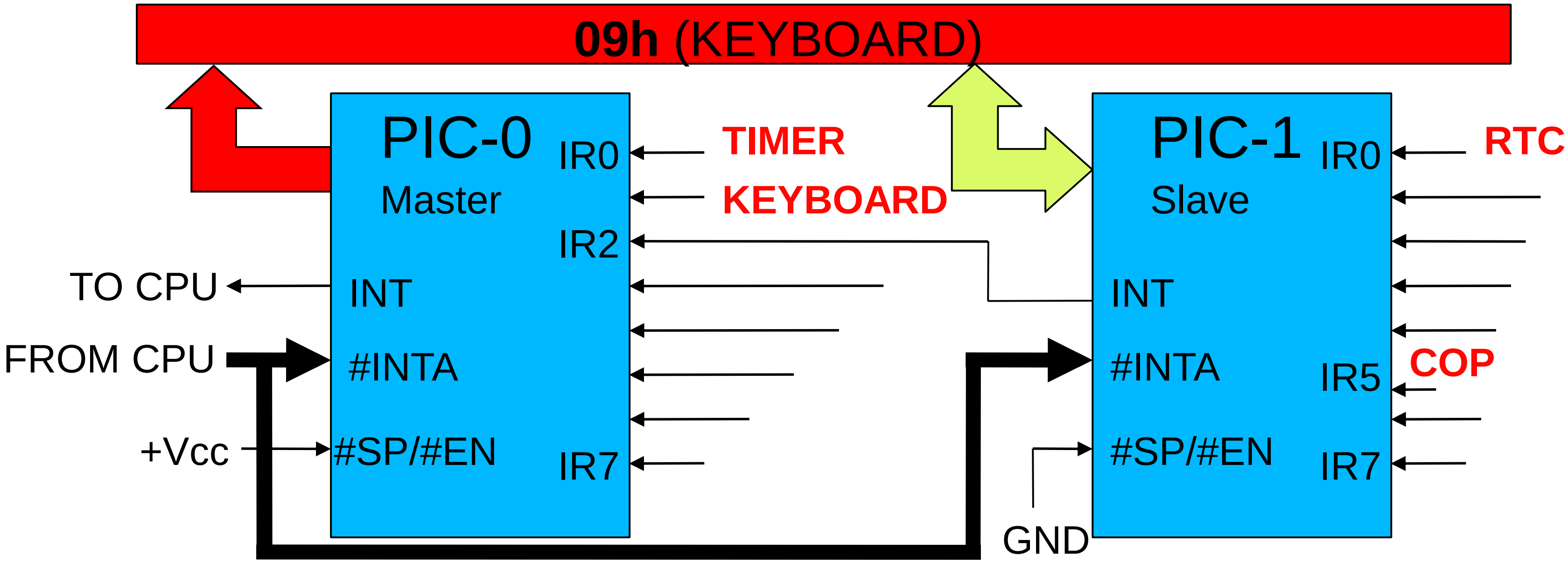


(5)

5.5. Interrupt management and programming in the 80x86 (VI)

- Interrupt request through the master PIC:

4. In the second acknowledge, master PIC writes interrupt number to data bus.



09h (KEYBOARD)

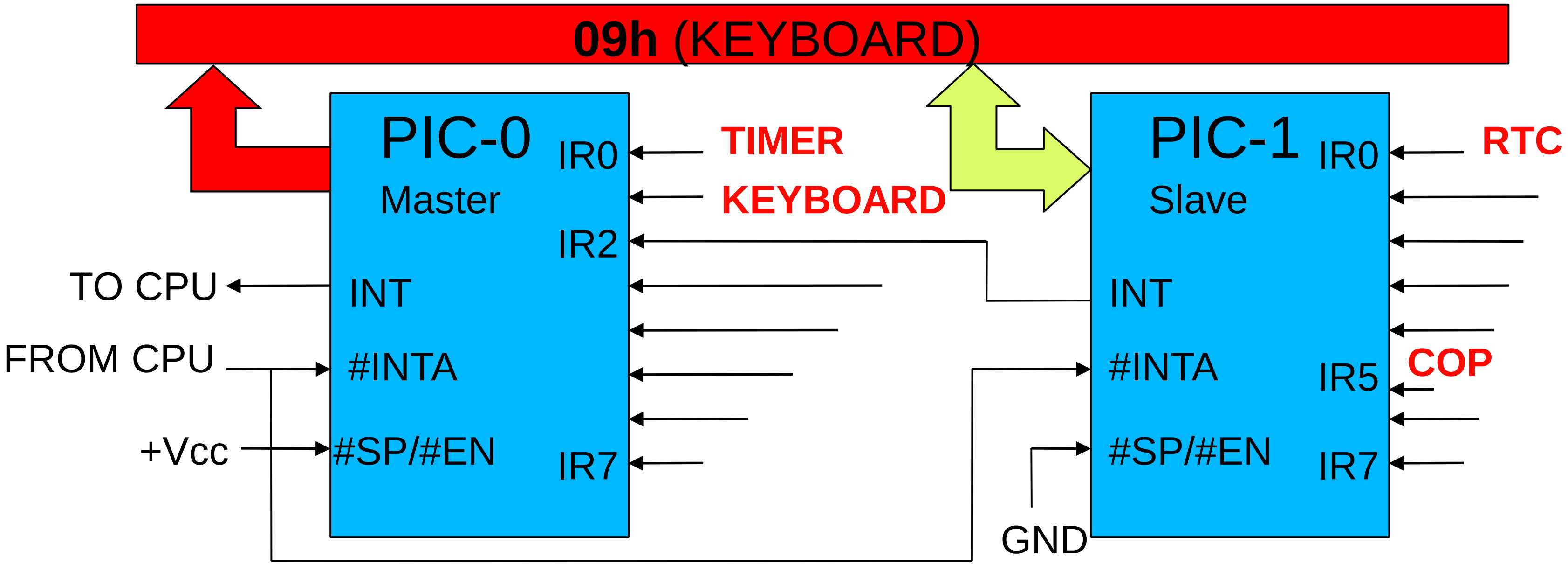
PIC-0
Master
IR0
IR2
IR7
INT
#INTA
#SP/#EN
TIMER
KEYBOARD
TO CPU
FROM CPU
+Vcc

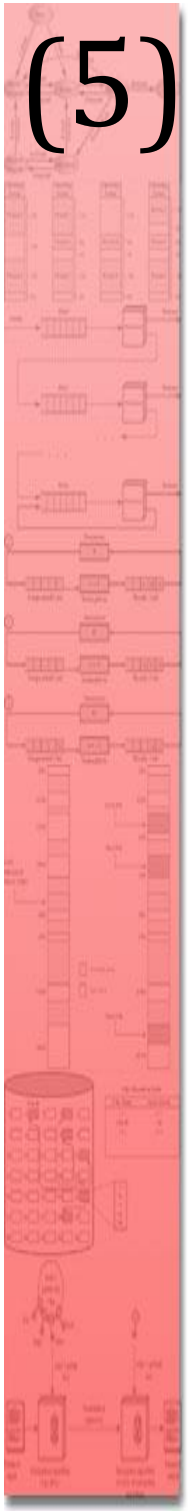
PIC-1
Slave
IR0
IR5
IR7
INT
#INTA
#SP/#EN
RTC
COP
GND

5.5. Interrupt management and programming in the 80x86 (VII)

- Interrupt request through the master PIC:

4. In the second acknowledge, master PIC writes interrupt number to data bus.
5. CPU obtains interrupt vector and executes service routine (ISR).

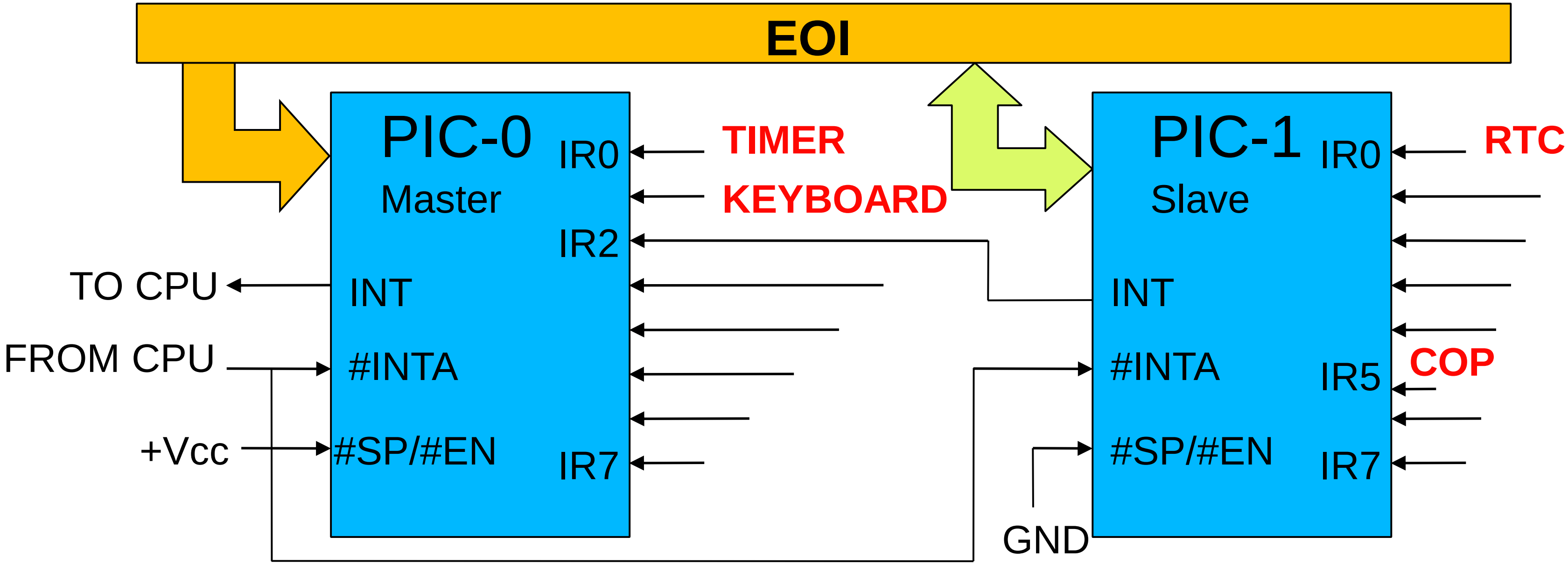


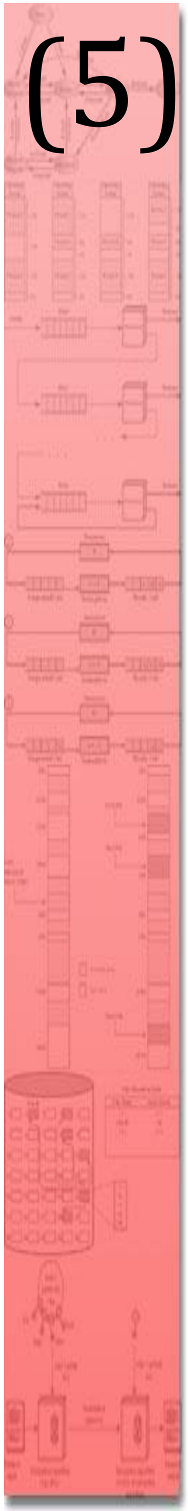


5.5. Interrupt management and programming in the 80x86 (VIII)

- Interrupt request through the master PIC:

4. In the second acknowledge, master PIC writes interrupt number into data bus.
5. CPU obtains interrupt vector and executes service routine (ISR).
6. Before ending, ISR sends End-Of-Interrupt command (*EOI*) to master PIC.
7. Master PIC terminates the interrupt request.

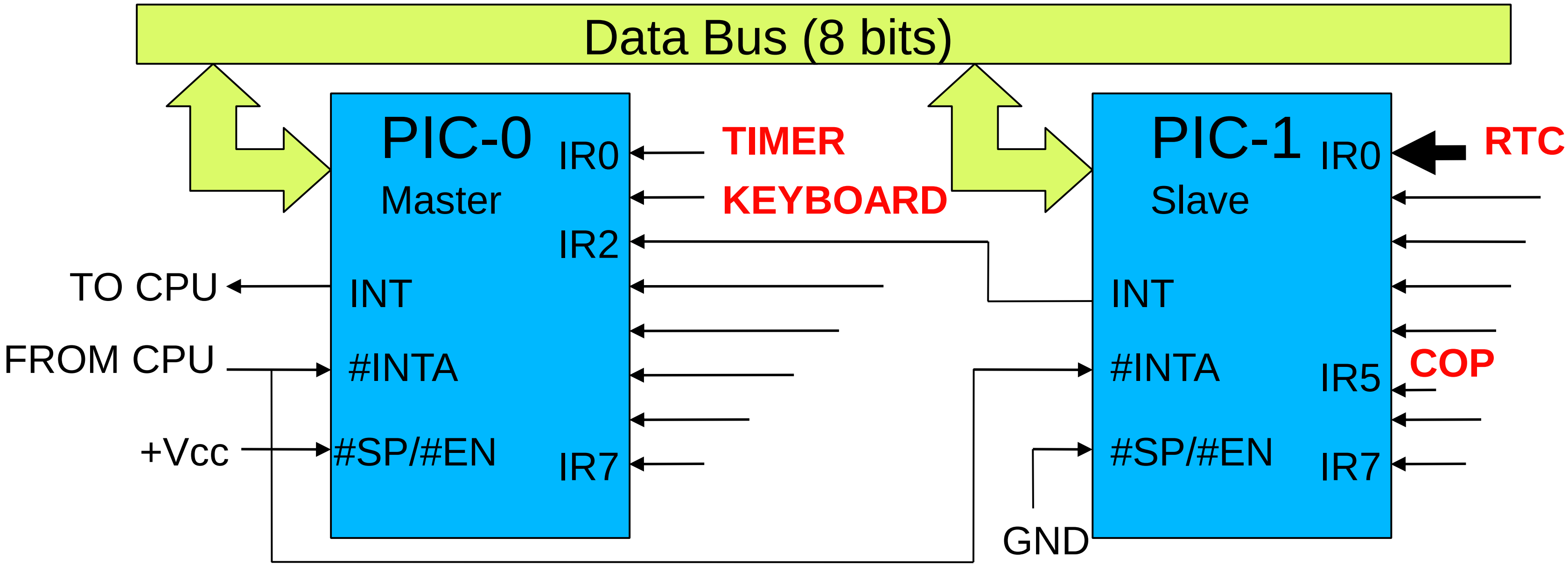




(5)

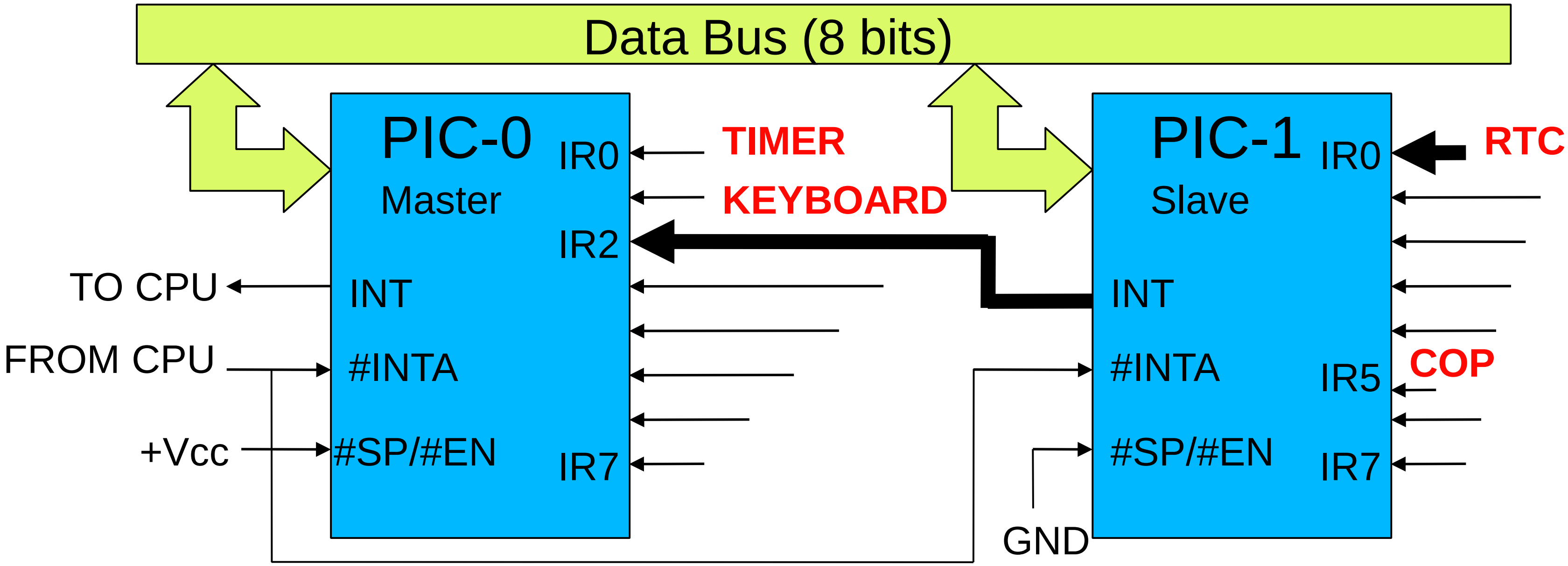
5.5. Interrupt management and programming in the 80x86 (IX)

- Interrupt request through the slave PIC:
    1. I/O port activates interrupt request of slave.



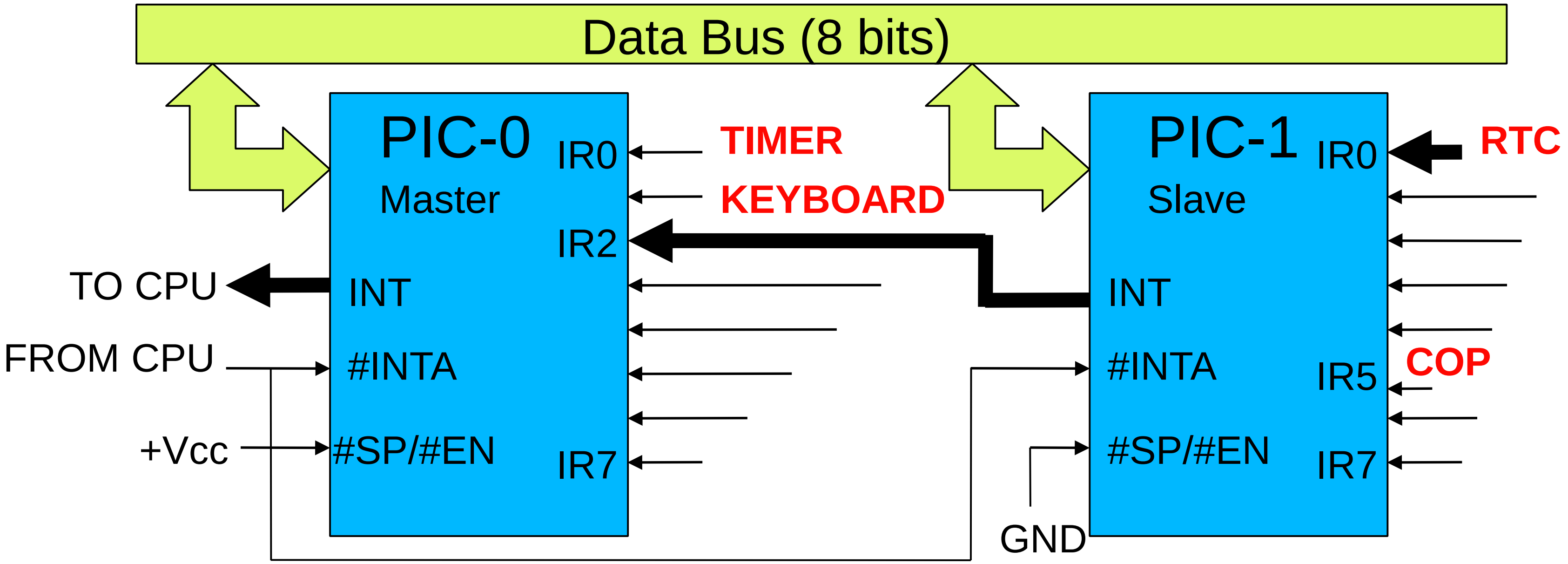
5.5. Interrupt management and programming in the 80x86 (X)

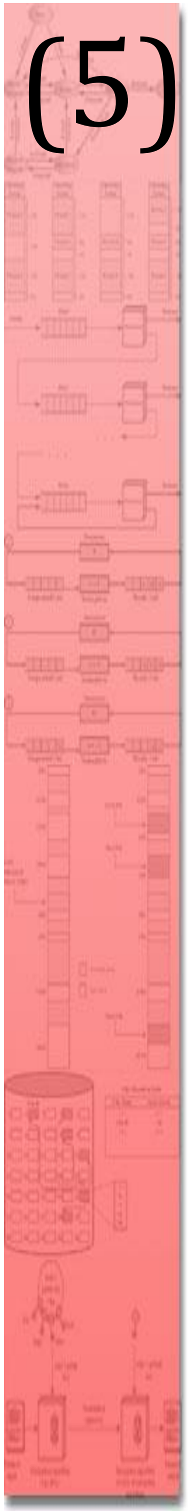
- Interrupt request through the slave PIC:
  1. I/O port activates interrupt request of slave.
  2. If interrupt has enough priority, slave PIC activates interrupt request of master PIC (IR2).



5.5. Interrupt management and programming in the 80x86 (XI)

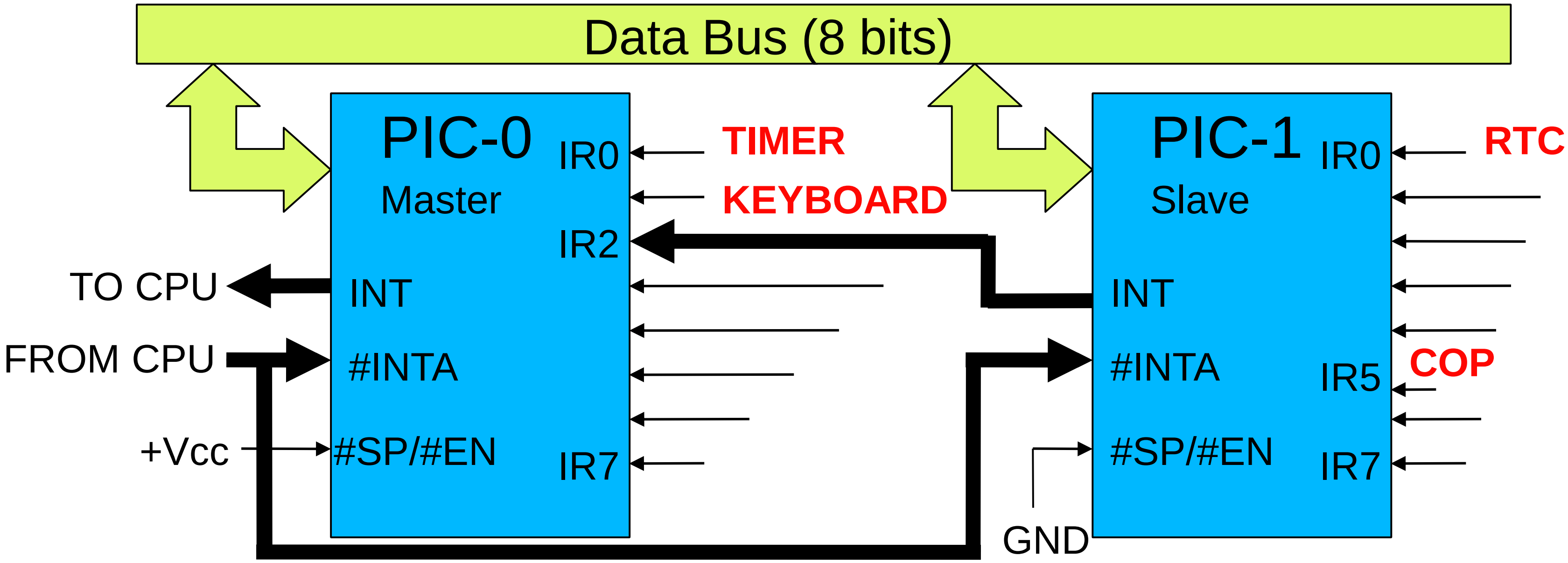
- Interrupt request through the slave PIC:
  1. I/O port activates interrupt request of slave.
  2. If interrupt has enough priority, slave PIC activates interrupt request of master PIC (IR2).
  3. If interrupt 2 has enough priority, master PIC activates interrupt of CPU (INT).

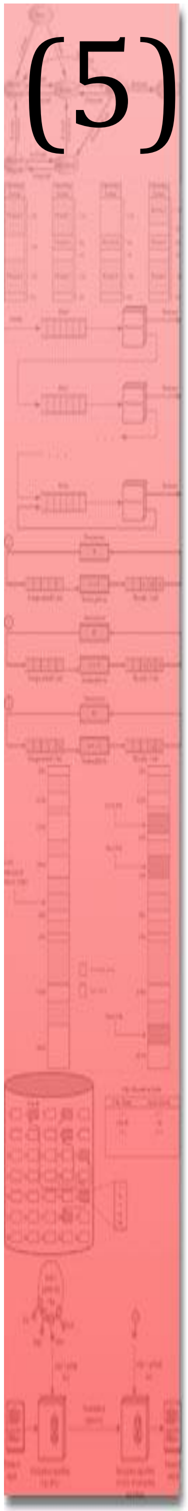




5.5. Interrupt management and programming in the 80x86 (XII)

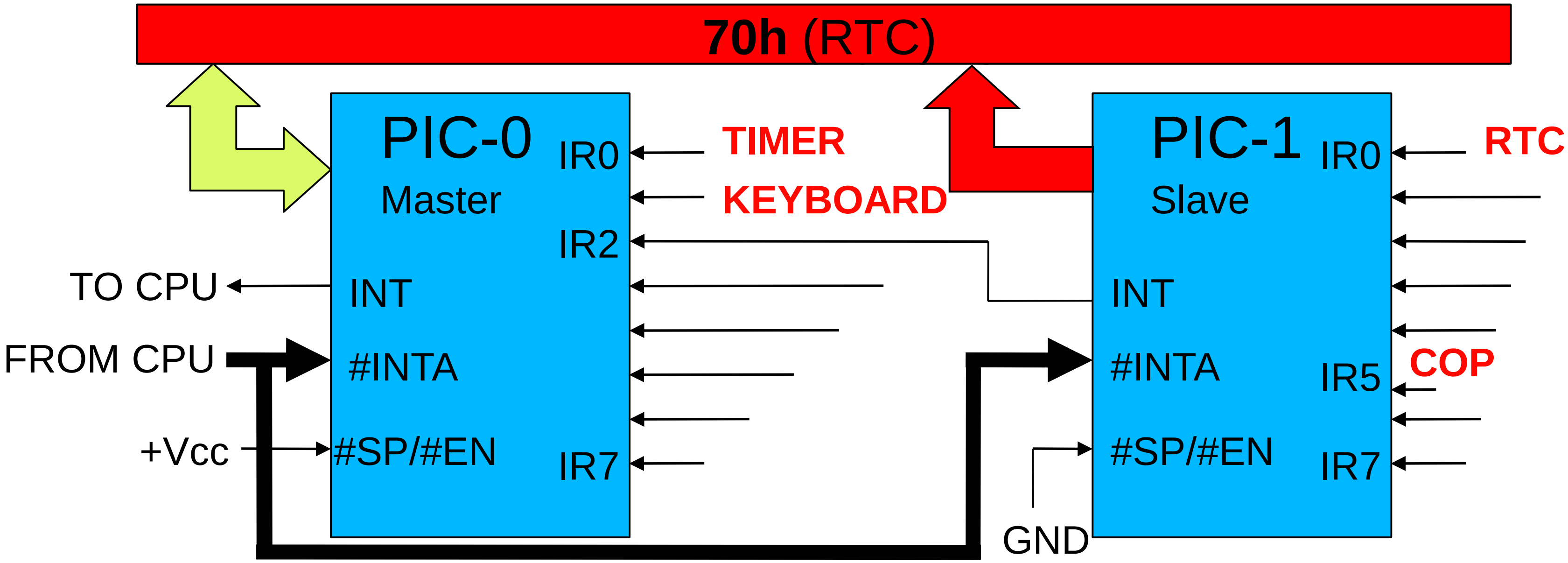
- Interrupt request through the slave PIC:
  1. I/O port activates interrupt request of slave.
  2. If interrupt has enough priority, slave PIC activates interrupt request of master PIC (IR2).
  3. If interrupt 2 has enough priority, master PIC activates interrupt of CPU (INT).
  4. CPU sends two consecutive acknowledges (#INTA).





5.5. Interrupt management and programming in the 80x86 (XIII)

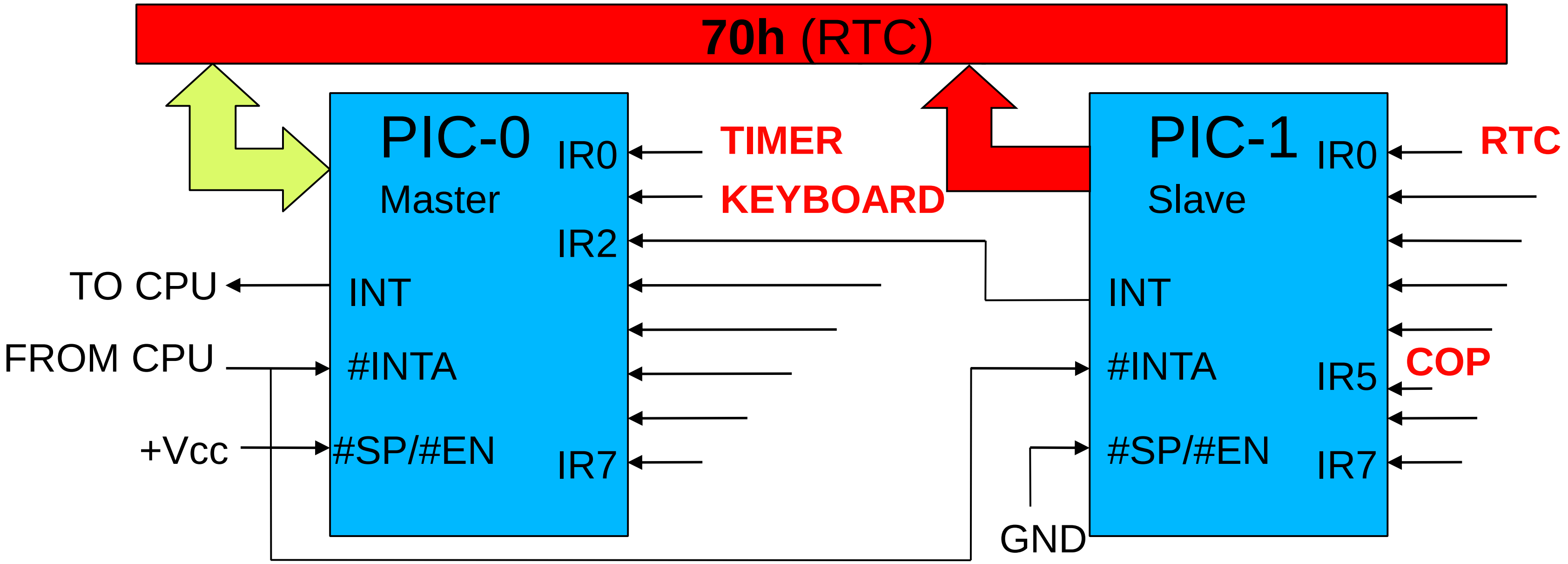
- Interrupt request through the slave PIC:
  5. In the second acknowledge, slave PIC writes interrupt number into data bus.



5.5. Interrupt management and programming in the 80x86 (XIV)

- Interrupt request through the slave PIC:

5. In the second acknowledge, slave PIC writes interrupt number into data bus.
6. CPU obtains interrupt vector and executes service routine (ISR).

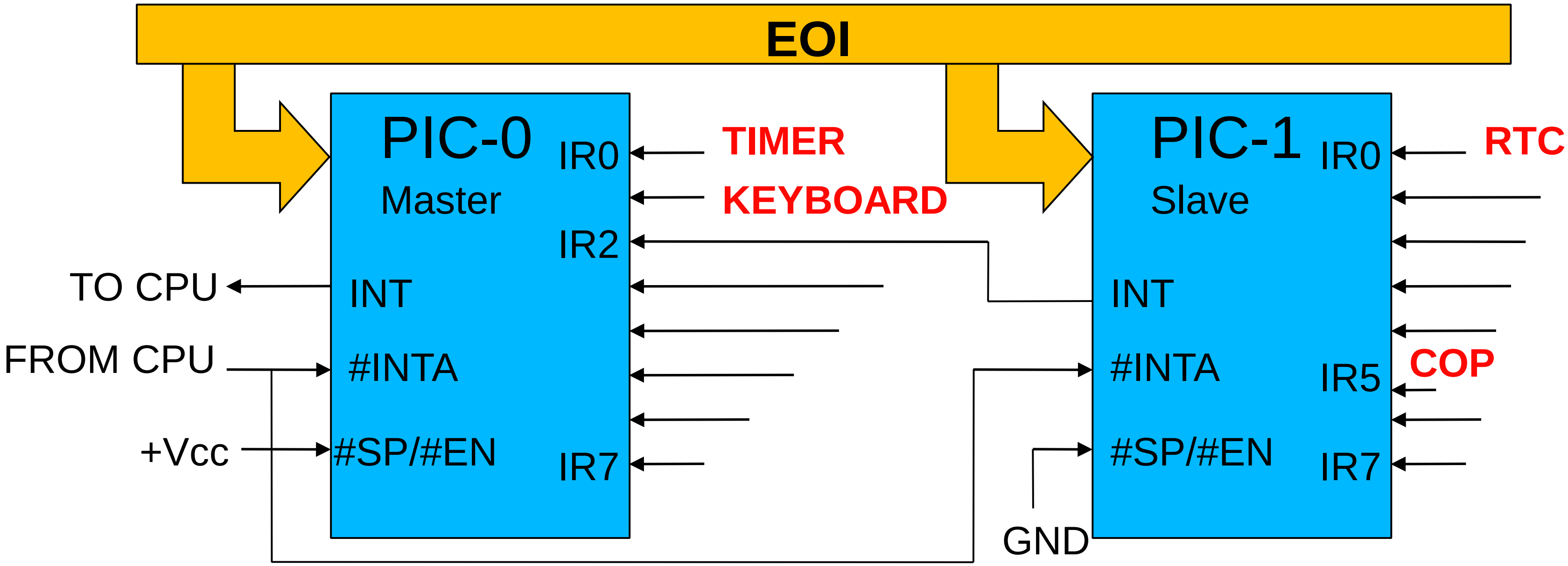


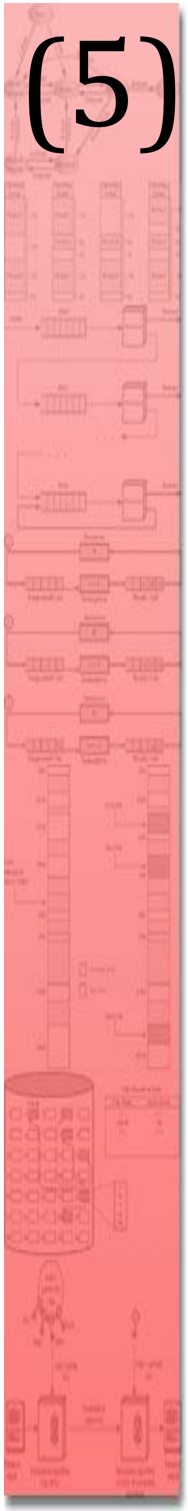
(5)

5.5. Interrupt management and programming in the 80x86 (XV)

- Interrupt request through the slave PIC:

5. In the second acknowledge, slave PIC writes interrupt number into data bus.
6. CPU obtains interrupt vector and executes service routine (ISR).
7. Before ending, ISR sends EOI command to both master and slave PICs.
8. Master and slave PICs terminate the interrupt request.



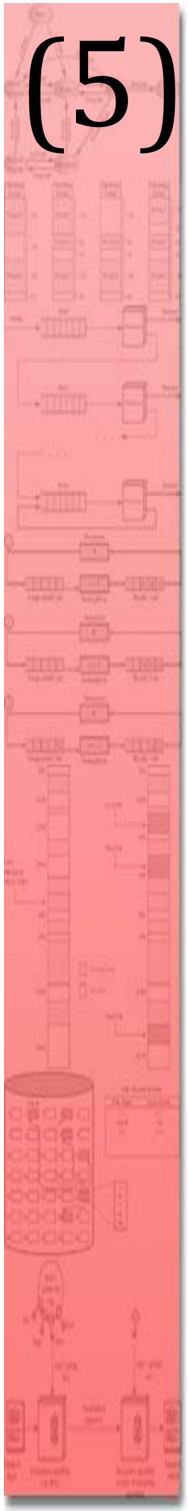


5.5. Interrupt management and programming in the 80x86 (XVI)

- Structure of the interrupt service routine of a maskable interrupt:

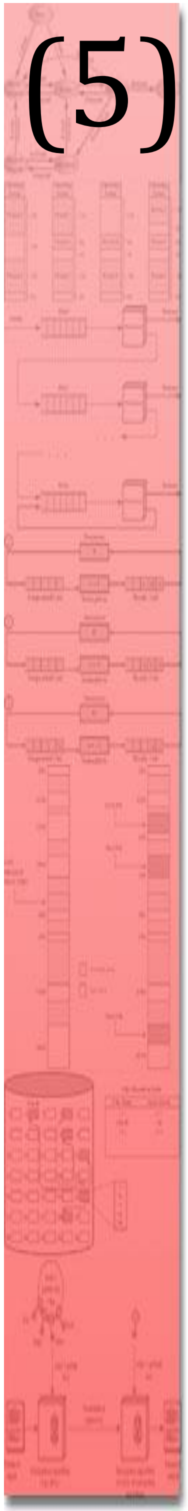
```
ISR:
        sti       ; Allows the ISR to be interrupted
        STACK REGISTERS
        .........................
        .........................
        .........................
        SEND EOI(s) (*)
        UNSTACK REGISTERS
        iret
```

(*) One **EOI** to MASTER if interrupt comes from MASTER.
One **EOI** to MASTER and one **EOI** to SLAVE if interrupt comes from SLAVE.



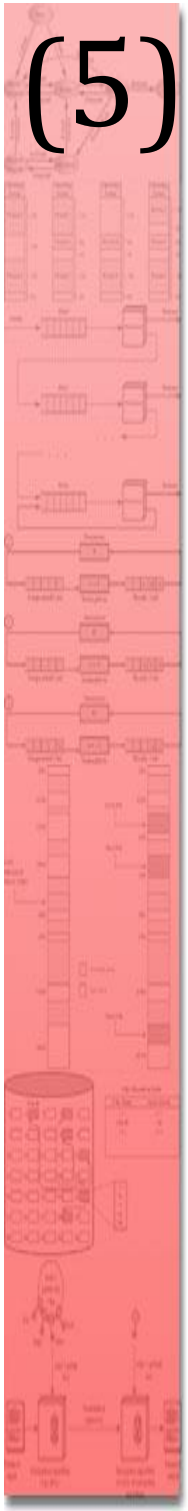
5.5. Interrupt management and programming in the 80x86 (XVII)

	IR	IRQ	Interrupt number	PC/XT	PC/AT
PIC-0	IR0	IRQ0	08h	Timer 18.2 i/s	Timer 18.2 i/s
	IR1	IRQ1	09h	Keyboard	Keyboard
	IR2	IRQ2	0Ah		PIC-1
	IR3	IRQ3	0Bh	COM2	COM2
	IR4	IRQ4	0Ch	COM1	COM1
	IR5	IRQ5	0Dh	Hard disk	LPT2
	IR6	IRQ6	0Eh	Floppy	Floppy
	IR7	IRQ7	0Fh	LPT	LPT1
PIC-1	IR0	IRQ8	70h		RTC
	IR1	IRQ9	71h		
	IR2	IRQ10	72h		
	IR3	IRQ11	73h		
	IR4	IRQ12	74h		
	IR5	IRQ13	75h		Coprocessor
	IR6	IRQ14	76h		Hard disk
	IR7	IRQ15	77h		


(5)

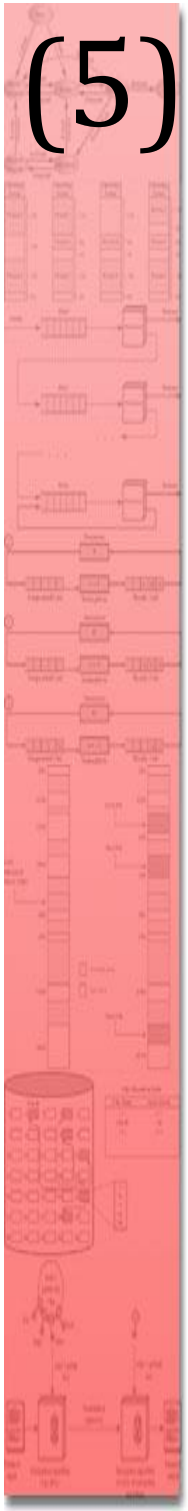
5.5. Interrupt management and programming in the 80x86 (XVIII)

- The 8259 has three 8-bit internal registers that allow the CPU to control its behavior and know its state.
- Each bit associated with an interrupt input (IR0 to IR7).
- **IMR** (Interrupt mask register)
  - A bit equal to one inhibits the requests from its associated interrupt input.
  - It can be read and modified at any time.
- **IRR** (Interrupt request register)
  - A bit equal to one indicates an interrupt request that has not been accepted by the CPU yet.
- **ISR** (Interrupt service register)
  - A bit equal to one indicates an interrupt request already accepted and not terminated yet (the CPU is executing its ISR).
  - The bit is set to zero when its ISR sends the EOI command.
  - Read only.



5.5. Interrupt management and programming in the 80x86 (XIX)

- The 8259 (PIC) has two families of “commands” to be configured by the CPU.
- *Initialization Command Words*
  - 4 consecutive commands sent from the CPU to the PIC.
  - Executed during the boot stage of the PC.
  - They configure the initial operation of the PIC.
- *Operation Command Words*
  - 3 commands that can be sent from the CPU at any time during the execution of a program.
  - They define the operation of certain aspects of the PIC and they allow the CPU to query their internal state.
  - They should be sent with the maskable interrupts having been inhibited (**IF** = 0).
- The CPU starts the execution of a command by sending one byte to the PIC with instruction **out.**

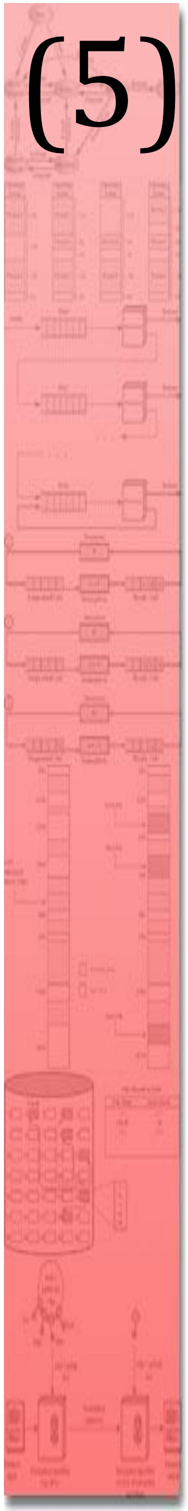


5.5. Interrupt management and programming in the 80x86 (XX)

- The type of command depends on the specified address and on some bits of the sent byte.
- Addresses: **PIC-0** (20h and 21h) , **PIC-1** (A0h and A1h)

Command name	Address	Distinguished by
ICW1	A0 = 0 (even)	D4 to 1
ICW2	A0 = 1 (odd)	After ICW1
ICW3	A0 = 1 (odd)	After ICW2
ICW4	A0 = 1 (odd)	After ICW3
OCW1	A0 = 1 (odd)	
OCW2	A0 = 0 (even)	D3 to 0 and D4 to 0
OCW3	A0 = 0 (even)	D3 to 1 & D4 to 0

A0 (address bus) , D3 and D4 (data bus)



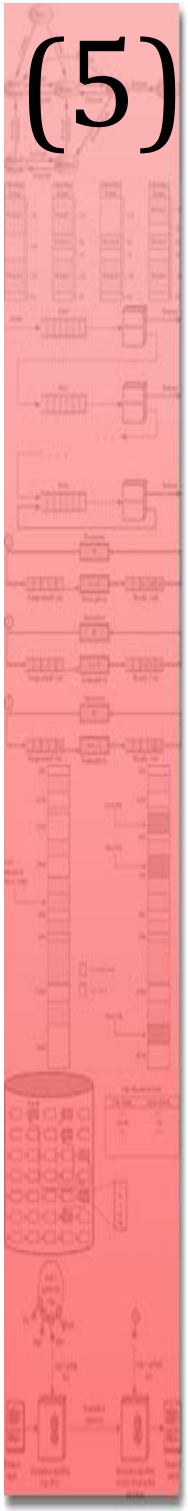
(5)

5.5. Interrupt management and programming in the 80x86 (XXI)

OCW1

- Odd address (21h or A1h)
- Used for reading or writing the interrupt mask register **IMR**.
- D7, ... ,D0: Interrupt mask **IMR**:
  - Bit to 0 = enabled interrupt
  - Bit to 1 = disabled interrupt (inhibited)
- Examples:
  - **out** 21h, al ; Write **IMR** of PIC-0
  - **in** al, 21h ; Read **IMR** of PIC-0
  - **out** A1h, al ; Write **IMR** of PIC-1
  - **in** al, A1h ; Read **IMR** of PIC-1

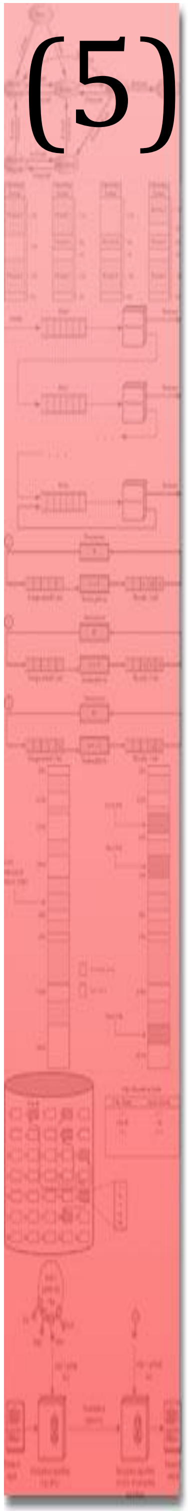
A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
1								



5.5. Interrupt management and programming in the 80x86 (XXII)

OCW2

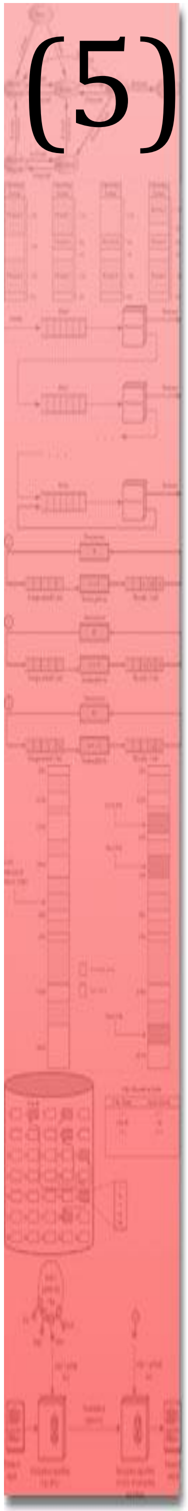
- Even address (20h or A0h) and D4 = D3 = 0.
- Used for:
  - Send **EOI**s (End of interrupt service routine):
    - Non-specific.
    - Specific.
  - Change priority of interrupts (**rotations**)
    - Specific.
    - Automatic.
  - A single OCW2 command is able to:
    - Send an EOI.
    - Perform a priority rotation.
    - Send an EOI and perform a priority rotation.



5.5. Interrupt management and programming in the 80x86 (XXIII)

OCW2

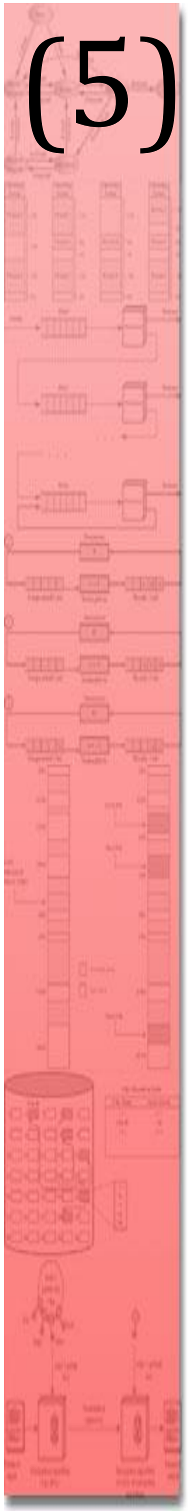
- Non-specific EOI
  - The terminating interrupt is not specified.
  - PIC sets to 0 the **ISR** bit of the IR with the largest priority.
  - Usual EOI.
- Specific EOI
  - It specifies the **ISR** bit that must be set to 0.
  - Necessary when the terminating interrupt is not the one with the largest priority.
- Automatic EOI (must be activated through ICW)
  - Service routine must not send EOI.
  - PIC sets to 0 the **ISR** bit after receiving the second #INTA (before executing the service routine).
  - An interrupt of any priority can interrupt the service routine that is being executed.



5.5. Interrupt management and programming in the 80x86 (XXIV)

OCW2

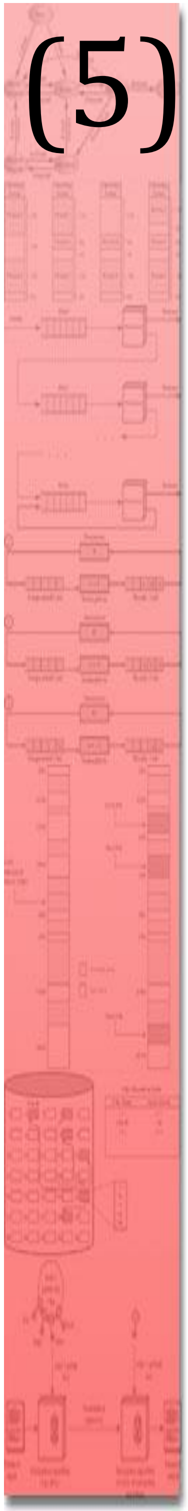
- Priority management in **normal mode** (by default)
  - Each IR has a fixed priority.
  - Initially ⇒ IR7 (lowest) < IR6 < … < IR0 (highest)
  - When an interrupt is being served, the PIC does not deliver the ones of lower of equal priority.
  - The priority order can be changed through a rotation command.
  - The rotation must indicate the IR with the lowest priority.
  - The priority of the other IRs is readjusted cyclically.
  - Example: Rotation of IR2
    - IR2 (min) < IR1 < IR0 < IR7 < IR6 < IR5 < IR4 < IR3 (max)
  - **Specific** rotation ⇒ OCW2 command indicates the IR with the new lowest priority.
  - **Automatic** rotation ⇒ The terminating IR (indicated by EOI) has the new lowest priority (*round-robin*)



5.5. Interrupt management and programming in the 80x86 (XXV)

OCW2

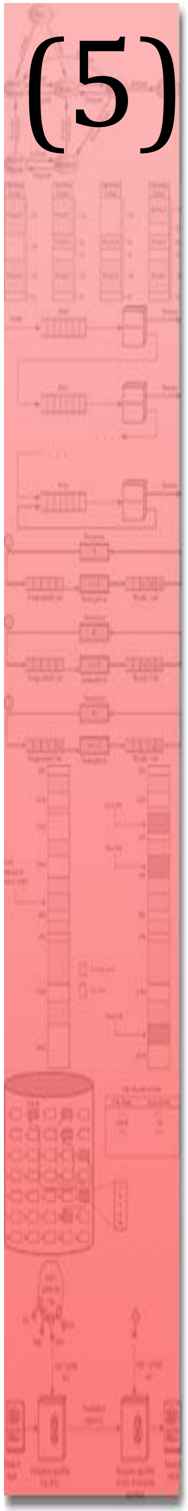
- Priority management in **normal mode** (by default)
  - **“Priority inversion”** problem**:**
    - PIC-0 (master) does not serve (does not send to the CPU) new interrupts from PIC-1 (slave) until the last interrupt sent by that slave terminates (master receives EOI).
    - If slave sends low-priority interrupt (e.g., hard disk) and then one of higher priority (e.g., RTC), master holds off the second until the first terminates ⇒ IR of higher priority becomes of lower effective priority (**priorities are inverted**).
    - **Solution:** Configure master PIC in **special mode** of priority management.



5.5. Interrupt management and programming in the 80x86 (XXVI)

OCW2

- Priority management in **special mode**
  - Activated or deactivated through an OCW3 command.
  - Only applicable to PIC-0 (master).
  - Master serves any new request from slave even if there are other interrupts from the slave pending to be terminated.
  - **EOI** has only to be sent to master when slave has no interrupts being served (slave's **ISR** is 0).



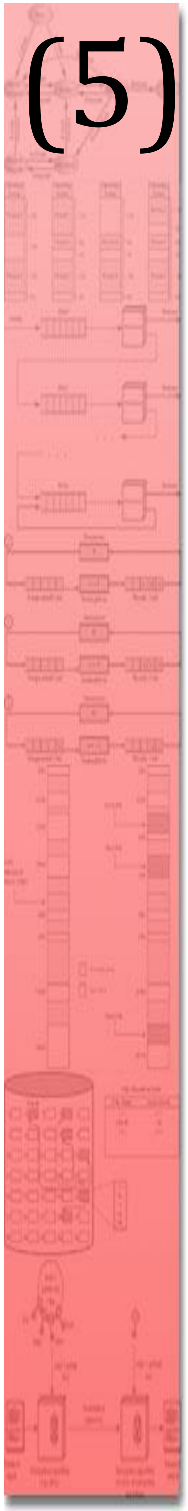
5.5. Interrupt management and programming in the 80x86 (XXVII)

OCW2

- D2,D1,D0
  - IR that terminates (if specific EOI) or with the lowest priority (if rotation).
- D4,D3
  - 0,0 (fixed)

A0		D7	D6	D5	D4	D3	D2	D1	D0
0					0	0			

- D7,D6,D5
  - 0,0,1 : **Non-specific EOI** (20h)
  - 0,1,1 : Specific EOI (specify D2-D0)
  - 1,0,1 : Non-specific EOI with automatic rotation
  - 1,0,0 : Activate automatic rotation in automatic EOI
  - 0,0,0 : Deactivate automatic rotation in automatic EOI
  - 1,1,1 : Specific EOI with automatic rotation (specify D2-D0)
  - 1,1,0 : Specific rotation (specify D2-D0)
  - 0,1,0 : Not used



5.5. Interrupt management and programming in the 80x86 (XXVIII)

OCW2

- Examples:

  -
    ```
    mov al, 20h
    out 20h, al            ; Non-specific EOI to PIC-0 (master)
    ```

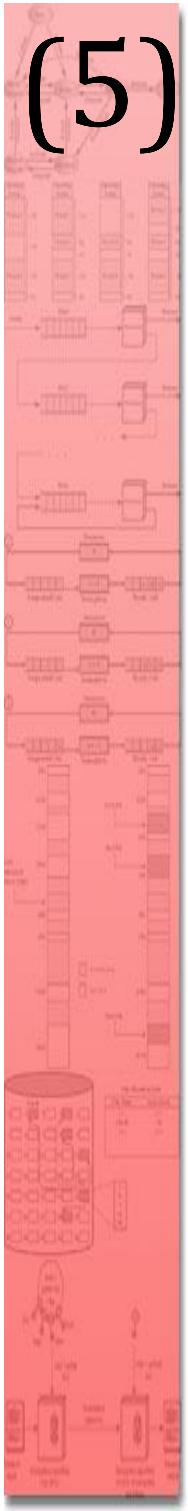
  -
    ```
    mov al, 01100011b
    out A0h, al            ; EOI 3 to PIC-1 (slave)
    ```

  -
    ```
    mov al, 10100000b
    out A0h, al            ; Non-specific EOI with automatic rotation
    ```

	Before	**After**
ISR (PIC-1)	10010000	10000000
Priority	76543210	21076543

  -
    ```
    mov al, 11100010b
    out 20h, al            ; EOI 2 with automatic rotation
    ```

	Before	**After**
ISR (PIC-0)	10000100	10000000
Priority	76543210	43210765



5.5. Interrupt management and programming in the 80x86 (XXIX)

OCW3

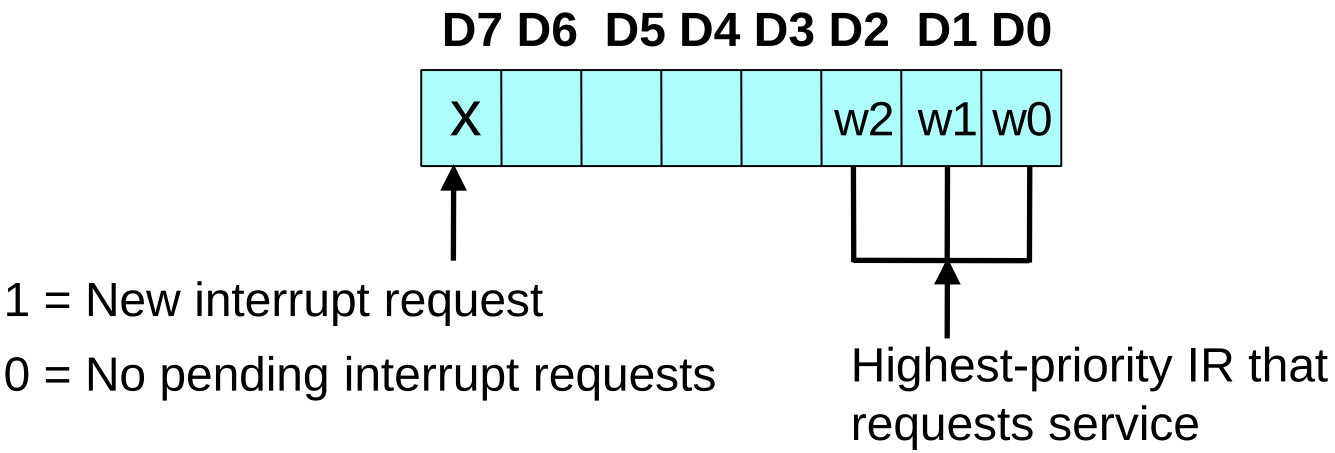
- Even address (20h or A0h), D4 = 0 and D3 = 1.
  - Used for reading the service register (**ISR**) or the request register (**IRR**), activation/deactivation of “special mode” and execution of POLL command.
- D1,D0: Register read in the next reading
  - 1,0 = **IRR**; 1,1 = **ISR**
- D2: POLL command
  - 0 = Inactive, 1 = Active
- D4,D3: 0,1 (fixed)
- D6,D5: Special mode of priority management
  - 1,0 = Deactivate;   1,1 = Activate;   0,0 = 0,1 = Ignore.
- D7: 0 (fixed)

A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
0	0			0	1			

5.5. Interrupt management and programming in the 80x86 (XXX)

OCW3

- POLL command.
  - It allows the CPU to use the PIC with inhibited maskable interrupts (**IF** = 0).
  - CPU sends POLL command to PIC.
  - CPU reads PIC next:
    - In case of pending requests in **IRR**, PIC sets the corresponding bit of the **ISR** to 1 according to the priority scheme (equivalent to #INTA).
    - PIC writes the following state byte into the data bus:

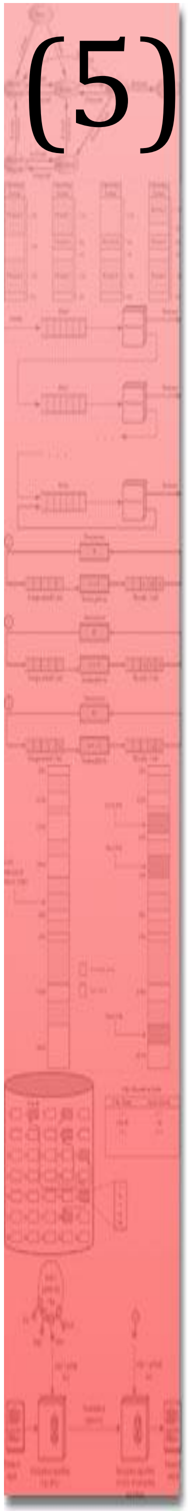


D7	D6	D5	D4	D3	D2	D1	D0
X					w2	w1	w0

1 = New interrupt request

0 = No pending interrupt requests

Highest-priority IR that requests service



5.5. Interrupt management and programming in the 80x86 (XXXI)

OCW3

- Examples:

  -
    ```
    mov al, 00001010b
    out 20h, al          ; Read request for IRR of PIC-0
    in al, 20h           ; Read IRR of PIC-0
    ```

  -
    ```
    mov al, 00001011b
    out A0h, al          ; Read request for ISR of PIC-1
    in al, A0h           ; Read ISR of PIC-1
    ```

  -
    ```
    mov al, 01101000b
    out 20h, al          ; Activate special mode in PIC-0
    ```

  -
    ```
    mov al, 01001000b
    out 20h, al          ; Return to normal mode in PIC-0
    ```

  -
    ```
    mov al, 00001100b
    out 20h, al          ; Send POLL command to PIC-0
    in al, 20h           ; Receive state byte from PIC-0
    ```