Interrupts
PIC – 8259A

CEN433

King Saud University

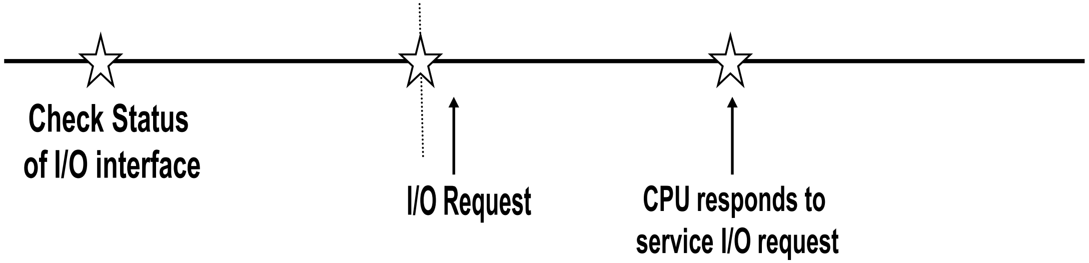
Dr. Mohammed Amer Arafah

Methods to Service an I/O Device

- Programmed I/O (Polling)
- Interrupt Driven I/O

Programmed I/O

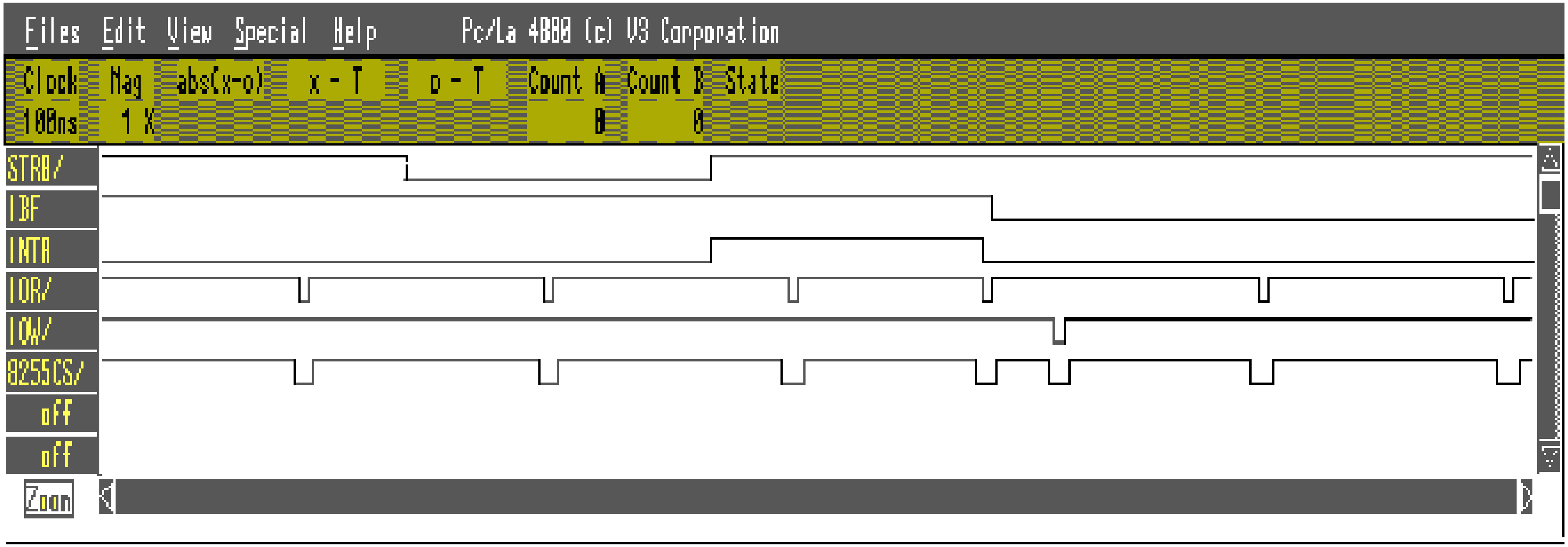
- Programmed I/O method is the first method used to check whether an I/O device needs a service.
- Programmed I/O checks the status of I/O interface periodically under the control of the software.
- It is appropriate for small systems and dedicated applications with limited number of I/O devices.
- The **drawbacks** of programmed I/O:
  - It wastes the CPU time in busy wait loops.
  - It is difficult to implement prioritized service.
  - It worsens the response time of a CPU to respond to a request.



Programmed I/O

```
START:          MOV     DX, PORTC
AGAIN:          IN      AL, DX
                TEST    AL, 00001000B
                JZ      AGAIN
;
                MOV     DX, PORTA
                IN      AL, DX
                INC     DX
                OUT     DX, AL

                JMP     START
```

Programmed I/O



Clock 100ns Mag 1 X Count A 0 Count B 0

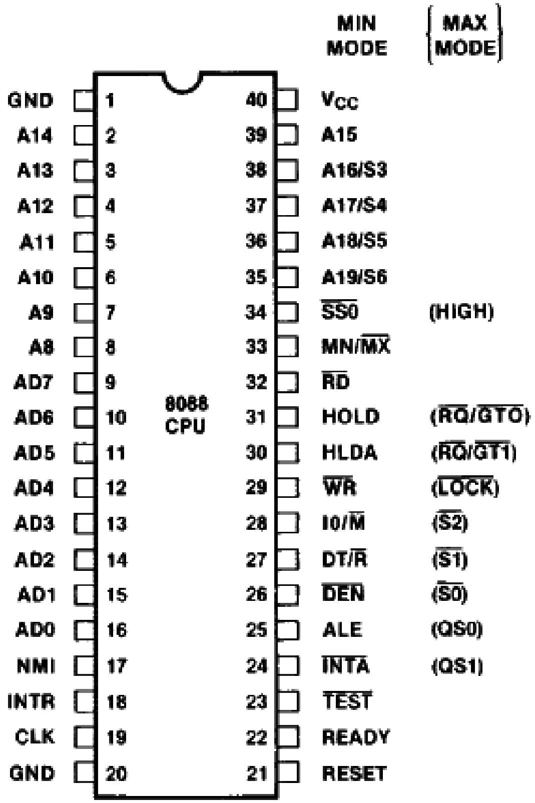
Interrupt Driven I/O

- Unlike the **polling technique**, interrupt processing allows the microprocessor to execute other software. As soon as the microprocessor gets an interrupt from I/O interface, it stops executing the running program and starts the execution of the **Interrupt Service Routine** (**ISR**) to service the I/O device. When the microprocessor finishes the execution of ISR, it returns to the interrupted program.
- The **advantages** of the **interrupt driven I/O** method:
  - □ It increases the **CPU throughput** since the microprocessor does not waste its time in the **busy wait loop**.
  - □ It is easy to implement **prioritized service**.
  - □ It improves the **response time**.

Interrupts

- The interrupts of entire Intel family of microprocessors include two hardware pins that request interrupts (**INTR** and **NMI**) and one hardware pin (**INTA/**) that acknowledges the interrupt requested through **INTR**.

- Two flag bits: **IF** (**Interrupt Flag**) and **TF** (**Trap Flag**) are also used with the interrupt structure.
  - Interrupt flag (**IF**) controls the INTR input pin.
  - Trap flag (**TF**) turns tracing (single stepping) on or off.



FLAGS		O	D	I	T	S	Z		A		P		C
	15	11	10	9	8	7	6	5	4	3	2	1	0

Interrupts

- The microprocessor also has **software interrupts** **INT n**, **INTO**, **INT3**, and **BOUND**.

- **INT n** Instruction.

- Special cases:
  → **n = 1:** Single step (trap)
  Enabled if **TF** (Trap flag) bit in Flags is set
  → **n = 3:** INT 3 (Breakpoint)
  → **n = 4:** INTO (Overflow)
  → **n = 5:** BOUND (Check limits)

- A special return instruction **IRET** is used at the end of the **Interrupt Service Routine** (**ISR**) for both hardware/software interrupts.

- IRET retrieves from the stack:
  - Return address (CS & IP)
  - The Flags register

Interrupts

- **Conditional:**
  - □ **INTO** (type 4): on Overflow (the OF flag bit)
  - □ **BOUND** (type 5): on the result of comparing the contents of a register with the contents of two successive words in memory, e.g.

    BOUND AX, DATA

    causes interrupt type 5 if AX contents are outside the range specified by the 4 bytes

    i.e., [AX] < {[DATA], [DATA+1]} or [AX] > {[DATA+2], [DATA+3]}

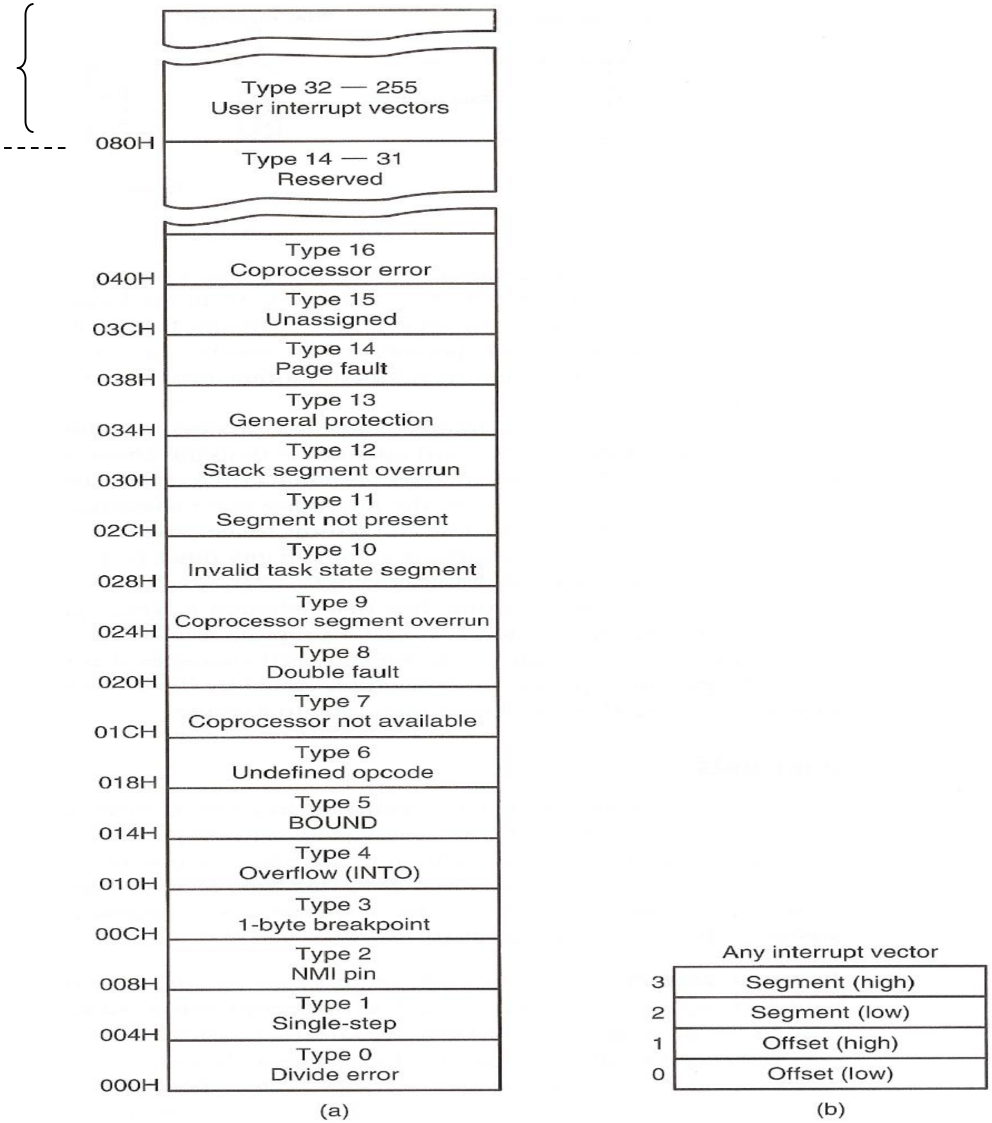
- **Unconditional:**
  - □ **INT n**: Executes the interrupt with the vector type n.
  - □ Instruction takes 2 bytes: a byte for the opcode and a byte for n.
  - □ e.g. **INT 80H** → Start of the 4-byte vector address = 4 x 80H = 200H (200H – 203H).
  - □ **INT 3**: is a special case of INT n which fits into one byte only. Useful in inserting a break point in the software for debugging purposes

Interrupt Vectors

- The **Interrupt Vector Table** (**IVT**) is located in the first 1024 bytes of memory at addresses 00000H-003FFH.
- It contains 256 different 4-byte interrupt vectors.
- An **interrupt vector** contains the starting address (segment and offset) of **Interrupt Service Routine** (**ISR**). The first two bytes of the vector contain the offset address, and the last two bytes contain the segment address.

Interrupt Vector Table

Available Interrupt Pointers (224)



	Type 32 — 255 User interrupt vectors
080H	Type 14 — 31 Reserved
	Type 16 Coprocessor error
040H	Type 15 Unassigned
03CH	Type 14 Page fault
038H	Type 13 General protection
034H	Type 12 Stack segment overrun
030H	Type 11 Segment not present
02CH	Type 10 Invalid task state segment
028H	Type 9 Coprocessor segment overrun
024H	Type 8 Double fault
020H	Type 7 Coprocessor not available
01CH	Type 6 Undefined opcode
018H	Type 5 BOUND
014H	Type 4 Overflow (INTO)
010H	Type 3 1-byte breakpoint
00CH	Type 2 NMI pin
008H	Type 1 Single-step
004H	Type 0 Divide error
000H	

(a)

Any interrupt vector

3	Segment (high)
2	Segment (low)
1	Offset (high)
0	Offset (low)

(b)

Example

- **Question**

  A particular interrupt has a type number n=41H. If the corresponding ISR begins at address FE00:0500H, determine the locations in the vector table to store this address.

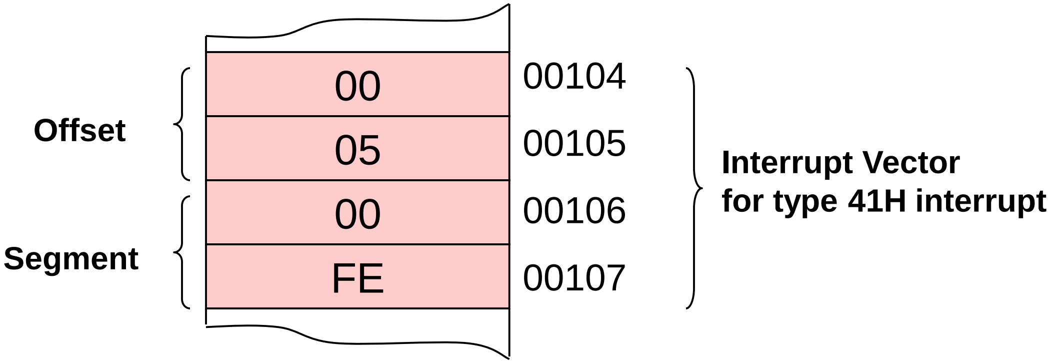
- **Solution:**

  The vector address is calculated by multiplying 41H by four.

  This is done by rotating 41H left twice.

  41H = 0100 0001; rotate left twice 01 0000 0100 = 104H

  The offset address of the ISR is stored in the low-word location and the segment address in the high-word location.



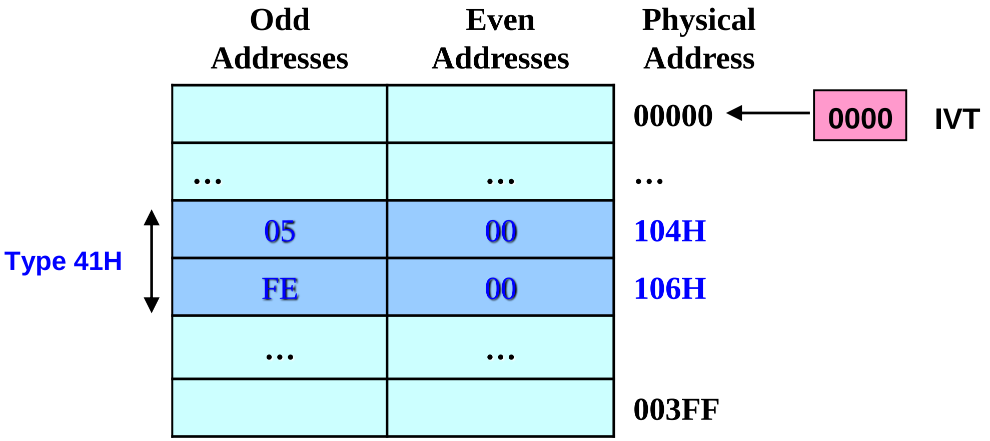
	Content	Address
Offset	00	00104
	05	00105
Segment	00	00106
	FE	00107

Interrupt Vector for type 41H interrupt

Example

```
IVT       SEGMENT AT   0H
          ORG      41H * 4
INT41     LABEL    DWORD
          DD       FIRST
IVT       ENDS
```

```
MOV    WORD  PTR  IR41,     OFFSET   INT41_SR_P
MOV    WORD  PTR  IR41+2,   SEG      INT41_SR_P
```



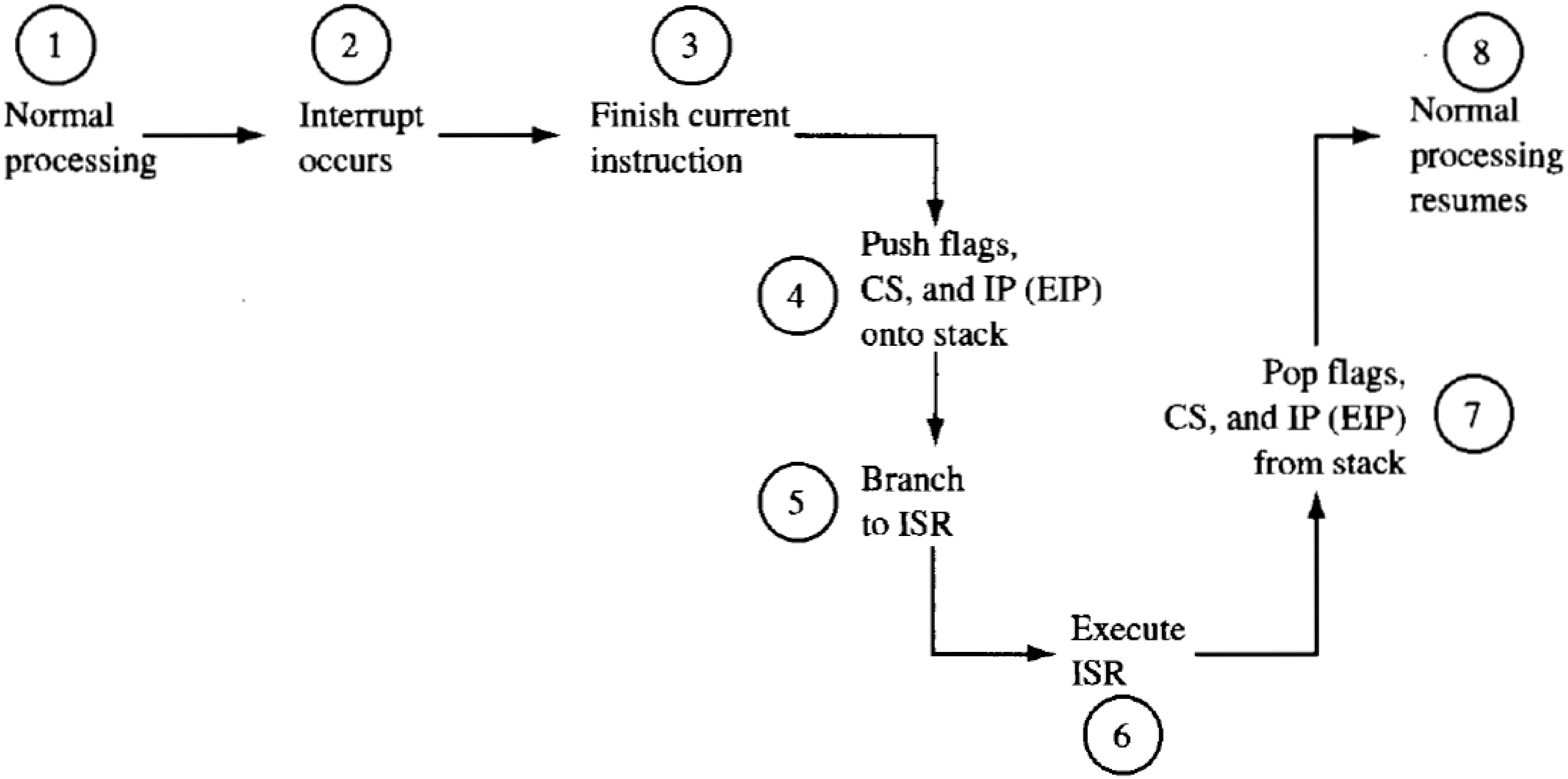
Odd Addresses	Even Addresses	Physical Address
		00000
…	…	…
05	00	104H
FE	00	106H
…	…	
		003FF

Example of Interrupt Service Routine

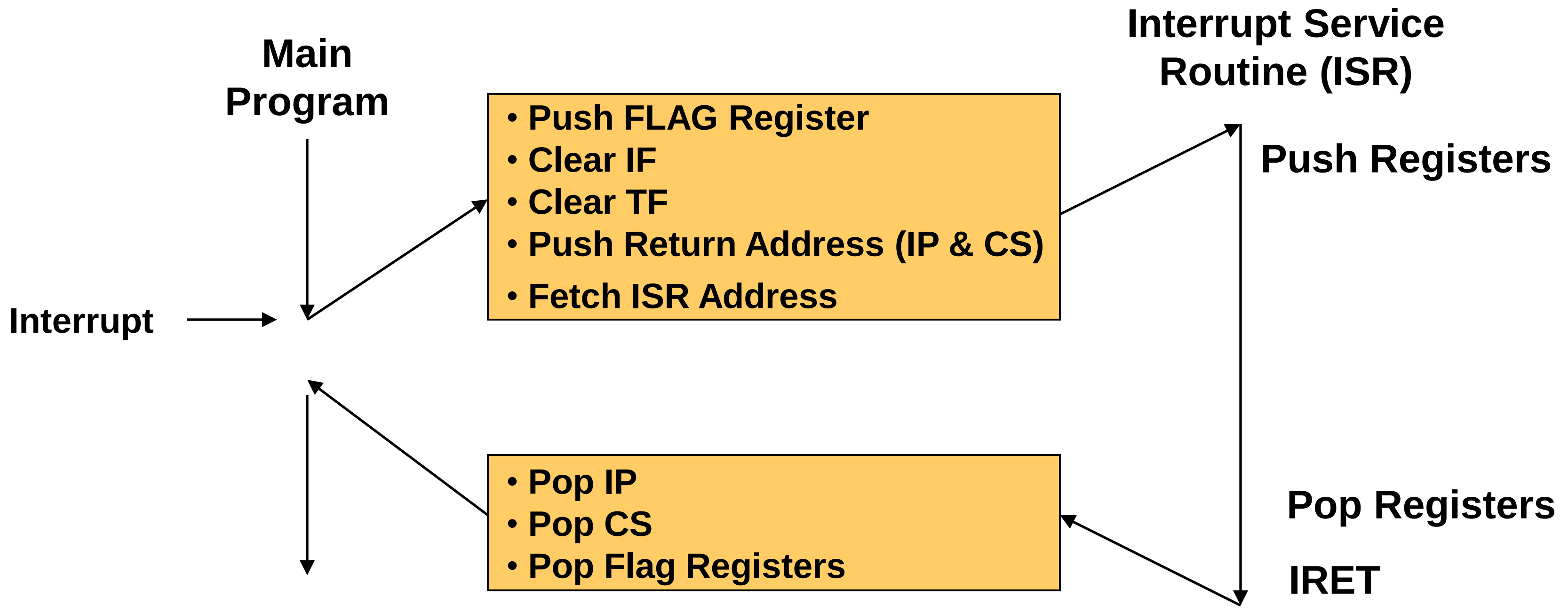
```
INT41_SR        SEGMENT  AT   0FE00H
                ASSUME  CS:INT41_SR, SS:STACK
                ORG        0500H
;
First                      LABEL    FAR
INT21_SR_PROC              PROC     FAR
; Pushing Registers
                PUSH        DX
                PUSH        AX
; Reading Port A of PPI
                MOV        DX, PortA
                IN         AL, DX
; Writing the value to Port B of PPI
                INC        DX
                OUT        DX, AL
; Popping Registers
                POP        AX
                POP        DX
                IRET
INT41_SR_PROC              ENDP
INT41_SR                   ENDS
```

Interrupt Processing

- When an interrupt occurs, normal processing is suspended while a special Interrupt Service Routine (ISR) is executed. Normal processing resumes when this routine is completed.


1
Normal processing
2
Interrupt occurs
3
Finish current instruction
4
Push flags, CS, and IP (EIP) onto stack
5
Branch to ISR
Execute ISR
6
Pop flags, CS, and IP (EIP) from stack
7
8
Normal processing resumes

Interrupt Processing



Main Program

Interrupt

- Push FLAG Register
- Clear IF
- Clear TF
- Push Return Address (IP & CS)
- Fetch ISR Address

Interrupt Service Routine (ISR)

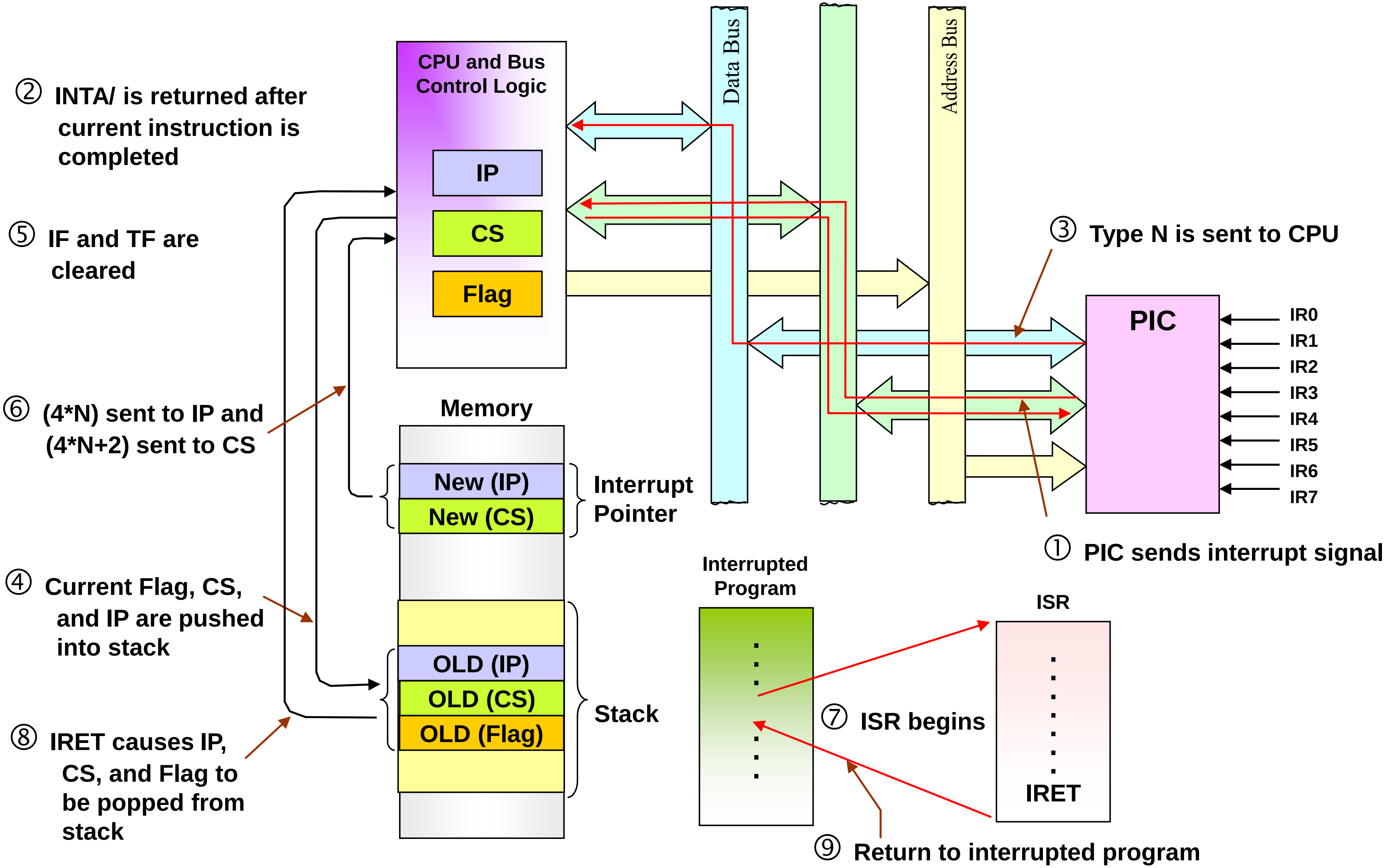
Push Registers

Pop Registers

IRET

- Pop IP
- Pop CS
- Pop Flag Registers

Sequence of Events During an Interrupt



① PIC sends interrupt signal

② INTA/ is returned after current instruction is completed

③ Type N is sent to CPU

④ Current Flag, CS, and IP are pushed into stack

⑤ IF and TF are cleared

⑥ (4*N) sent to IP and (4*N+2) sent to CS

⑦ ISR begins

⑧ IRET causes IP, CS, and Flag to be popped from stack

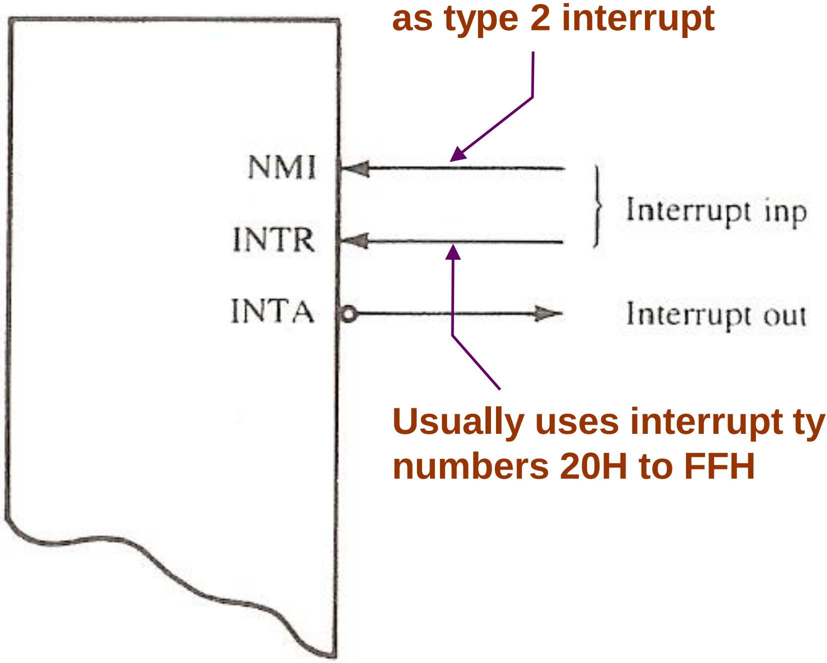
⑨ Return to interrupted program

When an Interrupt Occurs

- **Flags register** is pushed onto the stack.
- Both the interrupt (**IF**) and trap (**TF**) flags are cleared. This disables the INTR input and the trap from interrupting the interrupt being handled.
- The code segment register (**CS**) is pushed onto the stack.
- The instruction pointer (**IP**) is pushed onto the stack.
- The **interrupt vector** contents are fetched from the interrupt vector table and then placed into the IP and CS so that the next instruction executes at the interrupt service procedure addressed by the vector.

Hardware Interrupts

- The microprocessor has two hardware interrupt inputs:
  - □ **Non-Maskable Interrupt (NMI)**
  - □ **Interrupt Request (INTR)**



Internally decoded as type 2 interrupt

Usually uses interrupt types numbers 20H to FFH

Non-Maskable Interrupt (NMI)

- Whenever the **NMI** input is activated, a **type 2** interrupt occurs because **NMI** is internally decoded.
- The **NMI** is an **edge-triggered** input that requests an interrupt on the positive edge (0-to-1 transition).
- After a positive edge, the **NMI** pin must remain a logic 1 two T states at minimum.
- The NMI input is often used for major system faults such as power failures. Power failures are easily detected by monitoring the AC power line and causing an NMI interrupt whenever AC power drops out.

Interrupt Input (INTR)

- The **INTR** input is automatically *disabled* once it is accepted by the microprocessor and *re-enabled* by the **IRET** instruction at the end of the Interrupt Service Routine (ISR).

- The **INTR** input must be externally decoded to select a vector. Any **interrupt vector** can be chosen for the INTR pin, but we usually use an interrupt type number between **20H** and **FFH**.

- The microprocessor responds to the **INTR** input by pulsing the **INTA/** output in anticipation of receiving an interrupt vector type on data bus connections (**BD7-BD0**).

- There are **two INTA/ pulses** generated by the microprocessor that are used to insert the vector type number on the data bus.

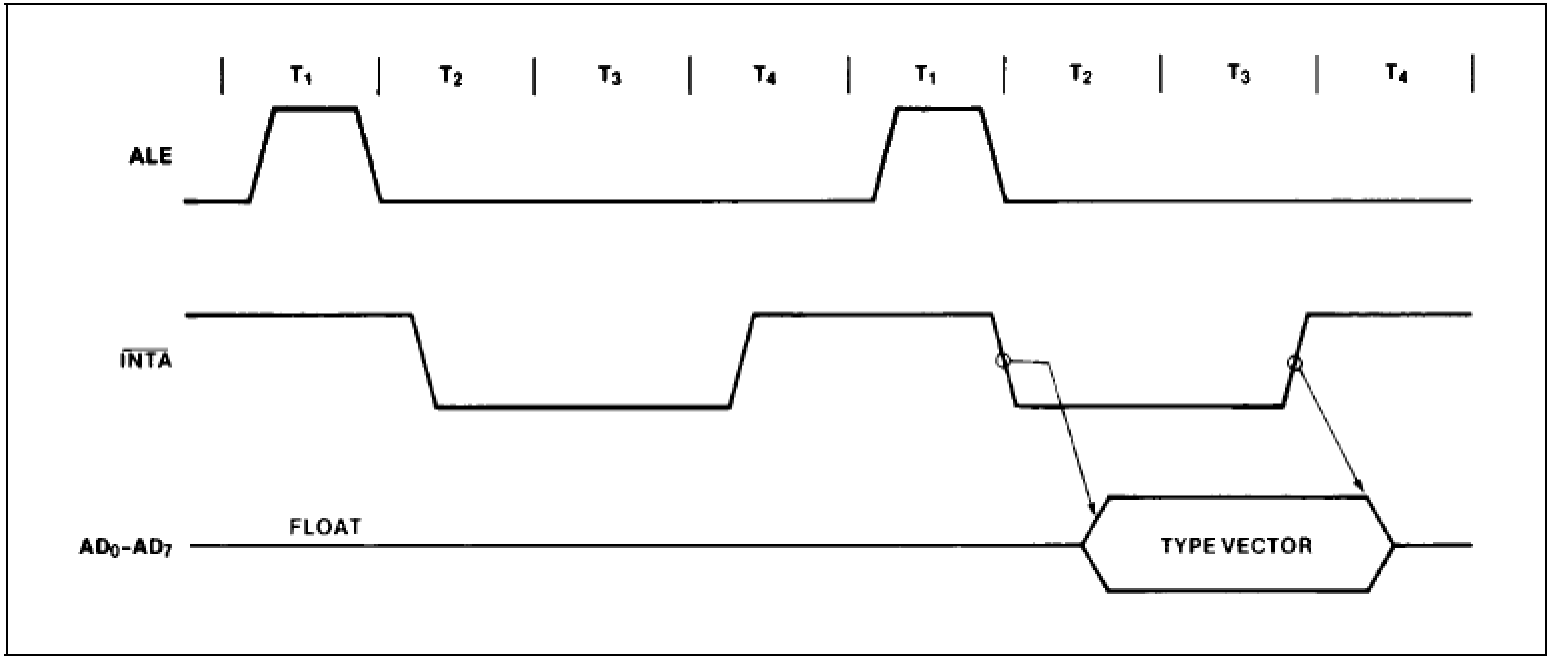
- The **INTR** can be disabled by a **CLI** instruction which clears the **IF**, and enabled by a **STI** instruction which sets the **IF**.

Interrupt Types

Name	Initiated by	Maskable	Trigger	Priority	Acknowledge Signal	Vector Table Address	Interrupt Latency
NMI	External Hardware	No	↑edge, hold 2T states min.	2	None	00008H-0000BH	Current Instruction + 51 T states
INTR	External Hardware	Yes via IF	High level until acknowledged	3	INTA/	n × 4	Current Instruction + 61 T states
INT n	Internal via Software	No	None	1	None	n × 4	51 T states
INT 3 (Breakpoint)	Internal via Software	No	None	1	None	0000CH-0000FH	52 T states
INTO	Internal via Software	No	None	1	None	00010H-00013H	53 T states
Divide-by-0	Internal via CPU	No	None	1	None	00000H-00003H	51 T states
Single-Step	Internal via CPU	Yes via TF	None	4	None	00004H-00007H	51 T states

All interrupt types cause the Flags, CS, and IP registers to be pushed onto the stack. In addition, the IF and TF are cleared.

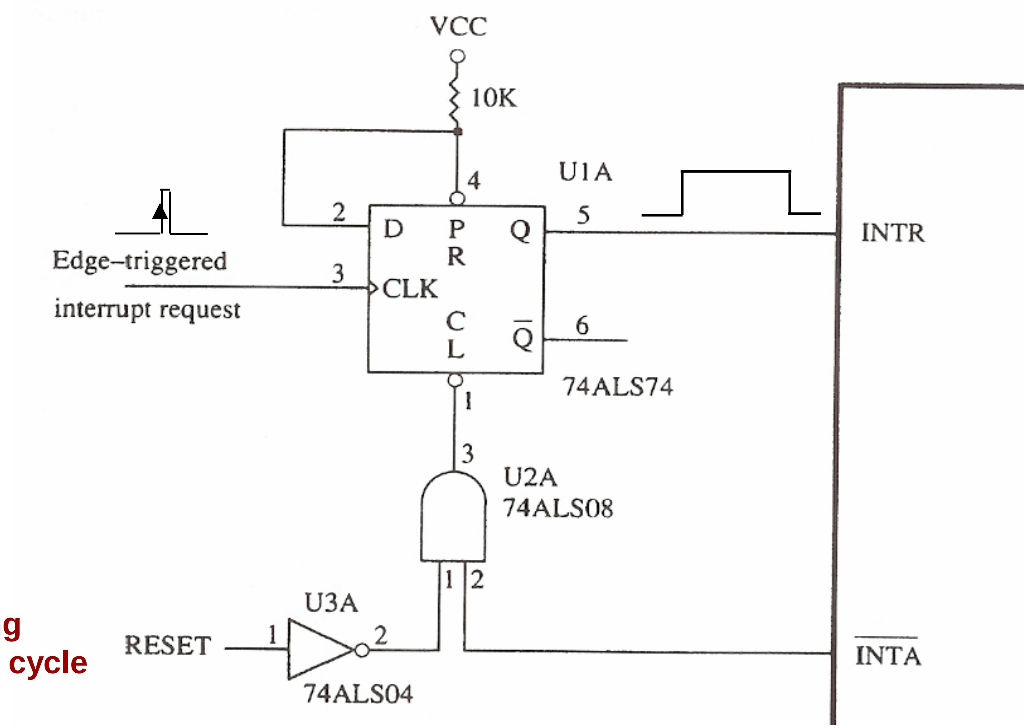
Interrupt Acknowledge Timing



Making the INTR Input Edge-Triggered

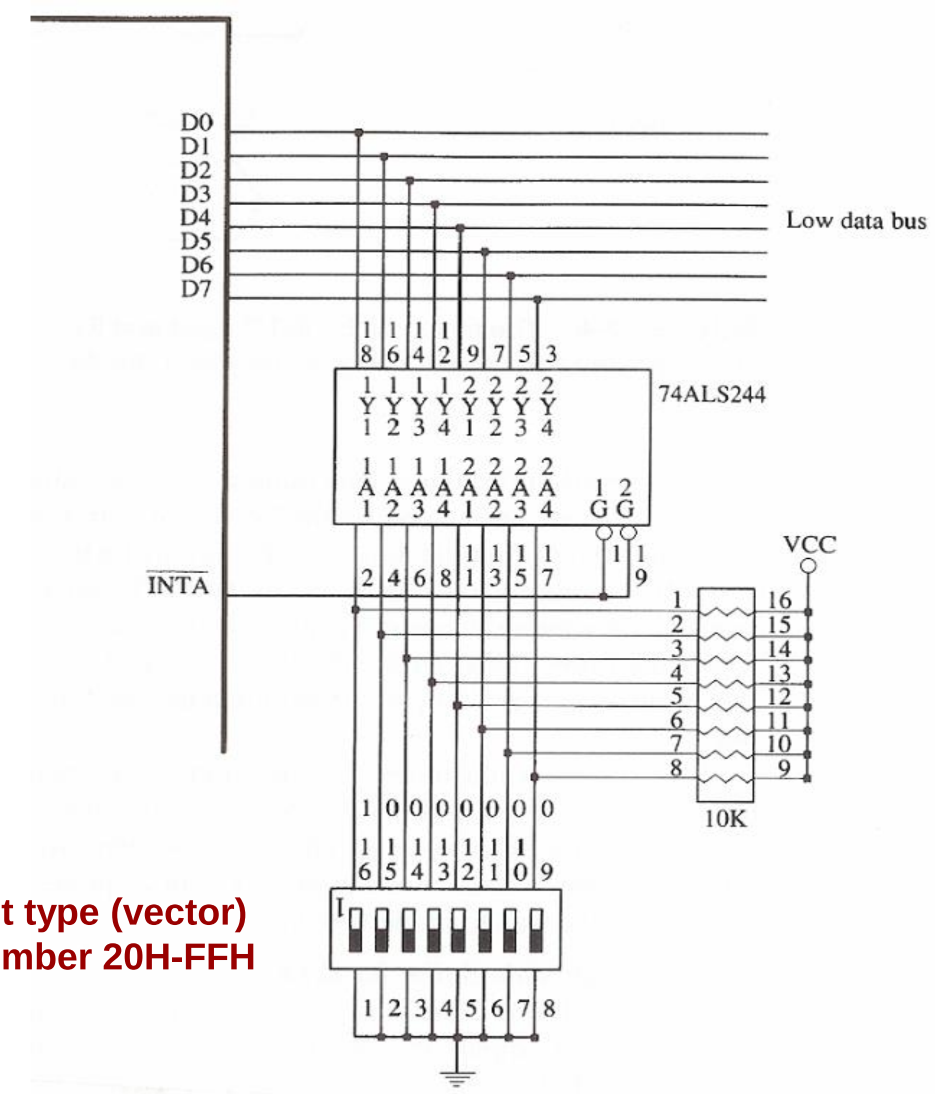
- Devices may produce narrow pulses as interrupt requests (suitable only for edge triggering).

- INTR must be kept high for enough time for processor to “see” it.



Prevent interrupts arriving during processor RESET cycle

Generating Interrupt Vector Type



Set type (vector) number 20H-FFH

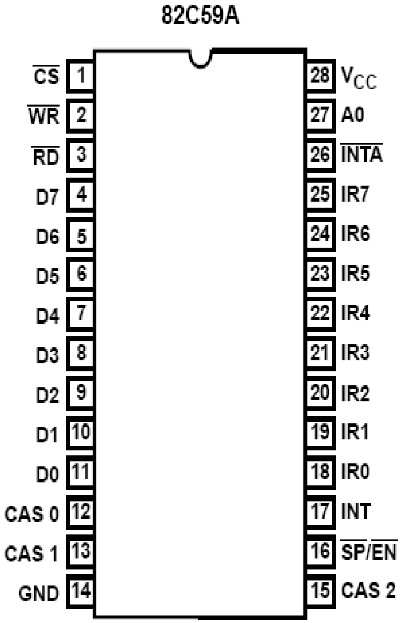
Structure of Assembly Program

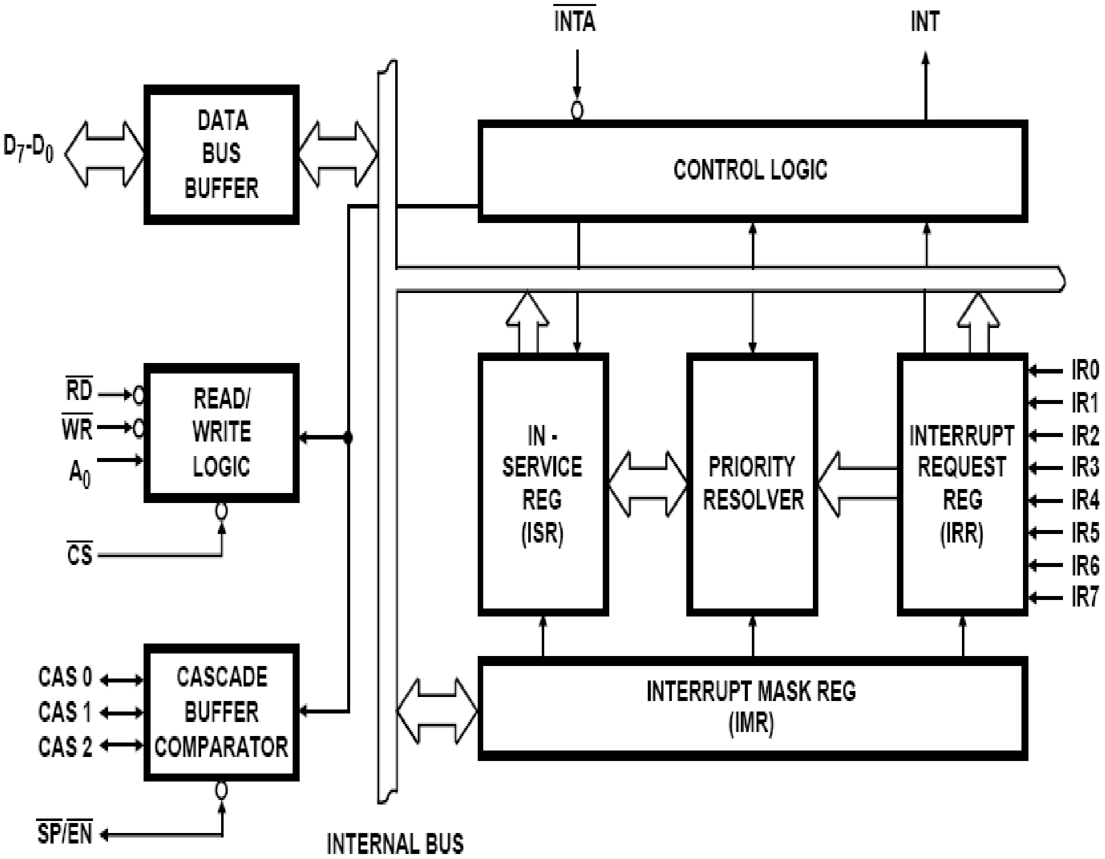
```
                        NAME          PROJECT
DATA                    SEGMENT  AT   40H
                        ORG           0H
VAR1                    DB            ?
DATA                    ENDS
;-------------------------------------------------------------------------
STACK                   SEGMENT  AT  50H
                        DW            10 DUP(?)
STK_TOP      LABEL      WORD
STACK                   ENDS
;-------------------------------------------------------------------------
INT_VEC_TABLE           SEGMENT AT    0H
                        ORG           21H*4
INT41                   LABEL         DWORD
                        DD            FIRST
INT_VEC_TABLE           ENDS
;-------------------------------------------------------------------------
EPROM                   SEGMENT  AT  0FE00H
                        ASSUME        CS:EPROM, SS:STACK
                        ORG           0H
BEGIN                   LABEL         FAR
;
                        ASSUME        DS:INT_VEC_TABLE
                        MOV           AX,INT_VEC_TABLE
                        MOV           DS,AX
                        MOV           WORD  PTR  INT41,      OFFSET  INT41_SR_PROC
                        MOV           WORD  PTR  INT41 + 2,  SEG     INT41_SR_PROC
                        ASSUME        DS:DATA
                        MOV           AX,DATA
                        MOV           DS,AX
                        ; INITIALIZATIONS
                        STI
AGAIN                   JMP           AGAIN
EPROM                   ENDS
```

Structure of Assembly Program

```
INT41_SR                                SEGMENT  AT   0FE00H
                        ASSUME          CS:INT41_SR, SS:STACK
                        ORG             0500H
First                   LABEL           FAR
INT41_SR_PROC           PROC            FAR
;
                        PUSH             DX
                        PUSH             AX

                        ; Core Service Routine for INT41

                        POP             AX
                        POP             DX
                        IRET
INT41_SR_PROC           ENDP
INT41_SR                                ENDS
;-------------------------------------------------------------------------------------------
CODE                    SEGMENT  AT   0FFFFH
                        ASSUME  CS:CODE,SS:STACK
                        ORG             0H
;
START:                                  CLI
                        MOV             AX,STACK
                        MOV             SS,AX
                        MOV             SP,OFFSET STK_TOP
                        JMP             BEGIN
CODE                    ENDS
END                     START
```

Priority Interrupt Controller (PIC)

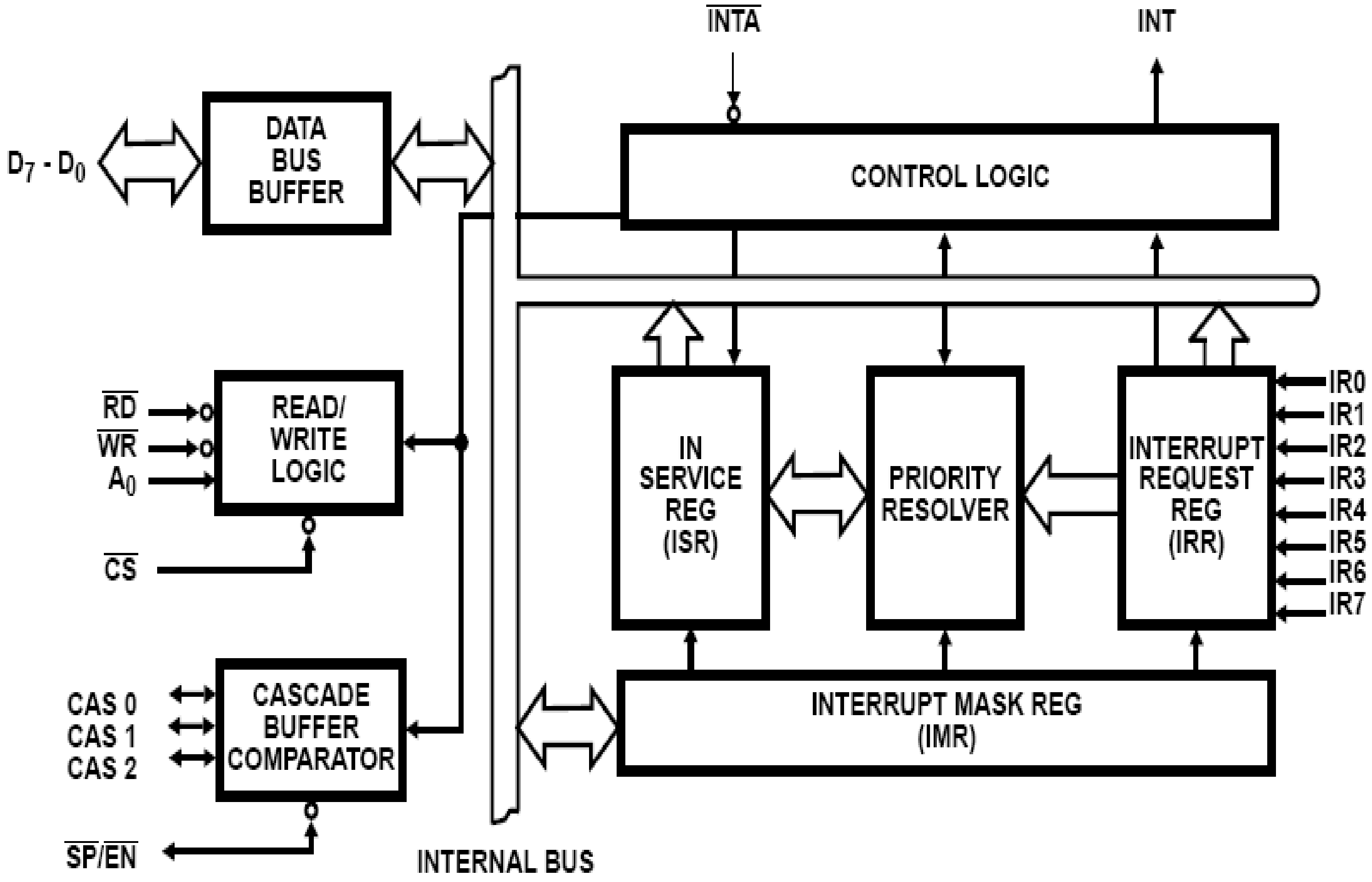




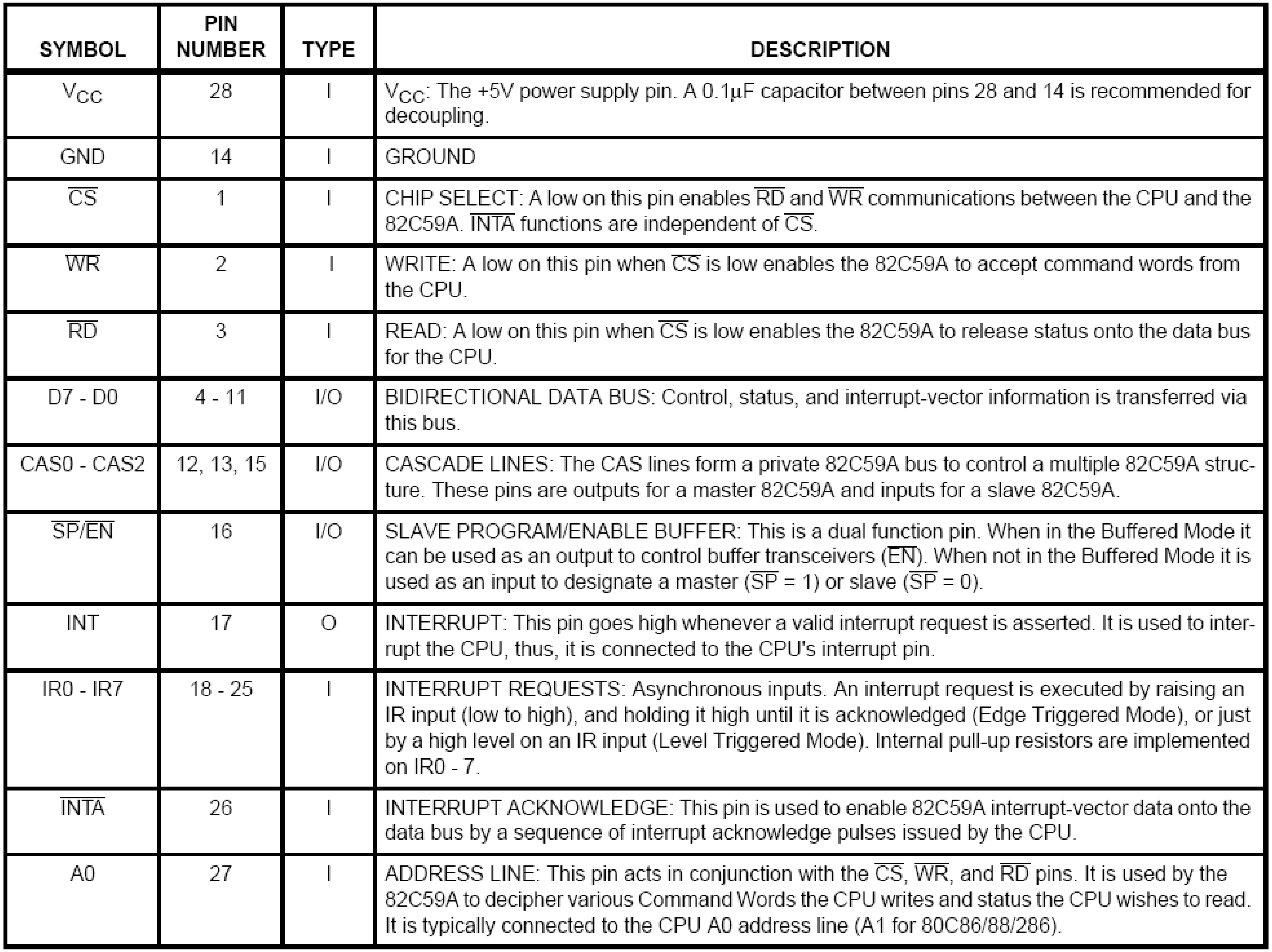
8259A: Main Features

- It accepts eight interrupt requests (expandable to 64 by cascading 8 slaves **PIC**s).
- It prioritizes the interrupt requests (selects the highest priority request for service).
- It includes the individual request mask capability.
- It issues a single interrupt request to the CPU.
- In response to the **INTA/**, it issues a unique type number (vector) for each interrupt request input. In addition, type numbers are programmable.
- It has variety of programmable modes of operations.

8259A Block Program

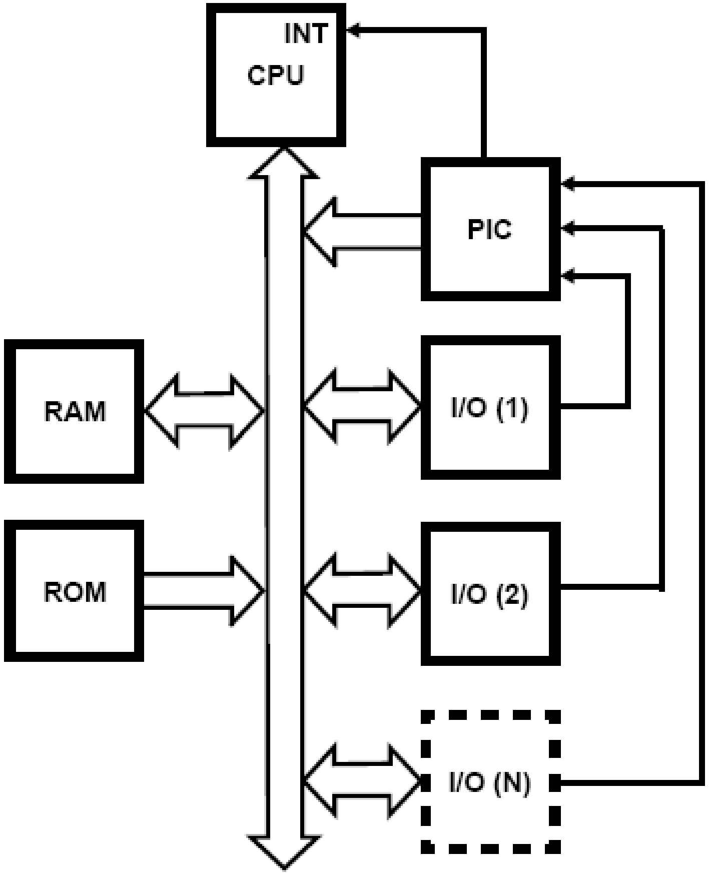


8259A Pin Description

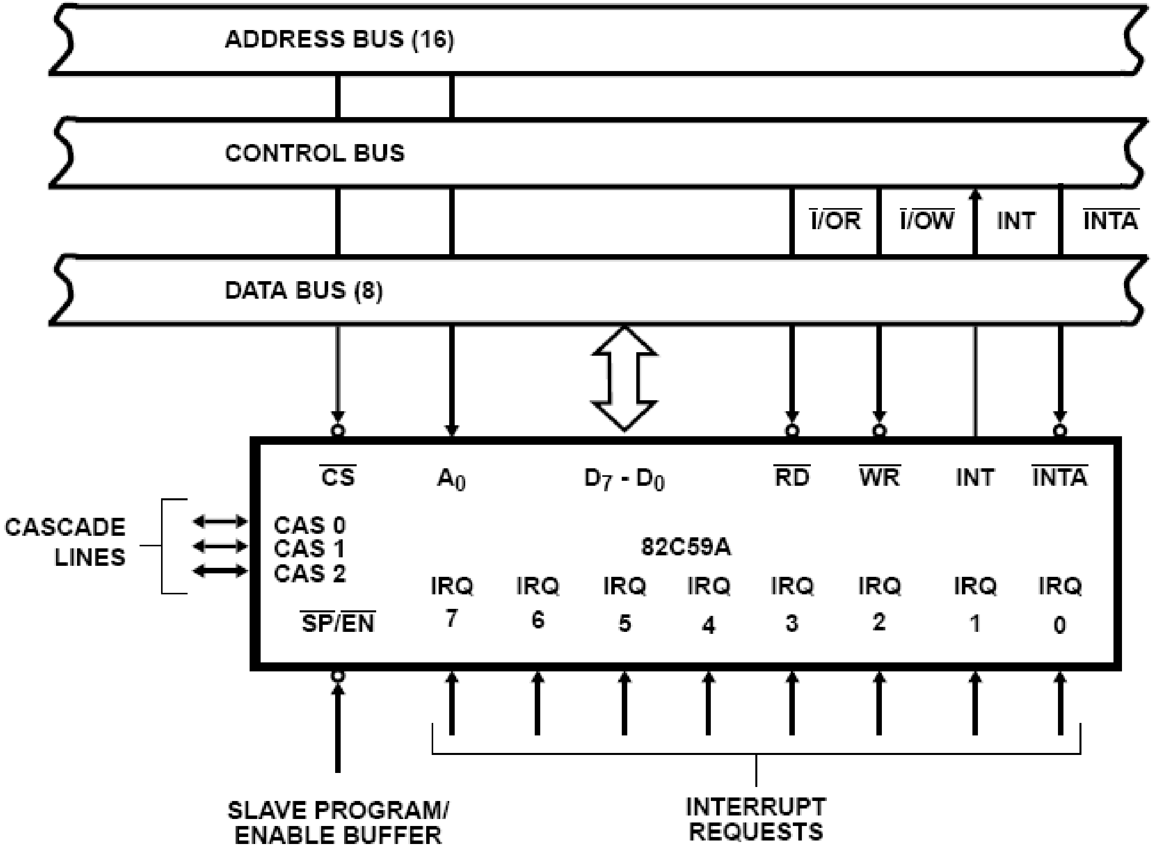


SYMBOL	PIN NUMBER	TYPE	DESCRIPTION
V_{CC}	28	I	V_{CC}: The +5V power supply pin. A 0.1μF capacitor between pins 28 and 14 is recommended for decoupling.
GND	14	I	GROUND
$\overline{CS}$	1	I	CHIP SELECT: A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communications between the CPU and the 82C59A. $\overline{INTA}$ functions are independent of $\overline{CS}$.
$\overline{WR}$	2	I	WRITE: A low on this pin when $\overline{CS}$ is low enables the 82C59A to accept command words from the CPU.
$\overline{RD}$	3	I	READ: A low on this pin when $\overline{CS}$ is low enables the 82C59A to release status onto the data bus for the CPU.
D7 - D0	4 - 11	I/O	BIDIRECTIONAL DATA BUS: Control, status, and interrupt-vector information is transferred via this bus.
CAS0 - CAS2	12, 13, 15	I/O	CASCADE LINES: The CAS lines form a private 82C59A bus to control a multiple 82C59A structure. These pins are outputs for a master 82C59A and inputs for a slave 82C59A.
$\overline{SP}/\overline{EN}$	16	I/O	SLAVE PROGRAM/ENABLE BUFFER: This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers ($\overline{EN}$). When not in the Buffered Mode it is used as an input to designate a master ($\overline{SP}$ = 1) or slave ($\overline{SP}$ = 0).
INT	17	O	INTERRUPT: This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus, it is connected to the CPU's interrupt pin.
IR0 - IR7	18 - 25	I	INTERRUPT REQUESTS: Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), and holding it high until it is acknowledged (Edge Triggered Mode), or just by a high level on an IR input (Level Triggered Mode). Internal pull-up resistors are implemented on IR0 - 7.
$\overline{INTA}$	26	I	INTERRUPT ACKNOWLEDGE: This pin is used to enable 82C59A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.
A0	27	I	ADDRESS LINE: This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. It is used by the 82C59A to decipher various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for 80C86/88/286).

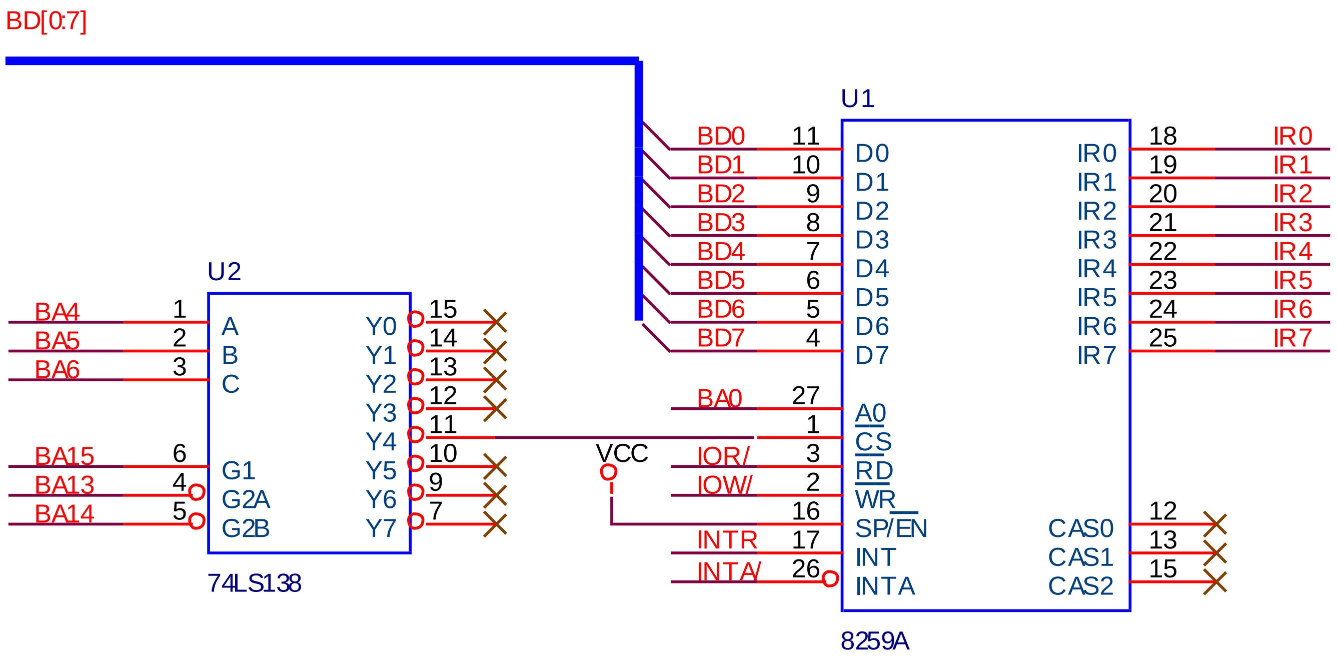
INTERRUPT METHOD



Interfacing 8259A to Standard Bus System



Interfacing 8259A to 8088 Microprocessor



8259A: Interrupt Sequence

- One or more of the **INTERRUPT REQUEST** lines (**IR7-IR0**) are raised high, setting the corresponding **IRR** bit(s).
- The 8259A evaluates these requests, and sends an **INT** to the CPU, if appropriate.
- The CPU acknowledges the **INT** and responds with an **INTA/** pulse.
- Upon receiving an **INTA/** from the CPU, the highest priority **ISR** bit is set, and the corresponding **IRR** bit is reset. The 8259A does not drive the data bus this cycle.
- The 8088/8086 will initiate a second **INTA/** pulse. During this pulse, the 8259A releases an 8-bit pointer onto the Data Bus where it is read by the CPU.
- This completes the interrupt cycle. In the **AEOI** mode, the **ISR** bit is reset at the end of the second **INTA/** pulse. Otherwise, the **ISR** bit remains set until the appropriate **EOI** command is issued at the end of the Interrupt Service Routine (**ISR**).

8259A: Interrupt Request Register (IRR)

- The **IRR** is a transparent latch. Bits of **IRR** are frozen by the **FREEZE/** signal starting at the beginning of first **INTA/** pulse till the end of the second **INTA/** pulse.
- Whatever interrupt requests arrive at the **IRR** will be latched transparently.
- If the interrupt request is not masked as per the **Interrupt Mask Register** (**IMR**), it will reach to the **Priority Resolver** (**PR**).
- If new request's priority is higher than the interrupt under service, **INT** signal is raised by the **PR** to make a request to the CPU.
- After some time, the CPU sends the first **INTA/** pulse. Now the requests in **IRR** are frozen, and the highest priority interrupt is identified by the **PR**. The corresponding bit in the **ISR** is set, and the corresponding bit of **IRR** is cleared.
- During the second **INTA/** pulse, the **vector type** is sent out, and then the **INT** signal is lowered.

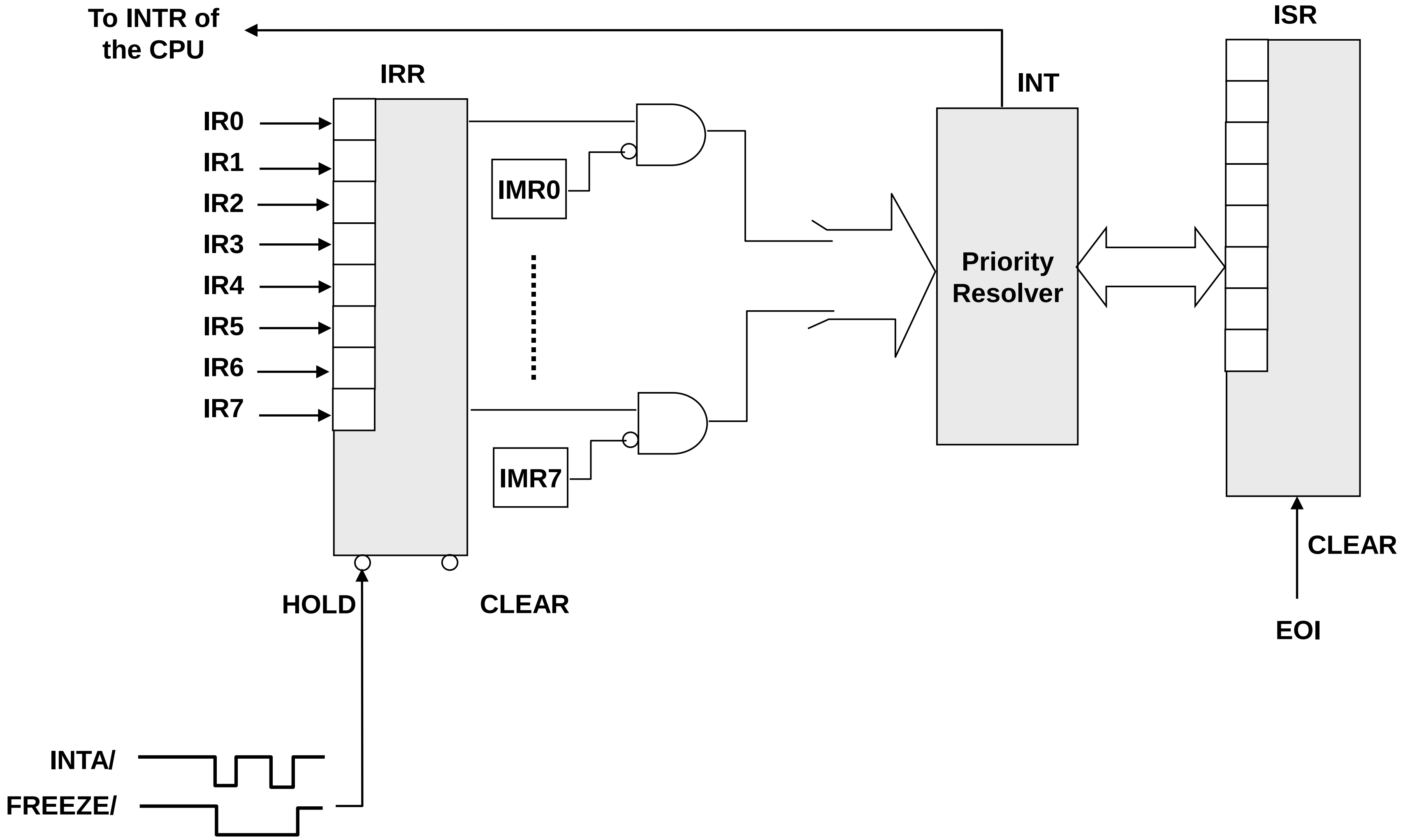
8259A: In-Service Register (ISR)

- The **ISR** records the requests currently being serviced.
- It enables the 8259A to check whether any of the incoming new requests has higher priority than the priority of the currently in-service requests. If so, the 8259A will again raise the **INT** signal and wait for **INTA/**.
- The **ISR** bit(s) get cleared:
  - At the end of the **ISR,** there should be an instruction to send **End Of Interrupt** (**EOI**) command to the 8259A. Then the highest priority **ISR** bit gets reset.

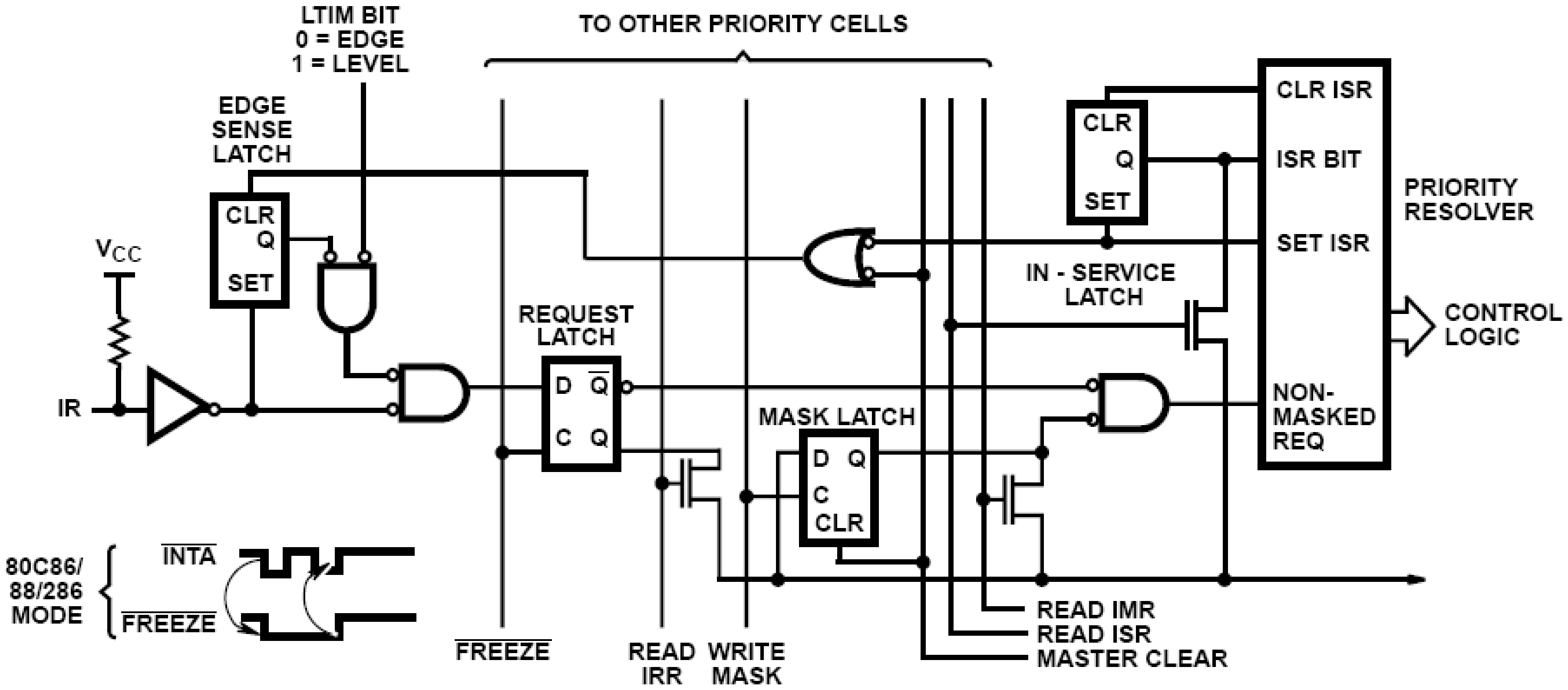
    Note:

    - If **End Of Interrupt** (**EOI**) is not issued at the end of **ISR**, all interrupt requests of equal or lower priority remain blocked until the **ISR** bit is cleared.
  - For nesting of interrupts, **ISR** bit is reset at the end of the second **INTA/** pulse (It is a programmable feature called **Automatic End Of Interrupt** (**AEOI**)).

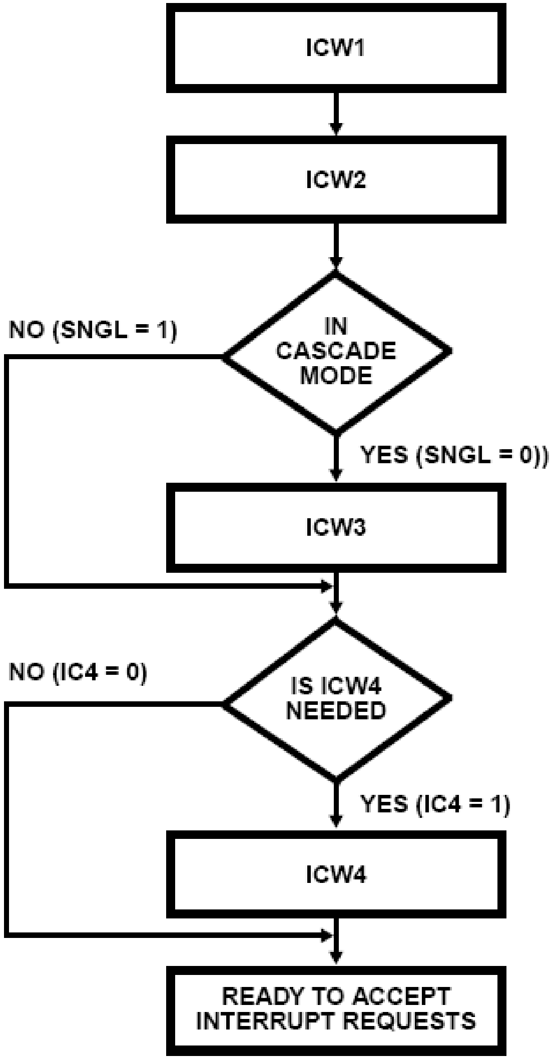
IRR, IMR, PR, and ISR



PRIORITY CELL



8259A Initialization Sequence



ICW1

ICW2

IN CASCADE MODE

NO (SNGL = 1)

YES (SNGL = 0))

ICW3

IS ICW4 NEEDED

NO (IC4 = 0)

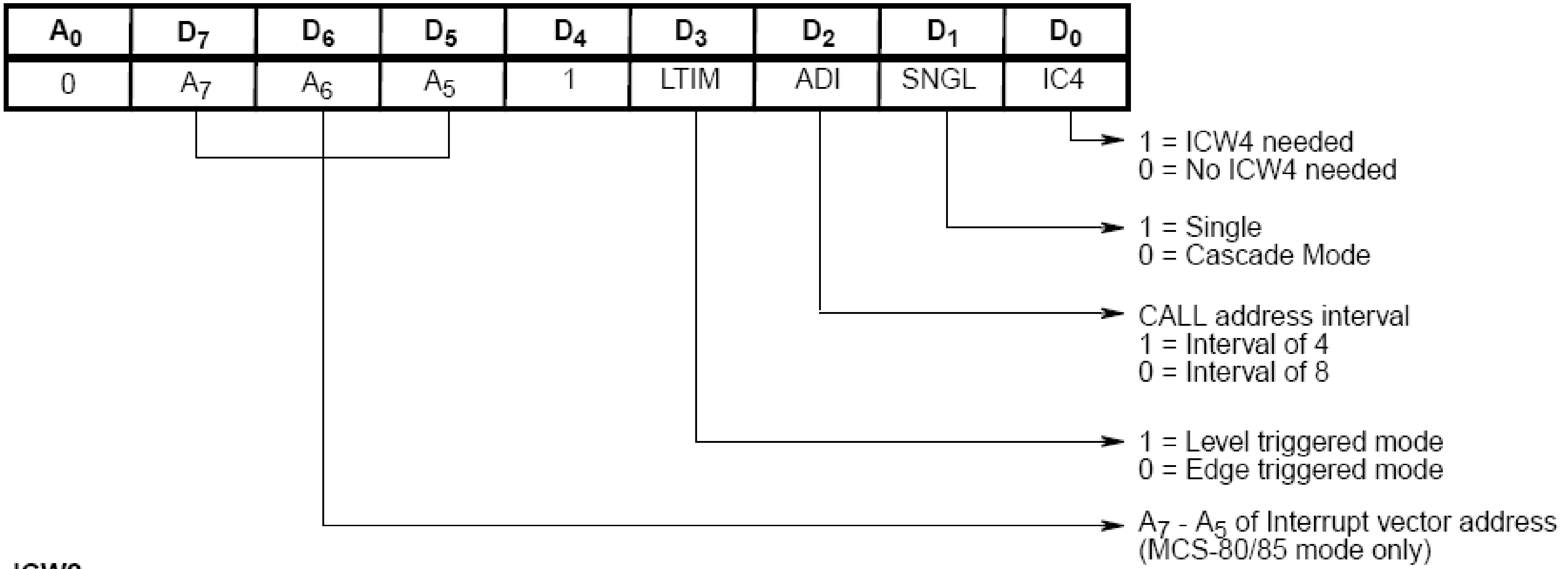
YES (IC4 = 1)

ICW4

READY TO ACCEPT INTERRUPT REQUESTS

8259A INITIALIZATION COMMAND WORD

ICW1



A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC4

1 = ICW4 needed
0 = No ICW4 needed

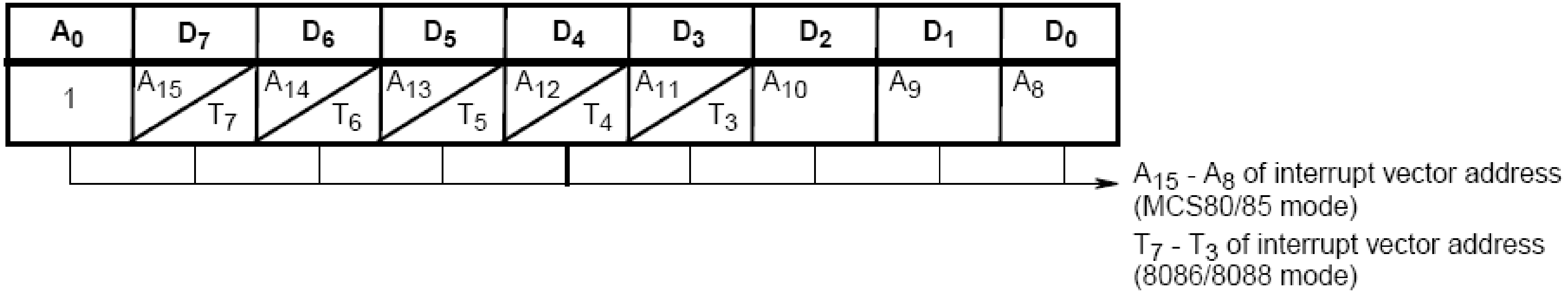
1 = Single
0 = Cascade Mode

CALL address interval
1 = Interval of 4
0 = Interval of 8

1 = Level triggered mode
0 = Edge triggered mode

A_7 - A_5 of Interrupt vector address
(MCS-80/85 mode only)

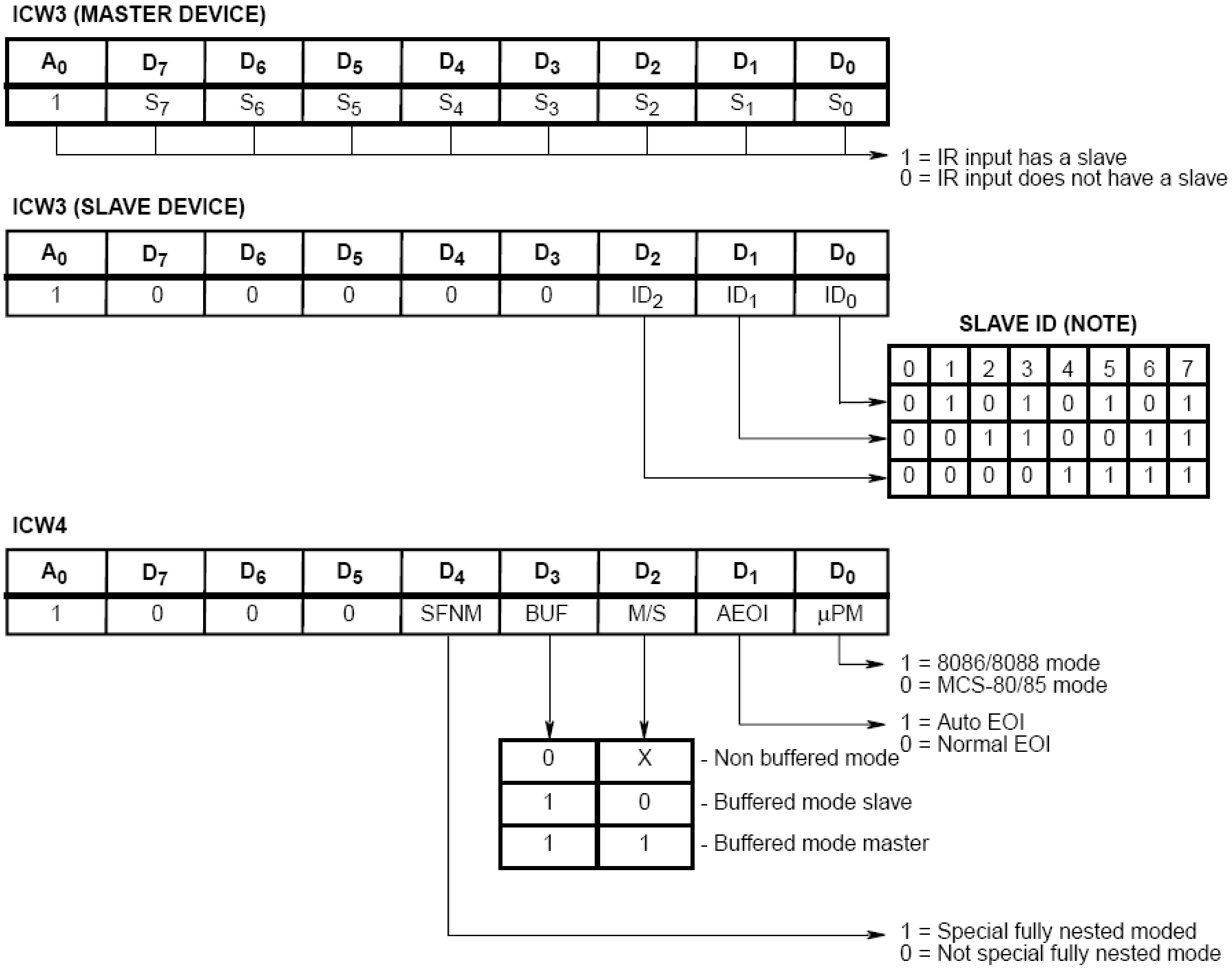
ICW2



A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	A_{15} / T_7	A_{14} / T_6	A_{13} / T_5	A_{12} / T_4	A_{11} / T_3	A_{10}	A_9	A_8

A_{15} - A_8 of interrupt vector address
(MCS80/85 mode)
T_7 - T_3 of interrupt vector address
(8086/8088 mode)

8259A INITIALIZATION COMMAND WORD



ICW3 (MASTER DEVICE)

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

1 = IR input has a slave
0 = IR input does not have a slave

ICW3 (SLAVE DEVICE)

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

SLAVE ID (NOTE)

0	1	2	3	4	5	6	7
0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1

ICW4

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	SFNM	BUF	M/S	AEOI	μPM

1 = 8086/8088 mode
0 = MCS-80/85 mode

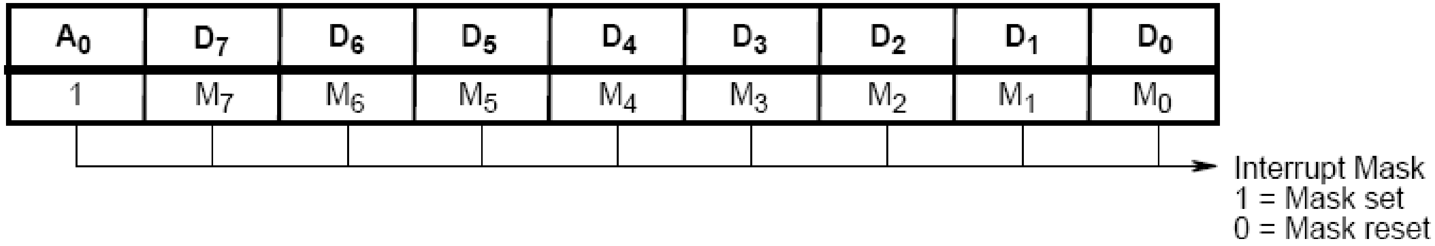
1 = Auto EOI
0 = Normal EOI

0	X	- Non buffered mode
1	0	- Buffered mode slave
1	1	- Buffered mode master

1 = Special fully nested moded
0 = Not special fully nested mode

8259A OPERATION COMMAND WORD

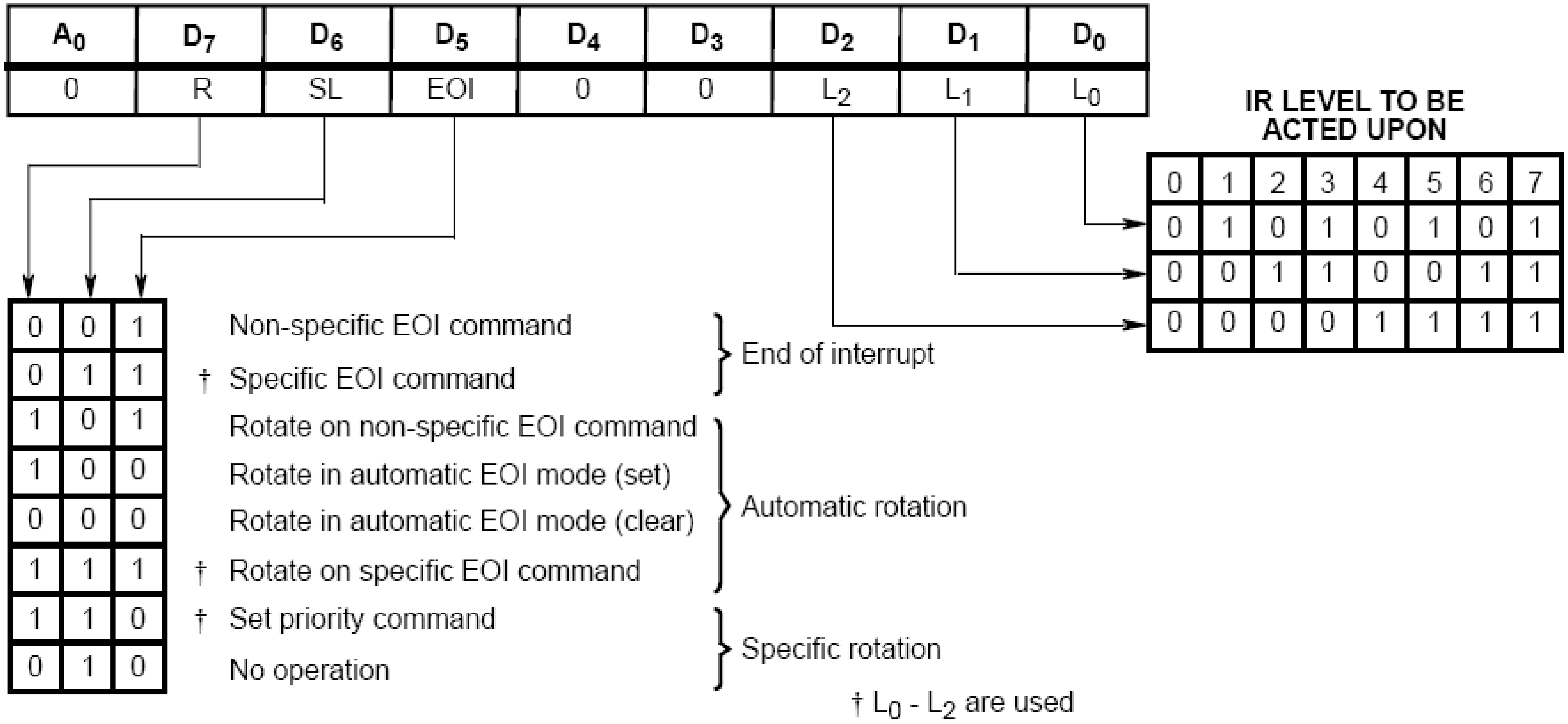
OCW1



A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

Interrupt Mask
1 = Mask set
0 = Mask reset

OCW2



A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	R	SL	EOI	0	0	L_2	L_1	L_0

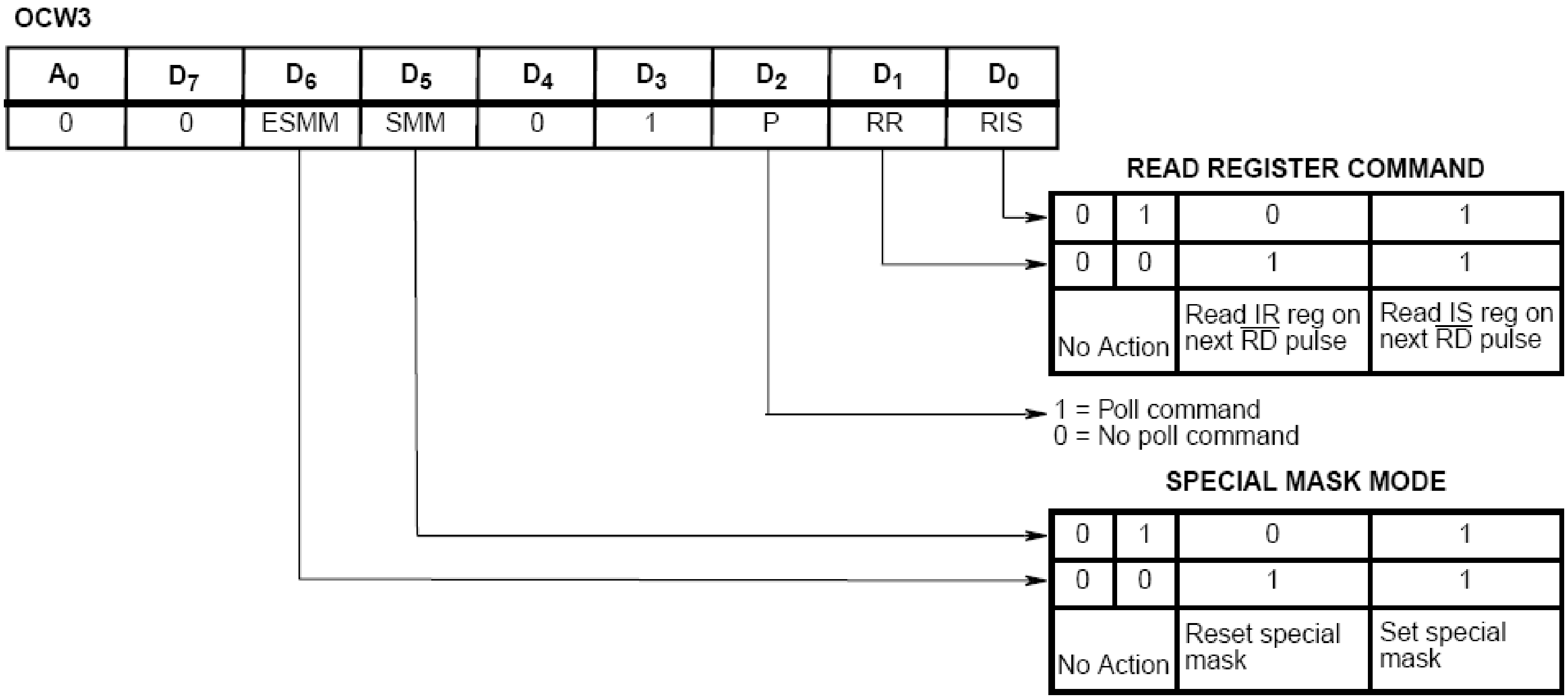
IR LEVEL TO BE ACTED UPON

0	1	2	3	4	5	6	7
0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1

0	0	1	Non-specific EOI command	End of interrupt
0	1	1	† Specific EOI command	
1	0	1	Rotate on non-specific EOI command	Automatic rotation
1	0	0	Rotate in automatic EOI mode (set)	
0	0	0	Rotate in automatic EOI mode (clear)	
1	1	1	† Rotate on specific EOI command	
1	1	0	† Set priority command	Specific rotation
0	1	0	No operation	

† L_0 - L_2 are used

8259A OPERATION COMMAND WORD



OCW3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	ESMM	SMM	0	1	P	RR	RIS

READ REGISTER COMMAND

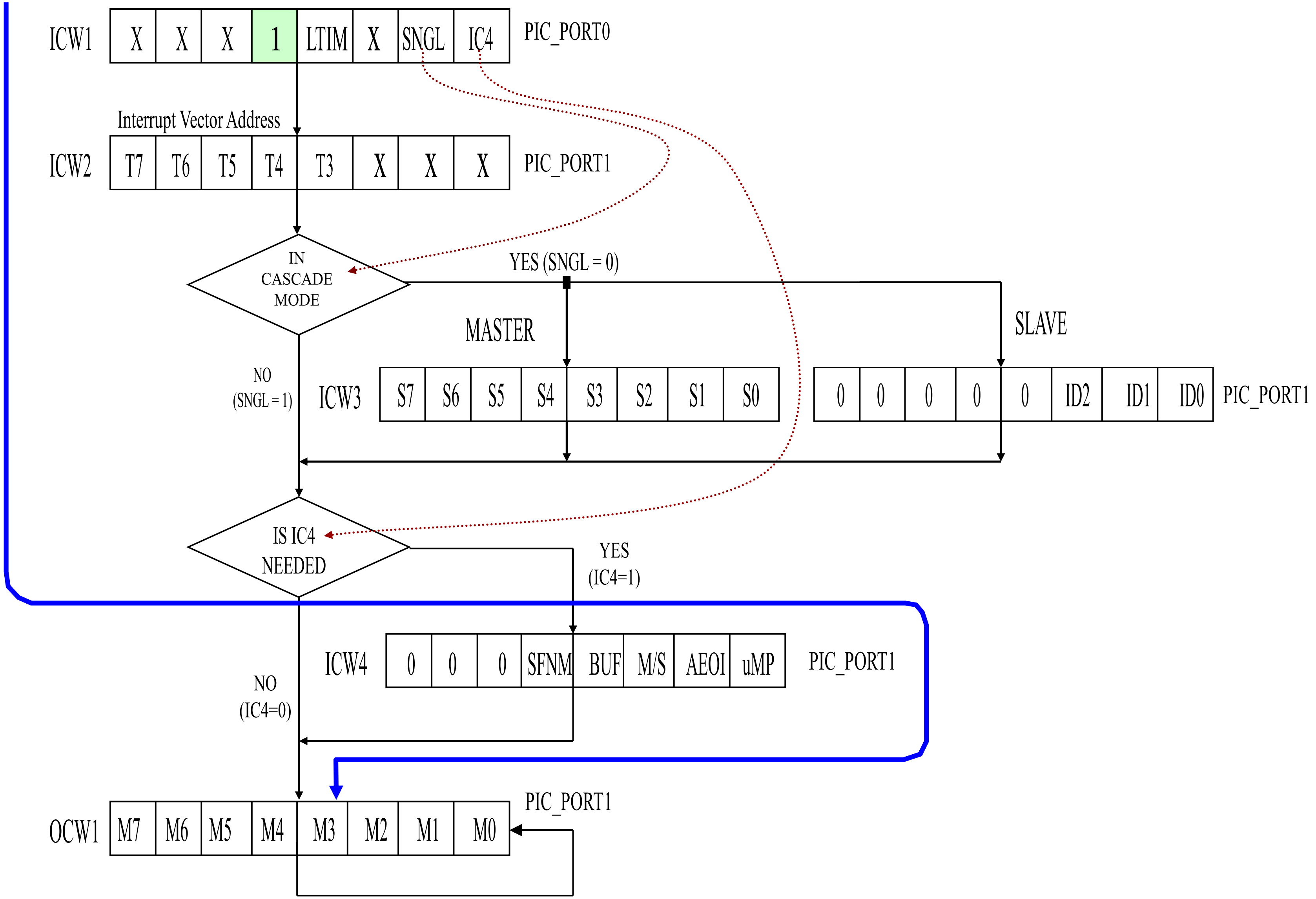
0	1	0	1
0	0	1	1
No Action		Read IR reg on next $\overline{RD}$ pulse	Read IS reg on next $\overline{RD}$ pulse

1 = Poll command
0 = No poll command

SPECIAL MASK MODE

0	1	0	1
0	0	1	1
No Action		Reset special mask	Set special mask

Programming the 8259A



ICW1	X	X	X	1	LTIM	X	SNGL	IC4	PIC_PORT0

Interrupt Vector Address

ICW2	T7	T6	T5	T4	T3	X	X	X	PIC_PORT1

IN CASCADE MODE

YES (SNGL = 0)

MASTER

SLAVE

NO (SNGL = 1)

ICW3	S7	S6	S5	S4	S3	S2	S1	S0

0	0	0	0	0	ID2	ID1	ID0	PIC_PORT1

IS IC4 NEEDED

YES (IC4=1)

NO (IC4=0)

ICW4	0	0	0	SFNM	BUF	M/S	AEOI	uMP	PIC_PORT1

PIC_PORT1

OCW1	M7	M6	M5	M4	M3	M2	M1	M0

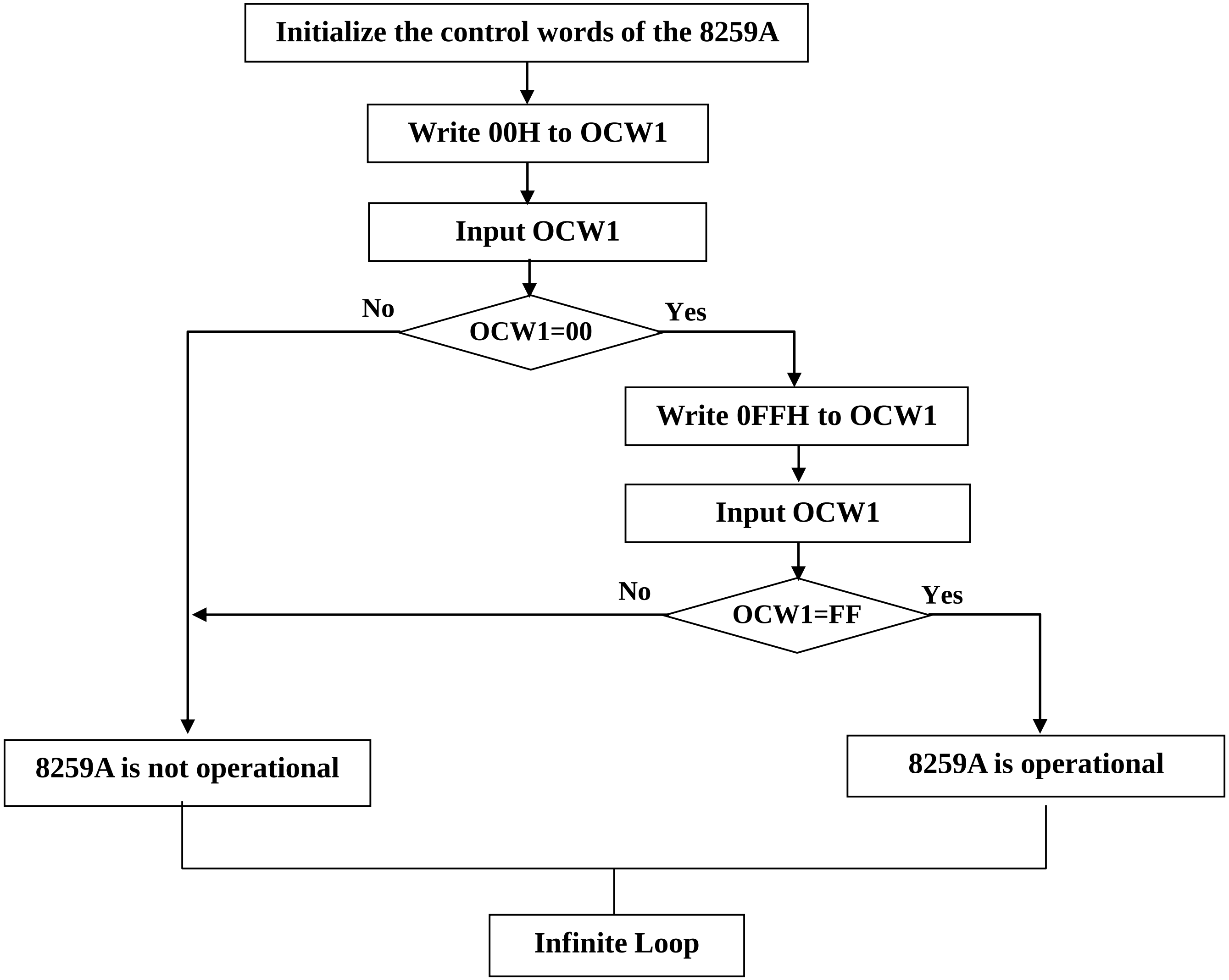
Programming the 8259A

ICW1	X	X	X	1	LTIM	X	SNGL	IC4	PIC_PORT0
OCW2	R	SL	EOI	0	0	L2	L1	L0	PIC_PORT0
OCW3	0	ESMM	SMM	0	1	P	RR	RIS	PIC_PORT0

Programming the 8259A

```
; ICW1 Initialization (IC4 Needed, Single, Edge Triggered Mode)
                    MOV       DX, PIC_PORT0
                    MOV       AL, 00010011B
                    OUT       DX, AL

; ICW2 Initialization (Interrupt Vector Address)
                    INC       DX
                    MOV       AL, VECTOR
                    OUT       DX, AL

; ICW4 Initialization (8086/8088 Mode, Normal EOI, Non-Buffered Mode,
; Not Special Fully Nested Mode)
                    MOV       AL, 00000001B
                    OUT       DX, AL


; OCW1 Initialization (Unmask IR1 Only)
                    MOV       AL, 11111101B
                    OUT       DX, AL
```

Testing the 8259A



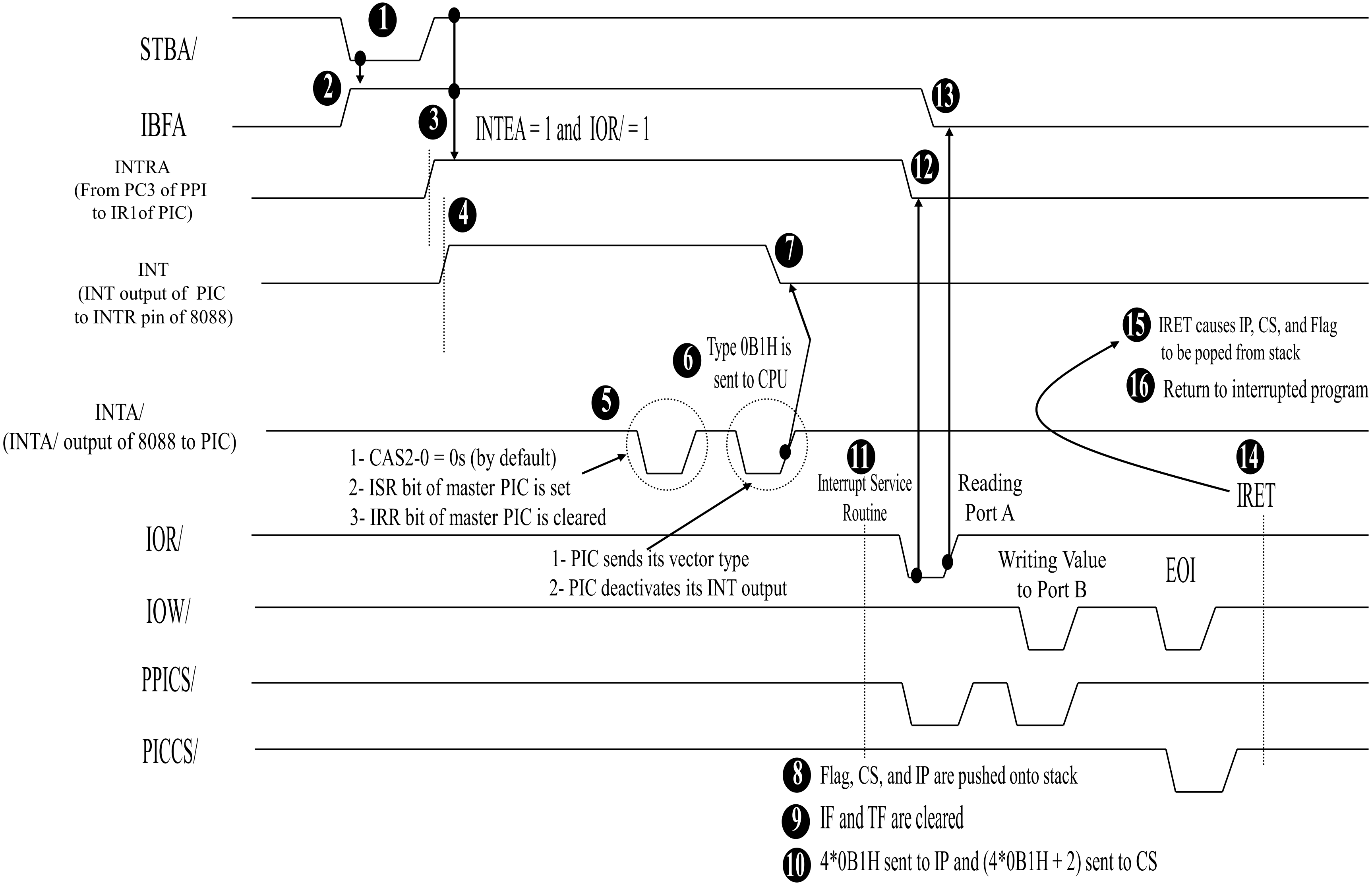
Testing the 8259A

```
MASK0   EQU     00H
MASK1   EQU     0FFH
;
        MOV     AL, MASK0
        OUT     DX, AL              ; Reset all mask bits in IMR
        IN      AL, DX              ; Read the mask register
        OR      AL, AL              ; Set zero flag
        JNZ     ERR
;
        MOV     AL, MASK1
        OUT     DX, AL              ; Set all mask bits in IMR
        IN      AL, DX              ; Read the mask register
        INC     AL                  ; Set zero flag
        JNZ     ERR
NO_ERR:
        ; No Error Handling Code
ERR:
        ; Error Handling Code
```

Reading IMR, ISR, and IRR

```
; To read IMR, simply:
                MOV       DX, PIC_PORT1
                IN        AL, DX
;-----------------------------------------------------------------------------
; To read ISR
                MOV       DX, PIC_PORT0
                MOV       AL, 00001011B                  ; Select ISR using OCW3
                OUT       DX, AL
                IN        AL, DX

; No need to write an OCW3 before every ISR read, as long as the ISR has been
; previously selected by OCW3
;-----------------------------------------------------------------------------
; To read IRR
                MOV       DX, PIC_PORT0
                MOV       AL, 00001010B                  ; Select IRR using OCW3
                OUT       DX, AL
                IN        AL, DX
; No need to write an OCW3 before every IRR read, as long as the IRR has been
; previously selected by OCW3
```

Programming the 8259A



❶

❷

❸ INTEA = 1 and IOR/ = 1

❹

❺

1- CAS2-0 = 0s (by default)
2- ISR bit of master PIC is set
3- IRR bit of master PIC is cleared

❻ Type 0B1H is
sent to CPU

1- PIC sends its vector type
2- PIC deactivates its INT output

❼

❽ Flag, CS, and IP are pushed onto stack

❾ IF and TF are cleared

❿ 4*0B1H sent to IP and (4*0B1H + 2) sent to CS

⓫ Interrupt Service
Routine

⓬

Reading
Port A

⓭

Writing Value
to Port B

EOI

⓮ IRET

⓯ IRET causes IP, CS, and Flag
to be poped from stack

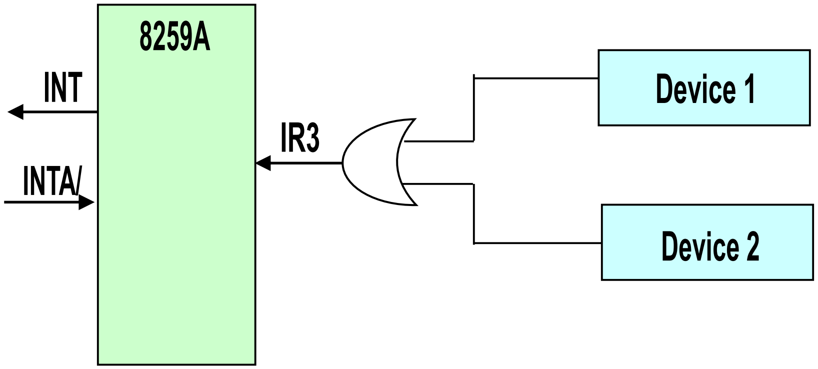
⓰ Return to interrupted program

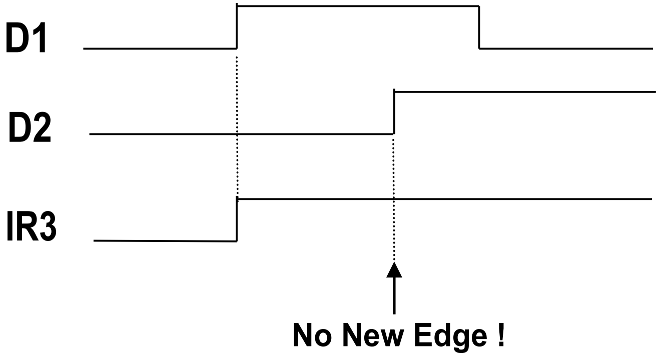
Edge Sensitive versus Level Sensitive Interrupt Request Input

- **Case 1: Interrupt Request (IR) is too long**

  CPU serves the interrupt and at the end of ISR, EOI is issued to clear ISR bit. Since the IR is still active, it appears like a new interrupt request to the CPU. Therefore, to handle this situation, 8259A should be programmed in Edge Triggered Mode.

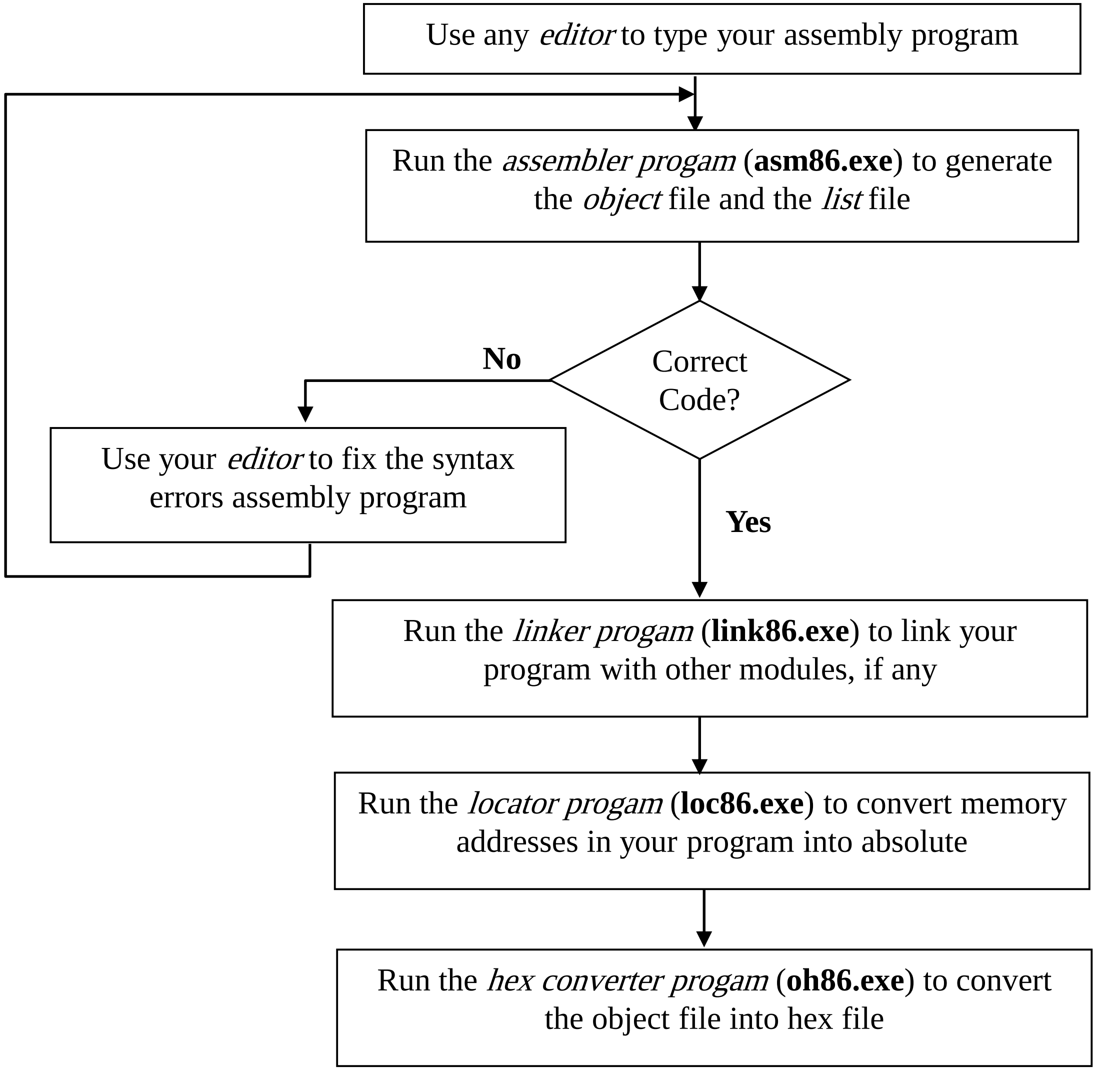
- **Case 2: Multiple Devices on a Single Interrupt Request**





Tutorial

Program Development Tools



Use any *editor* to type your assembly program

Run the *assembler progam* (**asm86.exe**) to generate the *object* file and the *list* file

Correct Code?

No

Use your *editor* to fix the syntax errors assembly program

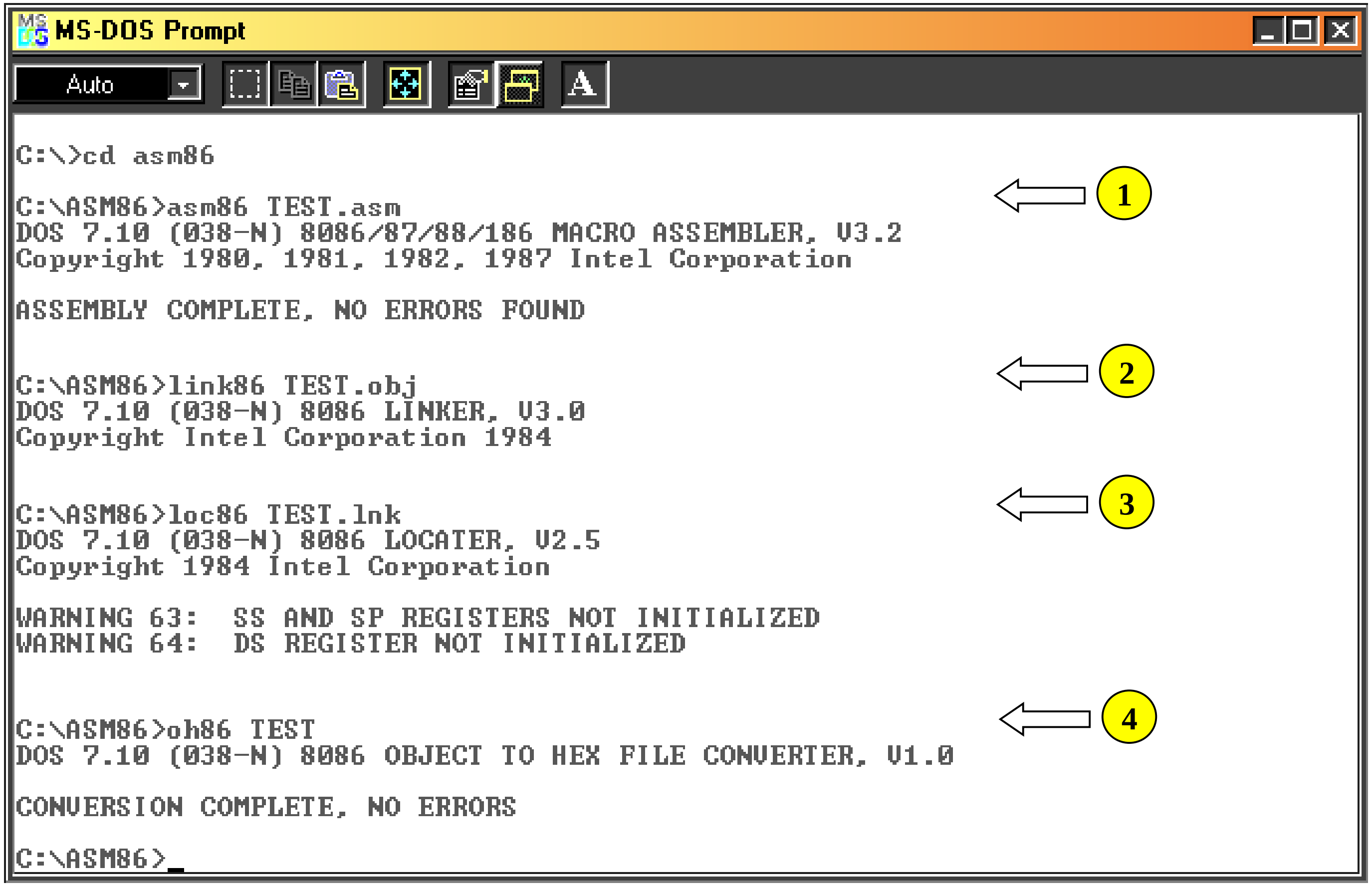
Yes

Run the *linker progam* (**link86.exe**) to link your program with other modules, if any

Run the *locator progam* (**loc86.exe**) to convert memory addresses in your program into absolute

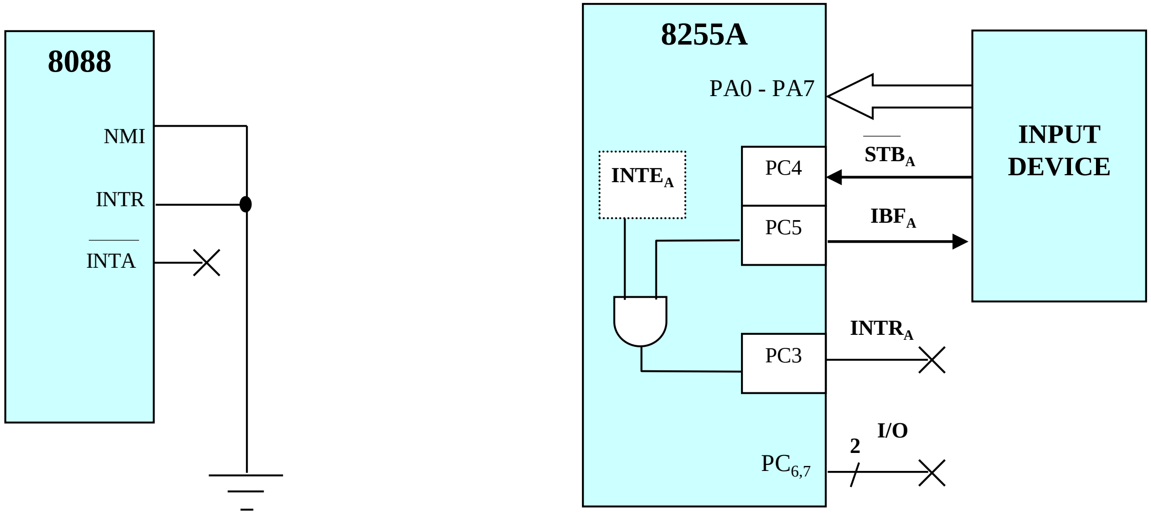
Run the *hex converter progam* (**oh86.exe**) to convert the object file into hex file

Program Development Tools



```
C:\>cd asm86

C:\ASM86>asm86 TEST.asm
DOS 7.10 (038-N) 8086/87/88/186 MACRO ASSEMBLER, V3.2
Copyright 1980, 1981, 1982, 1987 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND


C:\ASM86>link86 TEST.obj
DOS 7.10 (038-N) 8086 LINKER, V3.0
Copyright Intel Corporation 1984


C:\ASM86>loc86 TEST.lnk
DOS 7.10 (038-N) 8086 LOCATER, V2.5
Copyright 1984 Intel Corporation

WARNING 63:  SS AND SP REGISTERS NOT INITIALIZED
WARNING 64:  DS REGISTER NOT INITIALIZED


C:\ASM86>oh86 TEST
DOS 7.10 (038-N) 8086 OBJECT TO HEX FILE CONVERTER, V1.0

CONVERSION COMPLETE, NO ERRORS

C:\ASM86>_
```

Example 1: Programmed I/O Technique (Hardware)



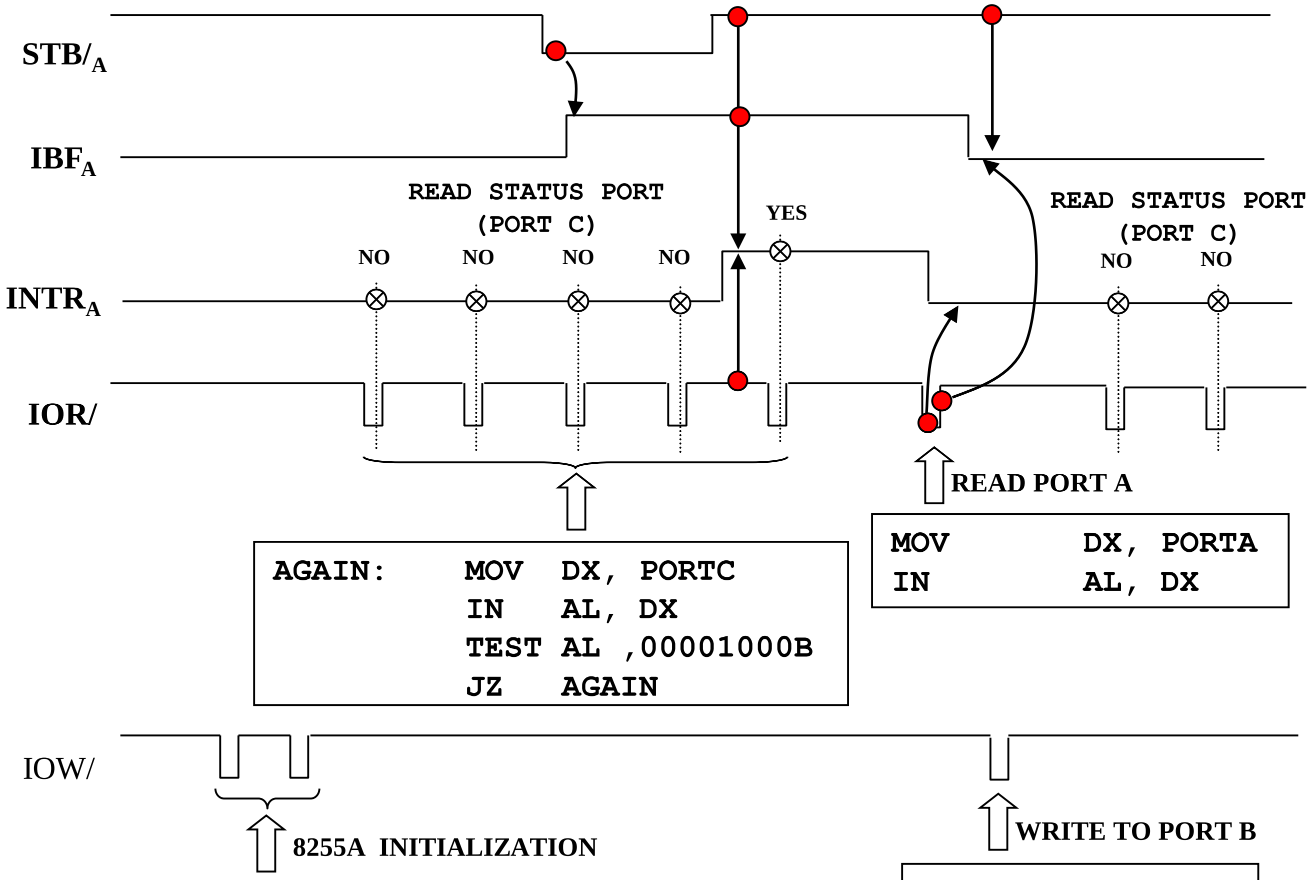
Example 1: Programmed I/O Technique (Software)

```
LOC   OBJ                  LINE     SOURCE
                              1                    NAME    PPI1
8000                          2     PORTA          EQU     8000H
8001                          3     PORTB          EQU     8001H
8002                          4     PORTC          EQU     8002H
8003                          5     PCW            EQU     8003H
                              6     ;-----------------------------------------------------------------------
----                          7     DATA           SEGMENT AT 40H
0000                          8                    ORG     0H
0000 ??                       9     VAR1           DB      ?
----                         10     DATA           ENDS
                             11     ;-----------------------------------------------------------------------
----                         12     STACK          SEGMENT AT 50H
0000  (50                    13                    DW      50 DUP(?)
      ????
      )
0064                         14     STK_TOP           LABEL WORD
----                         15     STACK          ENDS
                             16     ;-----------------------------------------------------------------------
----                         17     EPROM          SEGMENT AT 0FE00H
                             18                    ASSUME CS:EPROM, DS:DATA, SS:STACK
0000                         19                    ORG 0H
0000                         20     BEGIN          LABEL FAR
0000 B84000                  21                    MOV     AX,DATA                 ; DS Initialization
0003 8ED8                    22                    MOV     DS,AX
0005 BA0380                  23                    MOV     DX, PCW                 ; 8255A Initialization
0008 B0B9                    24                    MOV     AL, 10111001B
000A EE                      25                    OUT     DX, AL
000B B009                    26                    MOV     AL, 00001001B           ; Set INTEA (Set PC4)
000D EE                      27                    OUT     DX, AL
```

Example 1: Programmed I/O Technique (Software) (cont’d)

```
                    28   ; Checking Status
                    29                ;---------------------------------------------------
                    30                ; PC7 | PC6 | IBFA | INTEA | INTRA | PC2 | PC1 | PC0 |
                    31                ;---------------------------------------------------
000E BA0280         32   AGAIN:       MOV     DX, PORTC
0011 EC             33                IN      AL, DX
0012 A808           34                TEST    AL ,00001000B
0014 74F8           35                JZ      AGAIN
0016 BA0080         36                MOV     DX, PORTA
0019 EC             37                IN      AL, DX        ; Service I/O: Read Port A
001A 42             38                INC     DX
001B EE             39                OUT     DX, AL        ; Write to Port B
001C EBF0           40                JMP     AGAIN
----                41   EPROM        ENDS
                    42   ;-------------------------------------------------------------------
----                43   CODE         SEGMENT AT 0FFFFH
                    44                ASSUME CS:CODE, SS:STACK
                    45   ;
0000                46                ORG 0H
0000 FA             47   START:       CLI
0001 B85000         48                MOV AX,STACK
0004 8ED0           49                MOV SS,AX
0006 BC6400         50                MOV SP,OFFSET STK_TOP
0009 EA000000FE     51                JMP BEGIN
----                52   CODE         ENDS
                    53   END          START

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Example 1: Programmed I/O Technique (Timing)



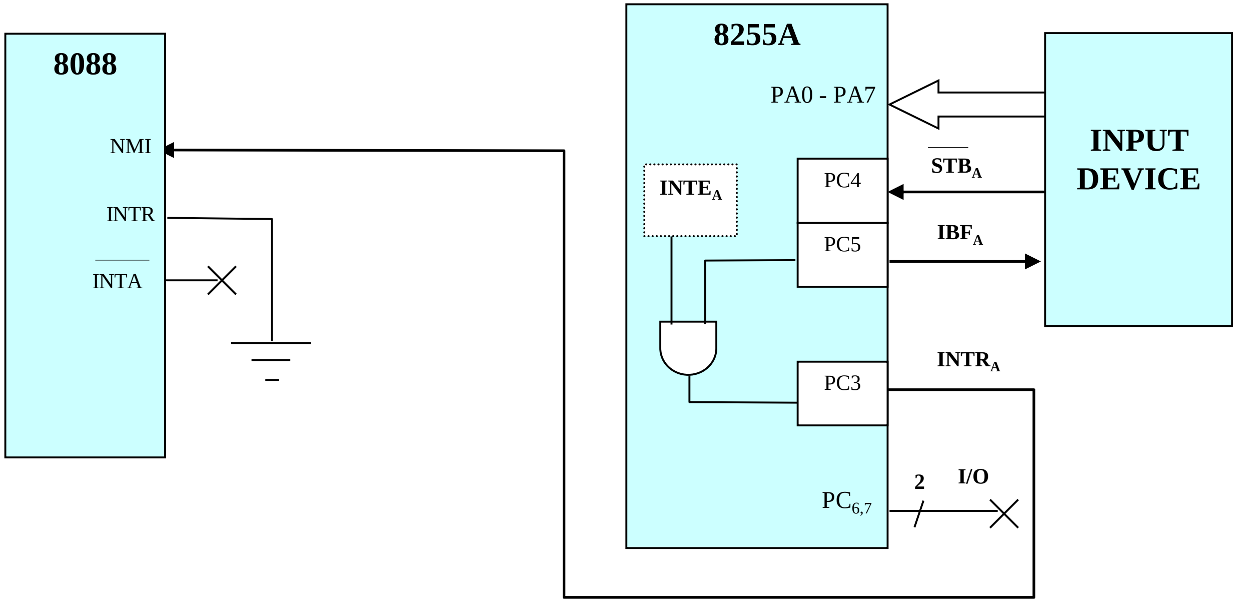
```
AGAIN:     MOV  DX, PORTC
           IN   AL, DX
           TEST AL ,00001000B
           JZ   AGAIN
```

```
MOV          DX, PORTA
IN           AL, DX
```

```
MOV          DX, PCW
MOV          AL,
10111001B
OUT          DX, AL
MOV          AL,
00001001B
OUT          DX, AL
```

```
INC          DX
OUT          DX, AL
```

Example 2: Interrupt Driven I/O (NMI) - (Hardware)



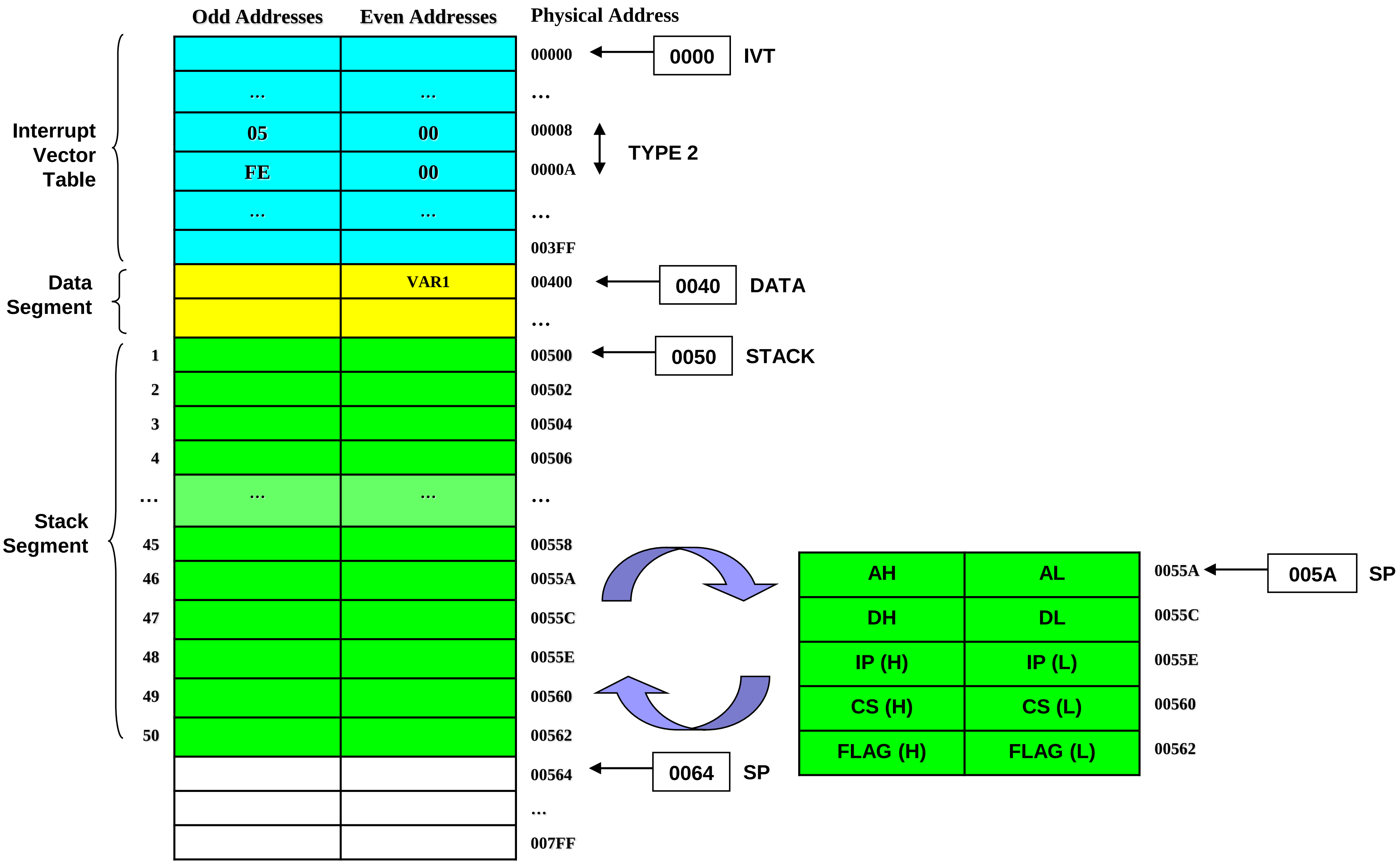
Example 2: Interrupt Driven I/O (NMI) - (Software)

```
LOC  OBJ                  LINE     SOURCE
                             1     NAME            PPI2
----                         2     DATA            SEGMENT AT 40H
0000                         3                     ORG     0H
0000 ??                      4     VAR1            DB      ?
----                         5     DATA            ENDS
                             6     ;----------------------------------------------------------
----                         7     STACK           SEGMENT AT 50H
0000 (50                     8                     DW      50 DUP(?)
     ????
     )
0064                         9     STK_TOP         LABEL   WORD
----                        10     STACK           ENDS
                            11     ;----------------------------------------------------------
----                        12     IVT             SEGMENT AT 0H
0008                        13                     ORG     2H*4
0008                        14     NMI_INT         LABEL   DWORD
0008 000500FE               15                     DD      FIRST
----                        16     IVT             ENDS
                            17     ;----------------------------------------------------------
----                        18     EPROM           SEGMENT AT 0FE00H
                            19                     ASSUME  CS:EPROM, SS:STACK
0000                        20                     ORG     0H
0000                        21     BEGIN           LABEL   FAR
                            22                     ASSUME  DS: IVT
0000 B80000                 23                     MOV     AX, IVT
0003 8ED8                   24                     MOV     DS, AX
0005 C70608000005           25                     MOV     WORD PTR NMI_INT, OFFSET NMI_SR_P
000B C7060A0000FE           26                     MOV     WORD PTR NMI_INT+2, SEG  NMI_SR_P
                            27                     ASSUME  DS:DATA
0011 B84000                 28                     MOV     AX, DATA
0014 8ED8                   29                     MOV     DS, AX
```

Example 2: Interrupt Driven I/O (NMI) - (Software) (Cont’d)

```
0016  BA0380                     MOV     DX, 8003H     ; 8255A Initialization
0019  B0B9                       MOV     AL, 0B9H
001B  EE                         OUT     DX, AL
001C  B009                       MOV     AL,09H
001E  EE                         OUT     DX,AL           ; Set PC4
001F  FB                         STI
0020  EBFE          AGAIN:       JMP     AGAIN
----                EPROM        ENDS
                   ;-------------------------------------------------------------
----                NMI_SR       SEGMENT  AT   0FE00H
                                 ASSUME  CS:NMI_SR, SS:STACK
0500                             ORG     0500H
0500                NMI_SR_P     PROC    FAR
0500                First        LABEL   FAR
0500 52                          PUSH    DX
0501 50                          PUSH    AX
0502 BA0080                      MOV     DX, 8000H
0505 EC                          IN      AL, DX      ; Service I/O: Read Port A
0506 42                          INC     DX
0507 EE                          OUT     DX, AL      ; Write to Port B
0508 58                          POP     AX
0509 5A                          POP     DX
050A CF                          IRET
                    NMI_SR_P     ENDP
----                NMI_SR       ENDS
                    ;-------------------------------------------------------------
----                CODE         SEGMENT  AT   0FFFFH
                                 ASSUME  CS:CODE, SS:STACK
0000                             ORG     0H
0000 FA             START:       CLI
0001 B85000                      MOV     AX,STACK
0004 8ED0                        MOV     SS,AX
0006 BC6400                      MOV     SP,OFFSET STK_TOP
0009 EA000000FE                  JMP     BEGIN
----                CODE         ENDS
                    END          START
ASSEMBLY COMPLETE, NO ERRORS FOUND
```

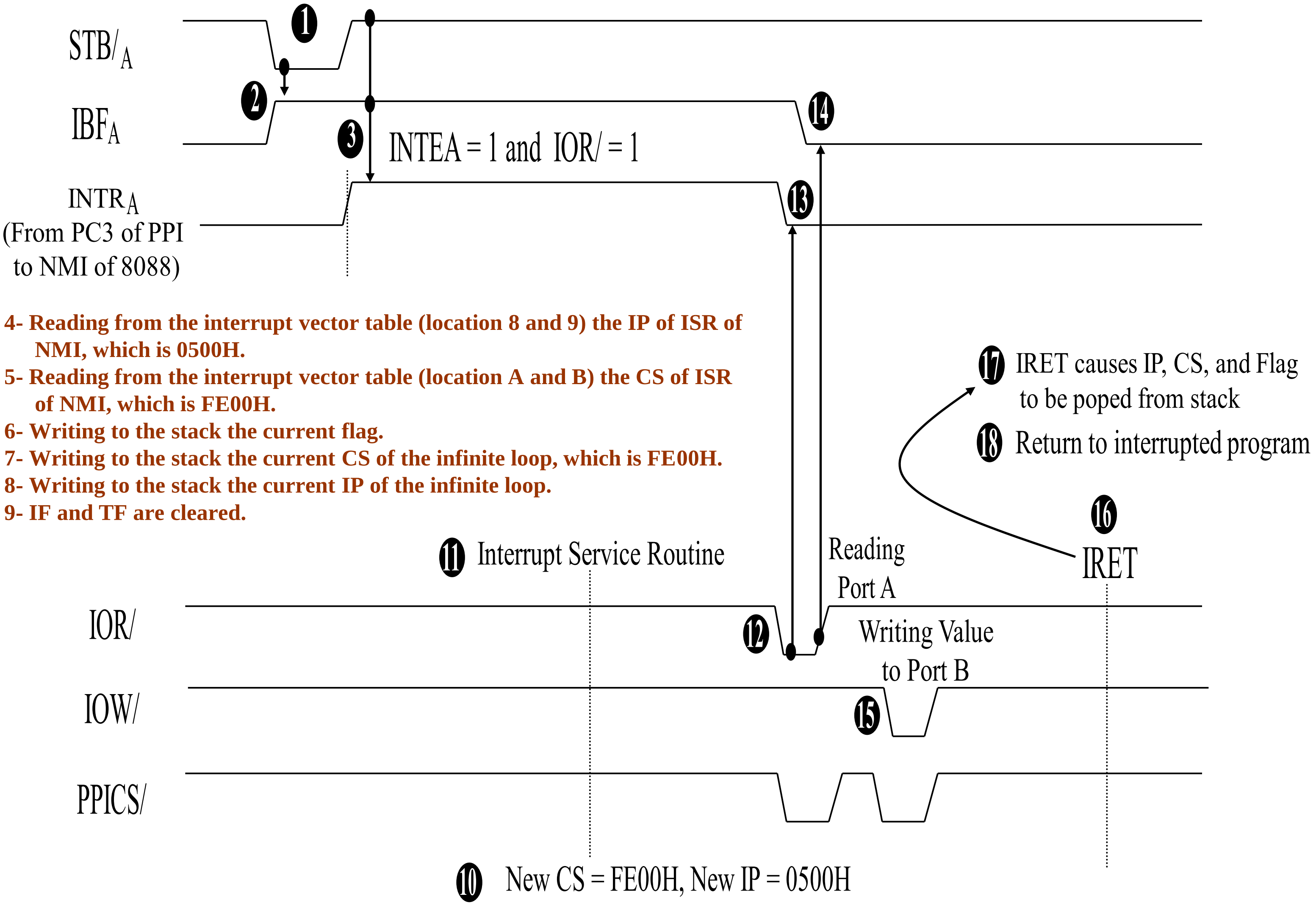
Example 2: RAM Mapping



	Odd Addresses	Even Addresses	Physical Address
			00000
	…	…	…
	05	00	00008
	FE	00	0000A
	…	…	…
			003FF
		VAR1	00400
			…
1			00500
2			00502
3			00504
4			00506
…	…	…	…
45			00558
46			0055A
47			0055C
48			0055E
49			00560
50			00562
			00564
			…
			007FF

AH	AL	0055A
DH	DL	0055C
IP (H)	IP (L)	0055E
CS (H)	CS (L)	00560
FLAG (H)	FLAG (L)	00562

Example 2: Interrupt Driven I/O (NMI) - (Timing)



4- Reading from the interrupt vector table (location 8 and 9) the IP of ISR of NMI, which is 0500H.

5- Reading from the interrupt vector table (location A and B) the CS of ISR of NMI, which is FE00H.

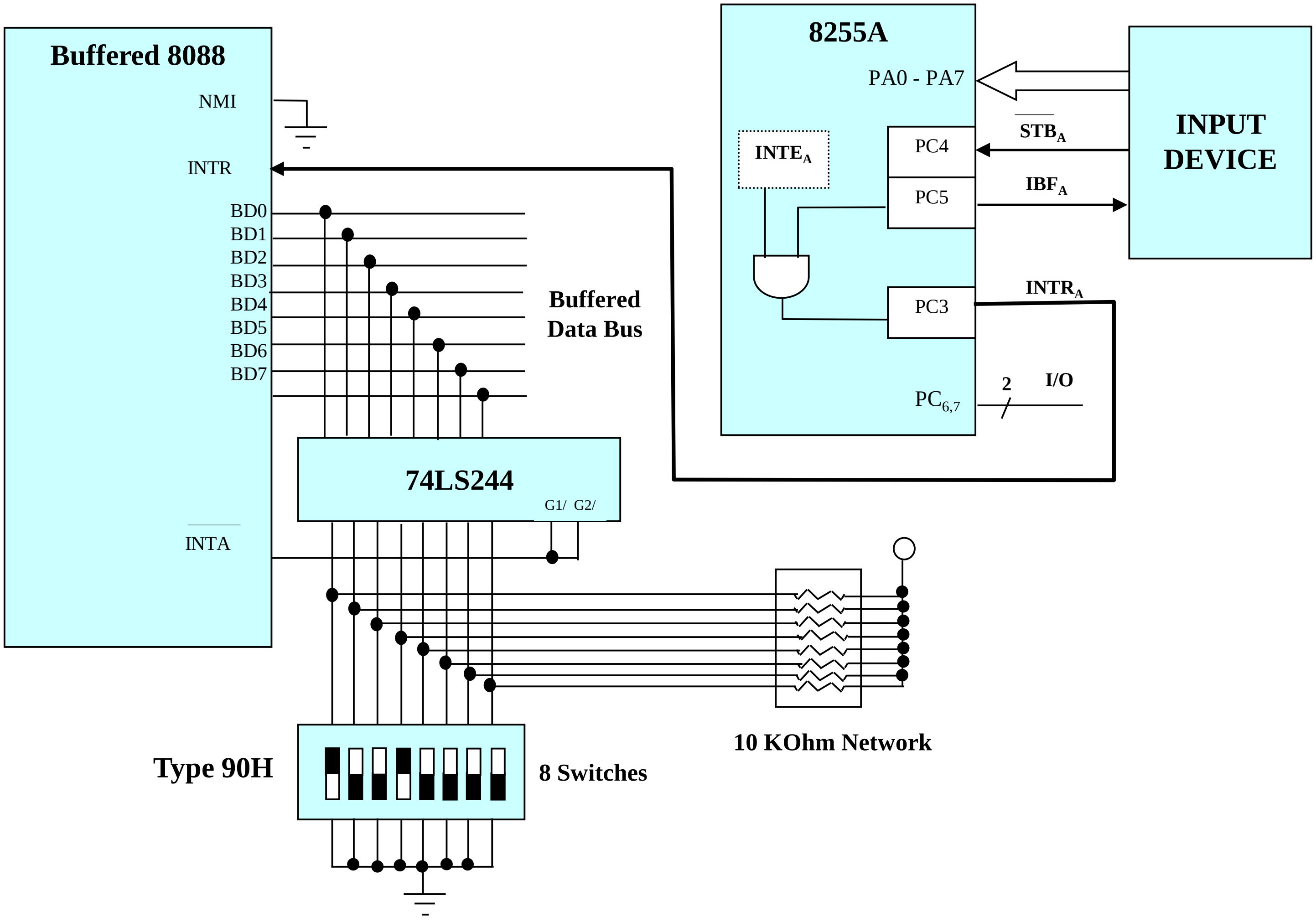
6- Writing to the stack the current flag.

7- Writing to the stack the current CS of the infinite loop, which is FE00H.

8- Writing to the stack the current IP of the infinite loop.

9- IF and TF are cleared.

Example 3: Interrupt Driven I/O (INTR) - (Hardware)



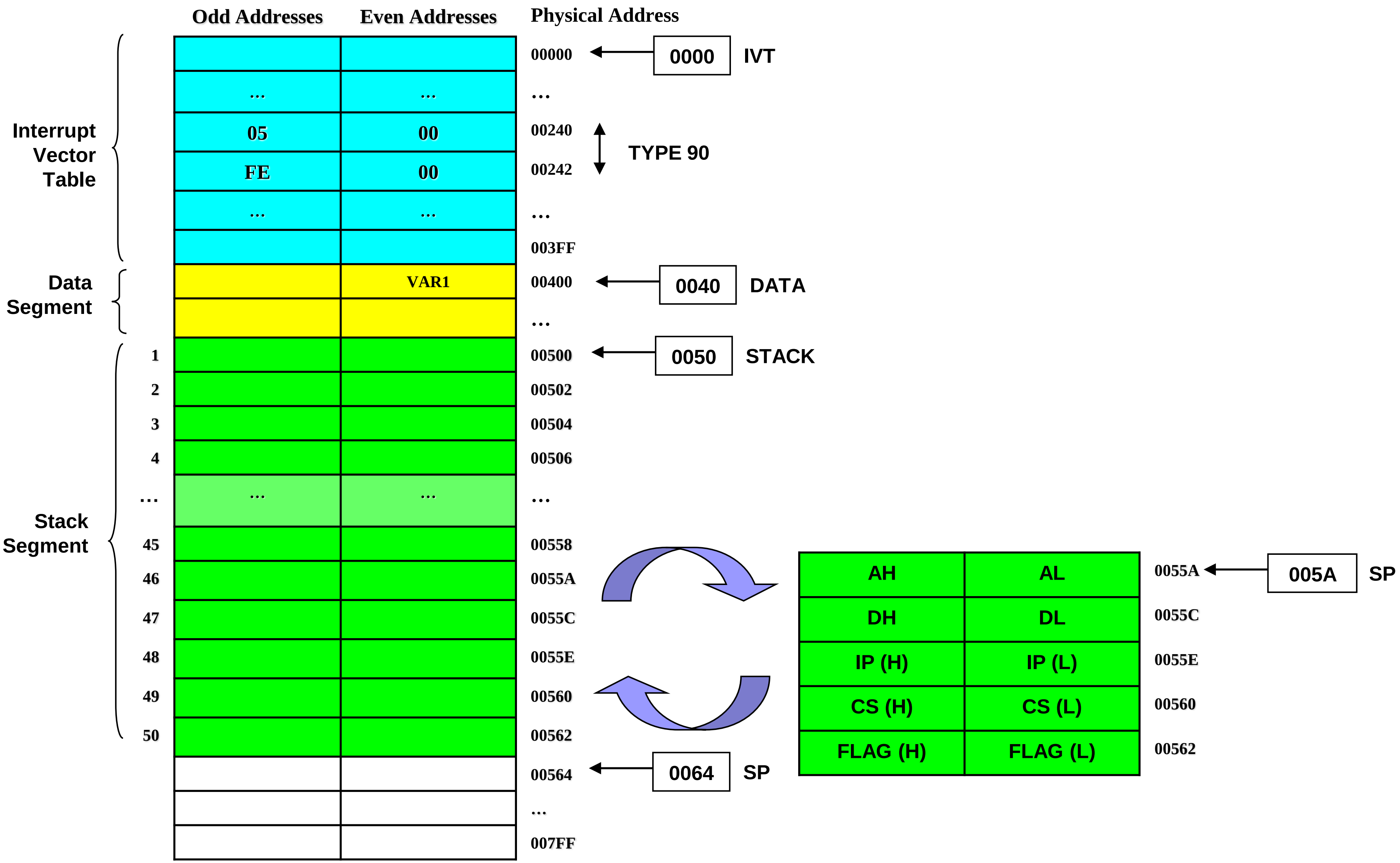
Example 3: Interrupt Driven I/O (NMI) - (Software)

```
LOC  OBJ                LINE     SOURCE
                           1     NAME           PPI3
----                       2     DATA           SEGMENT AT 40H
0000                       3                    ORG     0H
0000 ??                    4     VAR1           DB      ?
----                       5     DATA           ENDS
                           6     ;-----------------------------------------------
----                       7     STACK          SEGMENT AT 50H
0000 (50                   8                    DW      50 DUP(?)
     ????
     )
0064                       9     STK_TOP        LABEL   WORD
----                      10     STACK          ENDS
                          11     ;-----------------------------------------------
----                      12     IVT            SEGMENT AT 0H
0240                      13                    ORG     90H*4
0240                      14     INT90          LABEL   DWORD
0240 000500FE             15                    DD      FIRST
----                      16     IVT            ENDS
                          17     ;-----------------------------------------------
----                      18     EPROM          SEGMENT AT 0FE00H
                          19                    ASSUME  CS:EPROM, SS:STACK
0000                      20                    ORG     0H
0000                      21     BEGIN          LABEL   FAR
                          22                    ASSUME  DS: IVT
0000 B80000               23                    MOV     AX, IVT
0003 8ED8                 24                    MOV     DS, AX
0005 C70640020005         25                    MOV     WORD PTR INT90, OFFSET INT90_SR_P
000B C706420200FE         26                    MOV     WORD PTR INT90+2, SEG  INY90_SR_P
                          27                    ASSUME  DS:DATA
0011 B84000               28                    MOV     AX, DATA
0014 8ED8                 29                    MOV     DS, AX
```

Example 3: Interrupt Driven I/O (INTR) - (Software) (Cont'd)

```
0016  BA0380          30                      MOV     DX, 8003H     ; 8255A Initialization
0019  B0B9            31                      MOV     AL, 0B9H
001B  EE              32                      OUT     DX, AL
001C  B009            33                      MOV     AL,09H
001E  EE              34                      OUT     DX,AL          ; Set PC4
001F  FB              35                      STI
0020  EBFE            36      AGAIN:          JMP     AGAIN
----                  37      EPROM           ENDS
                      38     ;-------------------------------------------------------------
----                  39      INT90_SR        SEGMENT  AT   0FE00H
                      40                      ASSUME  CS:INT90_SR, SS:STACK
0500                  41                      ORG     0500H
0500                  42      INT90_SR_P            PROC    FAR
0500                  43      First           LABEL   FAR
0500 52               44                      PUSH    DX
0501 50               45                      PUSH    AX
0502 BA0080           46                      MOV     DX, 8000H
0505 EC               47                      IN      AL, DX      ; Service I/O: Read Port A
0506 42               48                      INC     DX
0507 EE               49                      OUT     DX, AL      ; Write to Port B
0508 58               50                      POP     AX
0509 5A               51                      POP     DX
050A CF               52                      IRET
                      53      INT90_SR_P      ENDP
----                  54      INT90_SR        ENDS
                      55      ;-------------------------------------------------------------
----                  56      CODE            SEGMENT  AT   0FFFFH
                      57                      ASSUME  CS:CODE, SS:STACK
0000                  58                      ORG     0H
0000 FA               59      START:          CLI
0001 B85000           60                      MOV     AX,STACK
0004 8ED0             61                      MOV     SS,AX
0006 BC6400           62                      MOV     SP,OFFSET STK_TOP
0009 EA000000FE       63                      JMP     BEGIN
----                  64      CODE            ENDS
                      65      END             START
ASSEMBLY COMPLETE, NO ERRORS FOUND
```

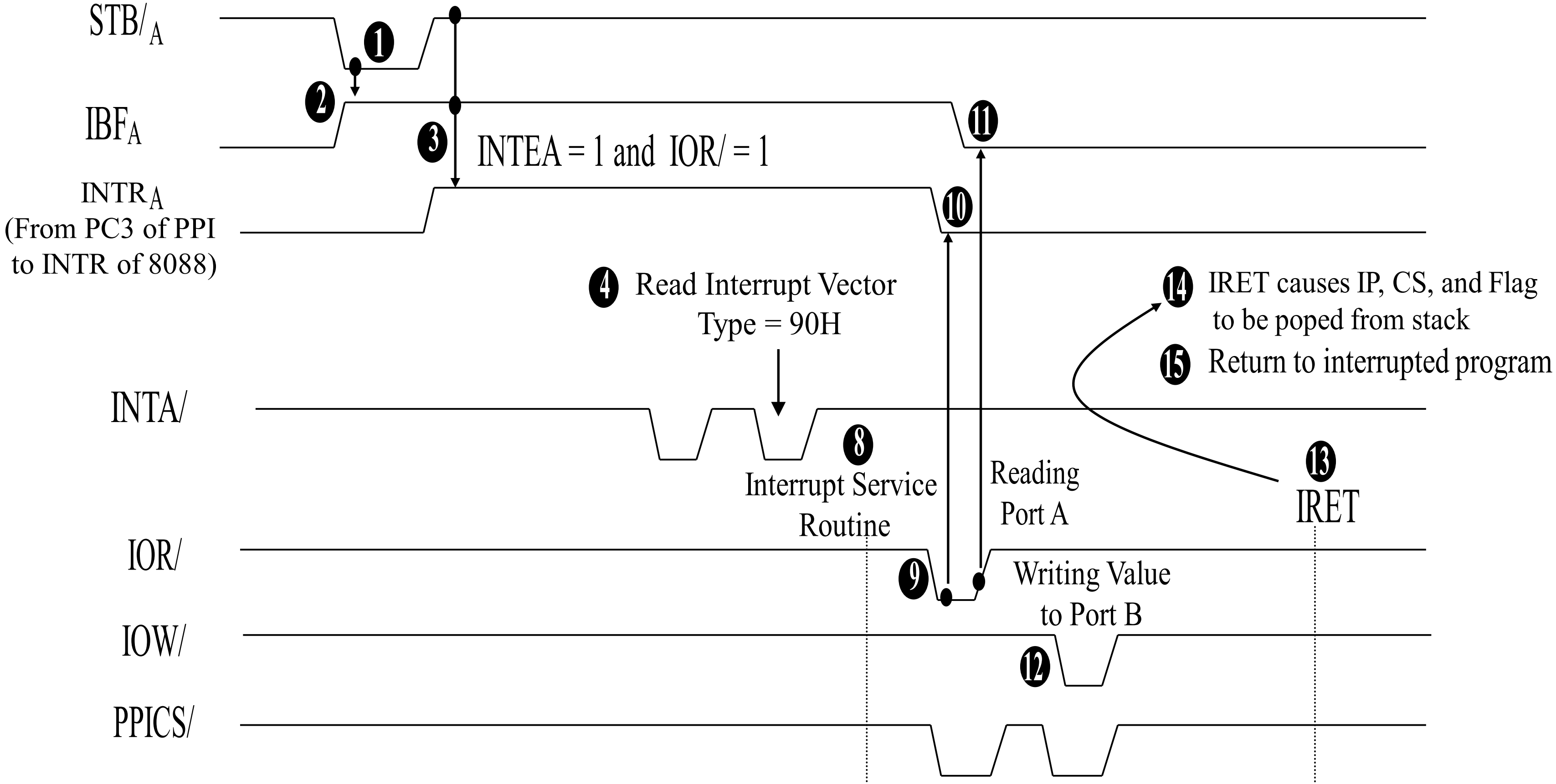
Example 3: RAM Mapping



		Odd Addresses	Even Addresses	Physical Address	
Interrupt Vector Table				00000	0000 IVT
		…	…	…	
		05	00	00240	TYPE 90
		FE	00	00242	
		…	…	…	
				003FF	
Data Segment			VAR1	00400	0040 DATA
				…	
Stack Segment	1			00500	0050 STACK
	2			00502	
	3			00504	
	4			00506	
	…	…	…	…	
	45			00558	
	46			0055A	
	47			0055C	
	48			0055E	
	49			00560	
	50			00562	
				00564	0064 SP
				…	
				007FF	

AH	AL	0055A	005A SP
DH	DL	0055C	
IP (H)	IP (L)	0055E	
CS (H)	CS (L)	00560	
FLAG (H)	FLAG (L)	00562	

Example 3: Interrupt Driven I/O (INTR) - (Timing)



❸ INTEA = 1 and IOR/ = 1

❹ Read Interrupt Vector Type = 90H

❽ Interrupt Service Routine

❾ Reading Port A

⓫

⓬ Writing Value to Port B

⓭ IRET

⓮ IRET causes IP, CS, and Flag to be poped from stack

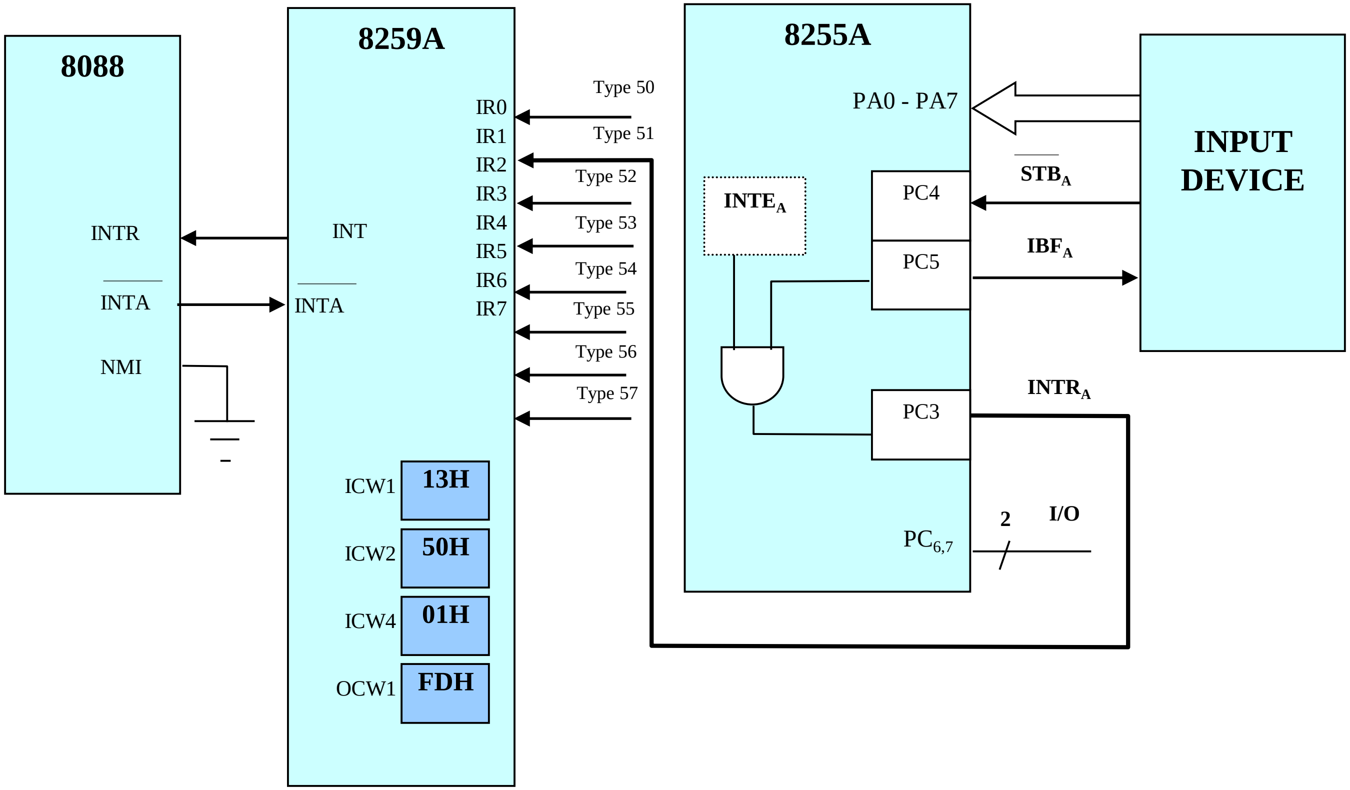
⓯ Return to interrupted program

❺ Flag, CS, and IP are pushed onto stack

❻ IF and TF are cleared

❼ New IP_LSB = Content of location 4*90H = 00
New IP_MSB = Content of location 4*90H + 1 = 05
New CS_LSB = Content of location 4*90H + 2 = 00
New CS_MSB = Content of location 4*90H + 3 = FE

Example 4: Interrupt Driven I/O (INTR) using PIC - (Hardware)



Example 4: Interrupt Driven I/O (INTR) using PIC - (Software)

```
LOC  OBJ                    LINE     SOURCE
                               1     NAME            PPI4
----                           2     DATA            SEGMENT AT 40H
0000                           3                     ORG     0H
0000 ??                        4     VAR1            DB      ?
----                           5     DATA            ENDS
                               6     ;----------------------------------------------------------
----                           7     STACK           SEGMENT AT 50H
0000 (50                       8                     DW      50 DUP(?)
     ????
     )
0064                           9     STK_TOP         LABEL   WORD
----                          10     STACK           ENDS
                              11     ;----------------------------------------------------------
----                          12     IVT             SEGMENT AT 0H
0144                          13                     ORG     51H*4
0144                          14     INT51           LABEL   DWORD
0144 000500FE                 15                     DD      FIRST
----                          16     IVT             ENDS
                              17     ;----------------------------------------------------------
----                          18     EPROM           SEGMENT AT 0FE00H
                              19                     ASSUME  CS:EPROM, SS:STACK
0000                          20                     ORG     0H
0000                          21     BEGIN           LABEL   FAR
                              22     ; IVT Initialization
                              23                     ASSUME  DS: IVT
0000 B80000                   24                     MOV     AX, IVT
0003 8ED8                     25                     MOV     DS, AX
0005 C70644010005             26                     MOV     WORD PTR INT51, OFFSET INT51_SR_P
000B C706460100FE             27                     MOV     WORD PTR INT51+2, SEG  INT51_SR_P
                              28    ; DS Initialization
                              29                     ASSUME  DS:DATA
0011 B84000                   30                     MOV     AX, DATA
0014 8ED8                     31                     MOV     DS, AX
```

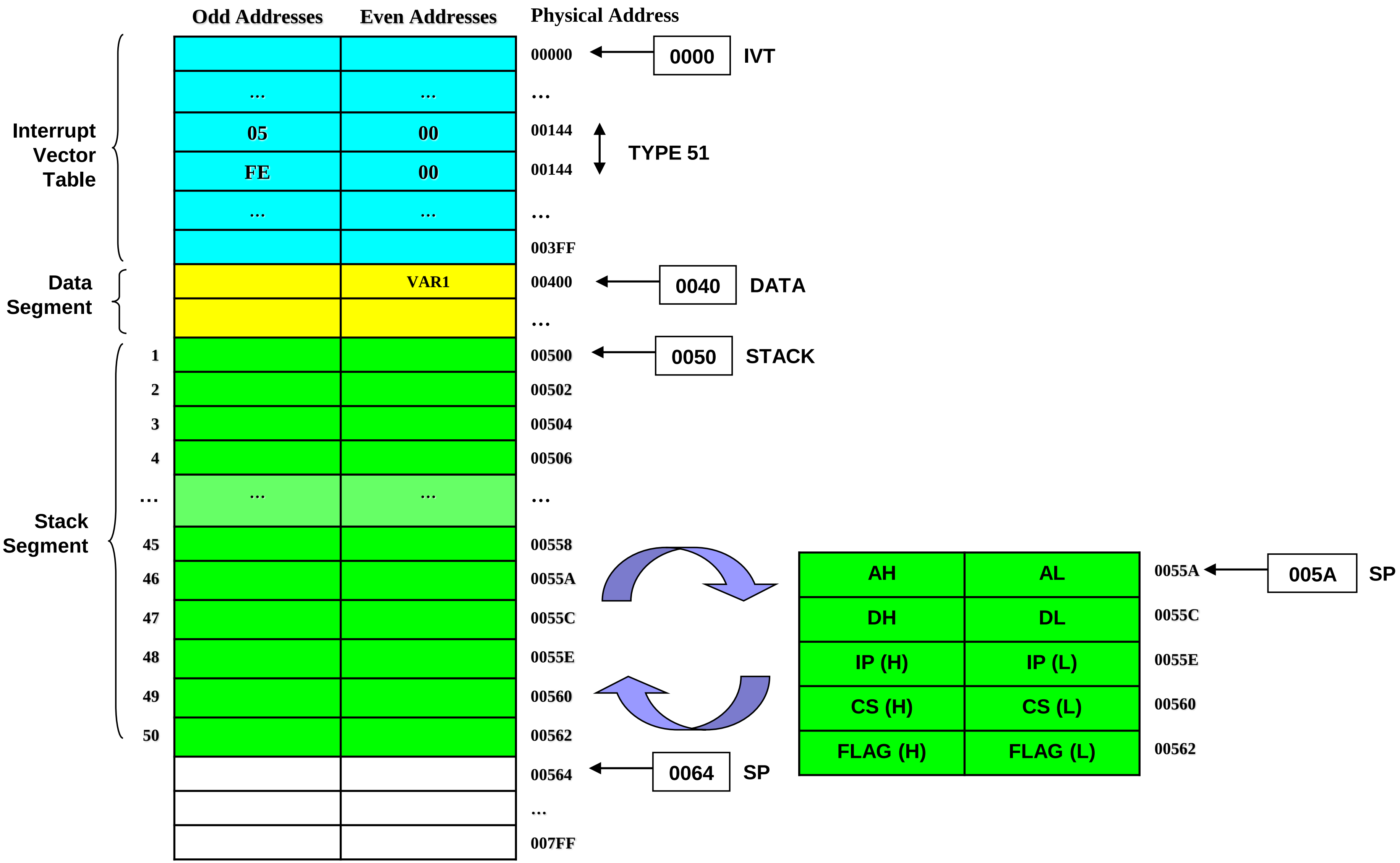
Example 4: Interrupt Driven I/O (INTR) using PIC - (Software) (Cont’d)

```
                        32      ; 8259A INITIALIZATION
0016 BA4080             33                      MOV     DX, 8040H
0019 B013               36                      MOV     AL, 13H
001B EE                 37                      OUT     DX, AL          ;ICW1
001C 42                 38                      INC     DX
001D B050               39                      MOV     AL, 50H
001F EE                 40                      OUT     DX, AL          ;ICW2
0020 B001               41                      MOV     AL, 1H
0022 EE                 42                      OUT     DX, AL          ;ICW4
0023 B0FD               43                      MOV     AL, 0FDH
0025 EE                 44                      OUT     DX, AL          ;OCW1
                        45      ; 8255A INITIALIZATION
0026 BA0380             46                      MOV     DX, 8003H
0029 B0B9               47                      MOV     AL, 0B9H
002B EE                 48                      OUT     DX, AL
002C B009               49                      MOV     AL,09H
002E EE                 50                      OUT     DX,AL
                        51      ;
002F FB                 52                      STI
0030 EBFE               53      AGAIN:          JMP     AGAIN
----                    54      EPROM           ENDS
```

Example 4: Interrupt Driven I/O (INTR) using PIC - (Software) (Cont’d)

```
                       55     ;Interrupt Service Routine
----                   56     INT51_SR        SEGMENT  AT   0FE00H
                       57                     ASSUME   CS:INT51_SR, SS:STACK
0500                   58                     ORG      0500H
0500                   59     INT51_SR_P      PROC     FAR
0500                   60     First           LABEL    FAR
0500 52                61                     PUSH     DX
0501 50                62                     PUSH     AX
0502 BA0080            63                     MOV      DX, 8000H
0505 EC                64                     IN       AL, DX      ; Service I/O: Read Port A
0506 42                65                     INC      DX
0507 EE                66                     OUT      DX, AL      ; Write to Port B
                       67     ;
0508 BA4080            68                     MOV      DX, 8040H
050B B020              69                     MOV      AL, 20H
050D EE                70                     OUT      DX, AL      ;NON-SPECIFIC EOI
                       71     ;
050E 58                72                     POP      AX
050F 5A                73                     POP      DX
0510 CF                74                     IRET
                       75     INT51_SR_P      ENDP
----                   76     INT51_SR        ENDS
                       77     ;-------------------------------------------------------------
----                   78     CODE            SEGMENT  AT   0FFFFH
                       79                     ASSUME   CS:CODE, SS:STACK
0000                   80                     ORG      0H
0000 FA                81     START:          CLI
0001 B85000            82                     MOV      AX,STACK
0004 8ED0              83                     MOV      SS,AX
0006 BC6400            84                     MOV      SP,OFFSET STK_TOP
0009 EA000000FE        85                     JMP      BEGIN
----                   86     CODE            ENDS
                       87     END             START

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

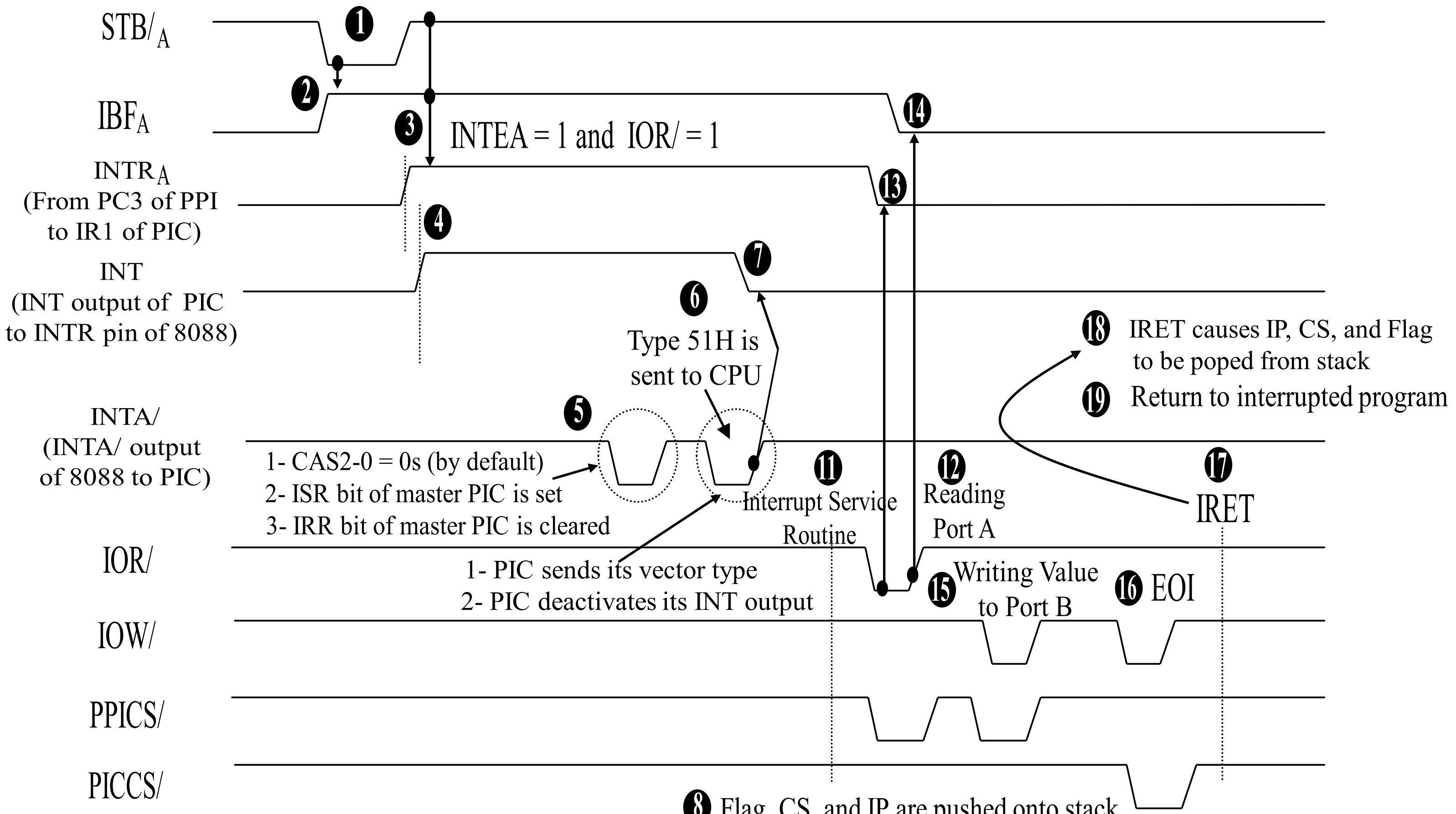
Example 4: RAM Mapping



		Odd Addresses	Even Addresses	Physical Address	
Interrupt Vector Table				00000	0000 IVT
		…	…	…	
		05	00	00144	TYPE 51
		FE	00	00144	
		…	…	…	
				003FF	
Data Segment			VAR1	00400	0040 DATA
				…	
Stack Segment	1			00500	0050 STACK
	2			00502	
	3			00504	
	4			00506	
	…	…	…	…	
	45			00558	
	46			0055A	
	47			0055C	
	48			0055E	
	49			00560	
	50			00562	
				00564	0064 SP
				…	
				007FF	

AH	AL	0055A	005A SP
DH	DL	0055C	
IP (H)	IP (L)	0055E	
CS (H)	CS (L)	00560	
FLAG (H)	FLAG (L)	00562	

Example 4: Interrupt Driven I/O (INTR) using PIC - (Timing)


STB/$_A$
IBF$_A$
INTR$_A$ (From PC3 of PPI to IR1 of PIC)
INT (INT output of PIC to INTR pin of 8088)
INTA/ (INTA/ output of 8088 to PIC)
IOR/
IOW/
PPICS/
PICCS/
1
2
3
4
5
6
7
11
12
13
14
15
16
17
18
19

INTEA = 1 and IOR/ = 1

1- CAS2-0 = 0s (by default)
2- ISR bit of master PIC is set
3- IRR bit of master PIC is cleared

Type 51H is sent to CPU

1- PIC sends its vector type
2- PIC deactivates its INT output

Interrupt Service Routine

Reading Port A

Writing Value to Port B

EOI

IRET

IRET causes IP, CS, and Flag to be poped from stack

Return to interrupted program

8 Flag, CS, and IP are pushed onto stack

9 IF and TF are cleared

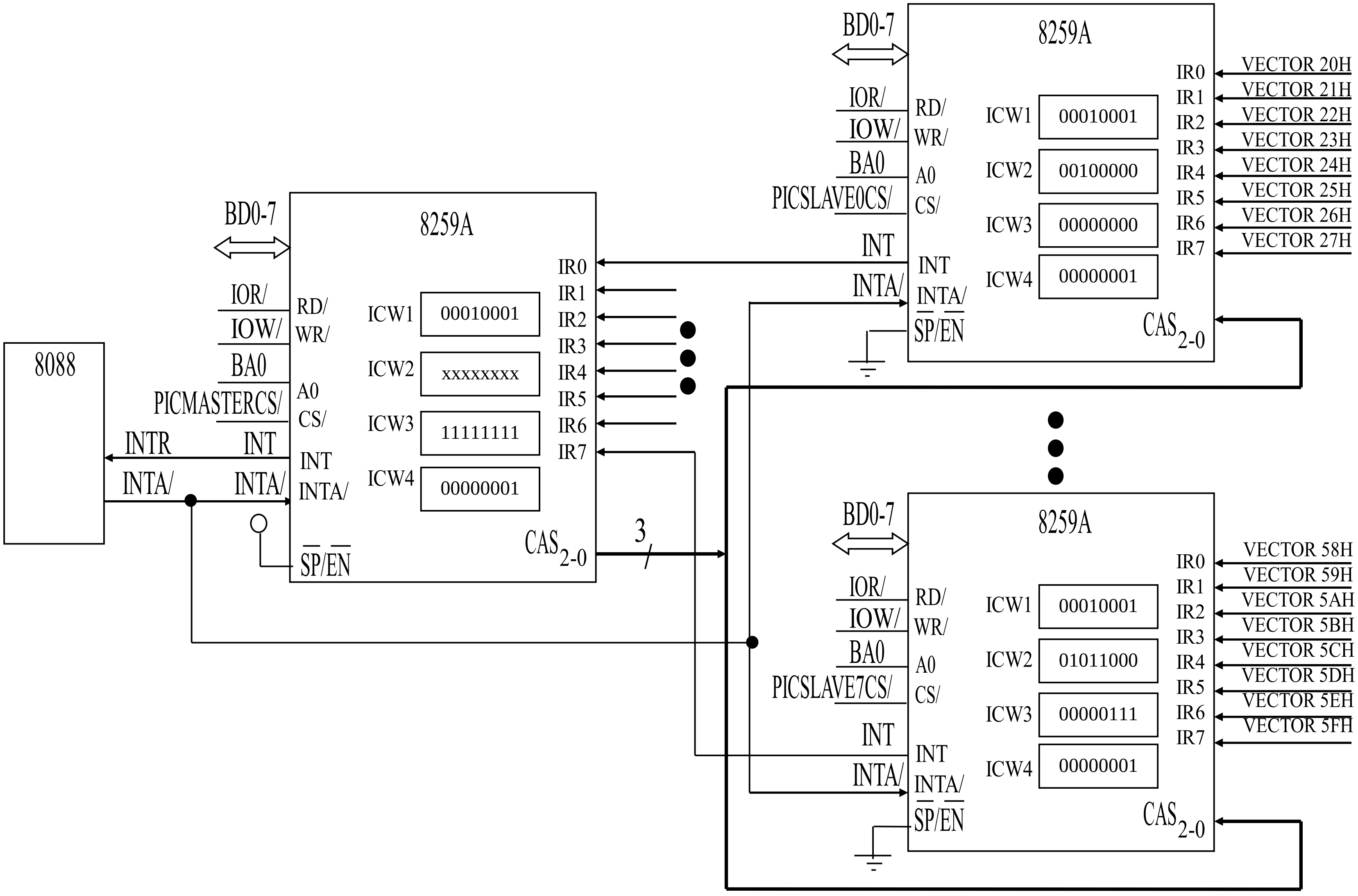
10
New IP_LSB = Content of location 4*51H = 00
New IP_MSB = Content of location 4*51H + 1 = 05
New CS_LSB = Content of location 4*51H + 2 = 00
New CS_MSB = Content of location 4*51H + 3 = FE

Optional Topic

Cascading Multiple 8259As

- The **INT**s of the slaves are connected to the Interrupt Requests (IRs) of the master.
- In the **Non Buffered Mode**, the master is designated by SP/=Vcc and a slave is designated by SP/=GND. In the **Buffered Mode**, this is done by software (**ICW4**).
- Master **PIC INT** is connected to the CPU's **INTR**.
- CPU's **INTA/** is connected to all **PIC**s' **INTA/**.
- **CAS2-0** are outputs for the master **PIC** and inputs for the slave **PIC**s.
- **CAS2-0** acts as private ID bus between the master and the slaves.
- Through software initialization, the master **PIC** is informed which of its **IR** inputs are connected to the slave **PIC**s.
- Through software initialization, each slave **PIC** is informed of its ID.

Cascading Multiple 8259As (Non-Buffered Mode)



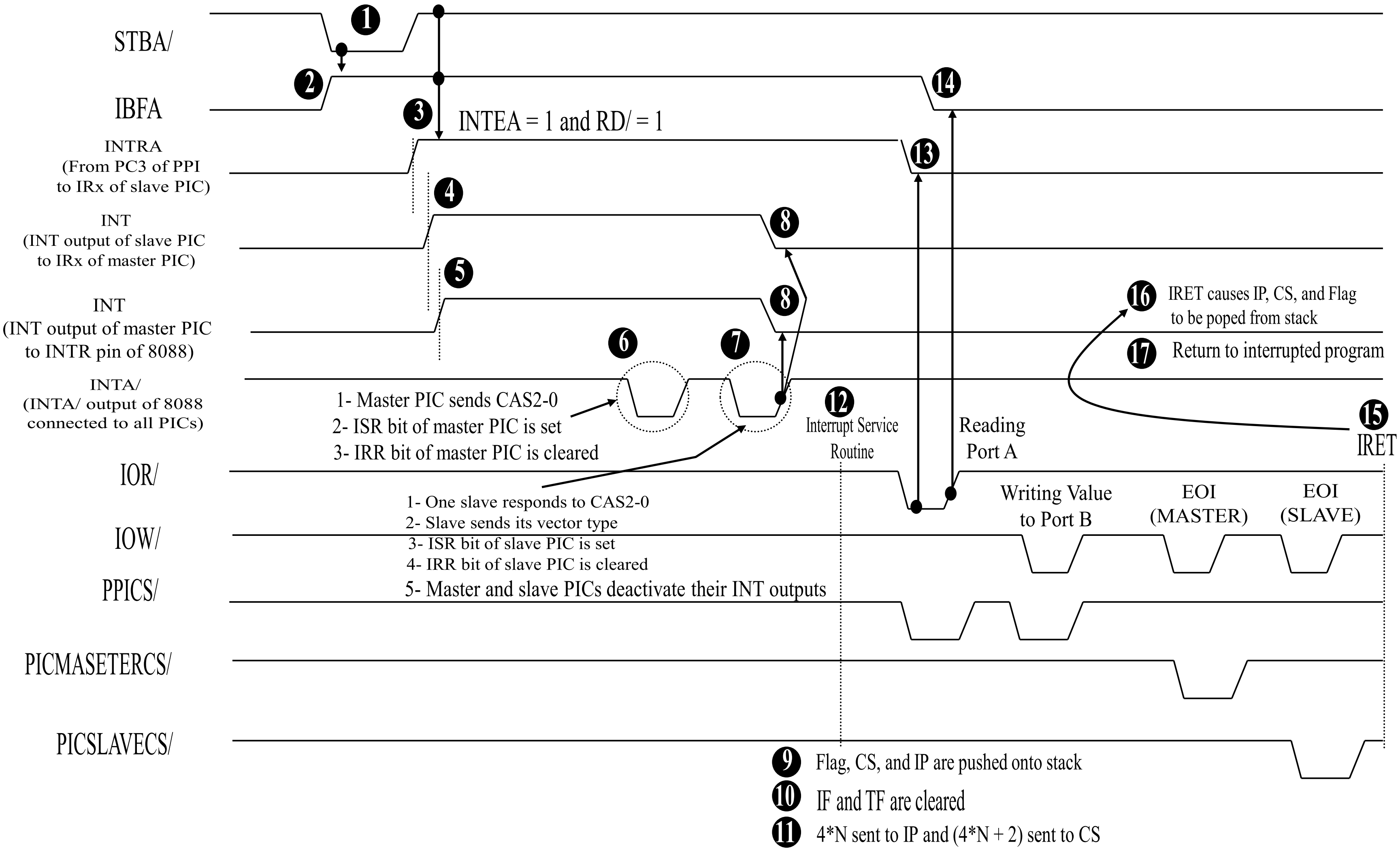
8259A: Operation Sequence in cascade Mode

- Slave **PIC** receives an interrupt request on one of its **IR** pins.
- Slave **PIC** raises **INT** output. Therefore, the master **PIC** receives an interrupt request on one of its **IR** pins.
- The master **PIC** raises the **INT** signal.
- The CPU gets **INTR**, and sends the first **INTA/** pulse.
- The master **PIC** sends cascade address (**CAS2-0**) and sets its **ISR** bit.
- The slave **PIC** with an ID matching to the address on **CAS2-0** lines responds during the second **INTA/** and sends its vector type. In addition, it sets its **ISR** bit.
- The master and the slave **PIC**s de-activate their **INT** outputs.

Note:

- **ISR** bit of both master and slave get set. This means two **EOI** commands must be issued (if not in the **AEOI** mode), one for the master and one for the slave.

Timing Waveforms of Cascading Multiple 8259As



1- Master PIC sends CAS2-0
2- ISR bit of master PIC is set
3- IRR bit of master PIC is cleared

1- One slave responds to CAS2-0
2- Slave sends its vector type
3- ISR bit of slave PIC is set
4- IRR bit of slave PIC is cleared
5- Master and slave PICs deactivate their INT outputs

16 IRET causes IP, CS, and Flag to be poped from stack

17 Return to interrupted program

9 Flag, CS, and IP are pushed onto stack

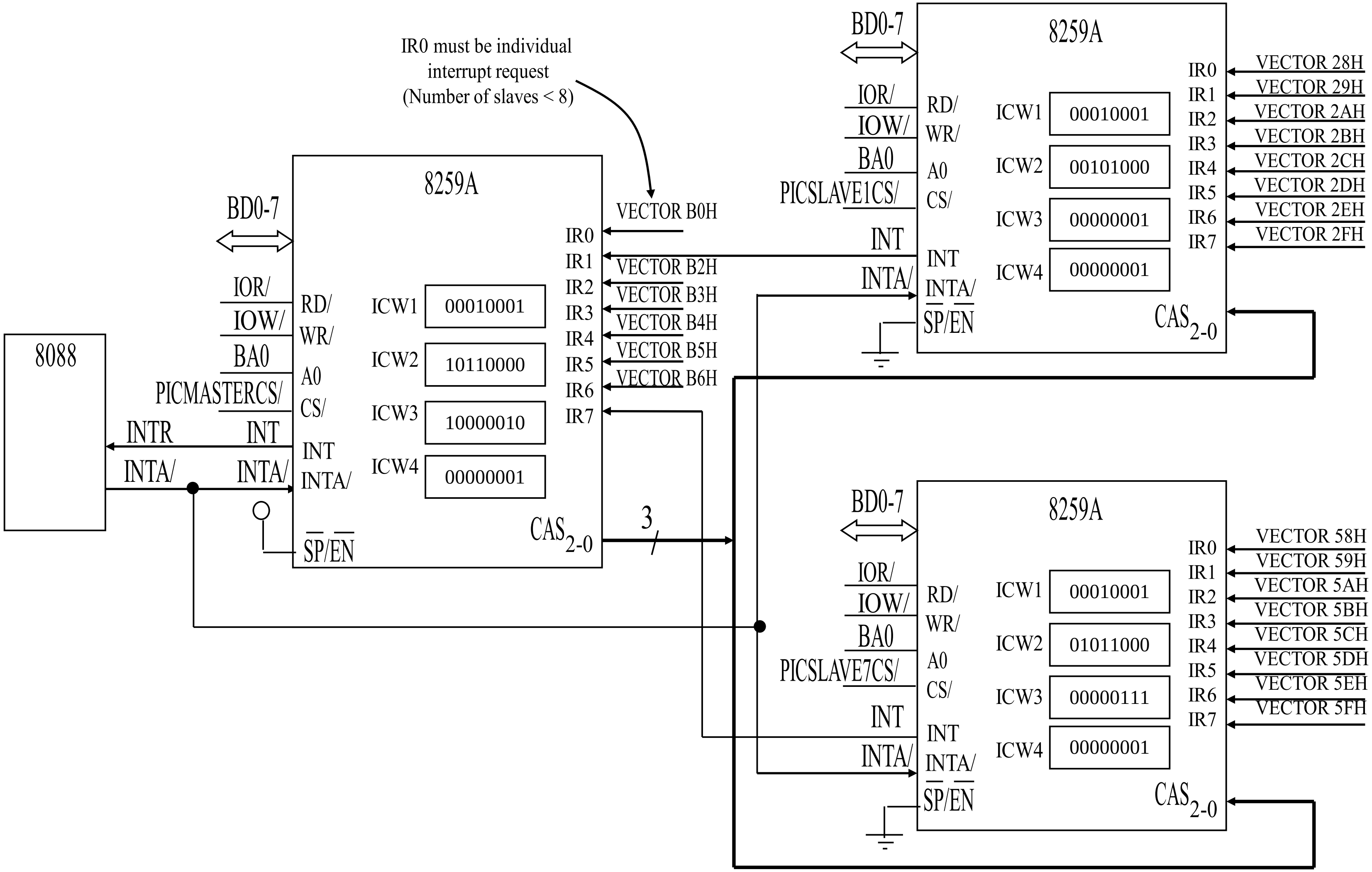
10 IF and TF are cleared

11 4*N sent to IP and (4*N + 2) sent to CS

Cascading Multiple 8259As

- Special consideration should be taken when mixed interrupt requests are assigned to a master 8259A; that is, when some of the master's **IR** inputs are used for **slave interrupt requests** and some are used for **individual interrupt requests**.
- In this type of structure, **the master's IR0 must NOT be used for a slave**. This is because when an **IR** input that is not initialized as a slave (i.e., individual interrupt request) receives an interrupt request, the **CAS2-0** lines will not be activated, thus staying in their default values (**LOW**), i.e., **the addressing for IR0** (**Slave IR0**)!
- Therefore, if a slave is connected to the master's **IR0** when a non-slave interrupt occurs on another master **IR** input, erroneous conditions may result.
- A slave **PIC** is connected to the master's **IR0** only if the rest of the seven **IR**s (**IR1-IR7**) are all connected to the slave **PIC**s.

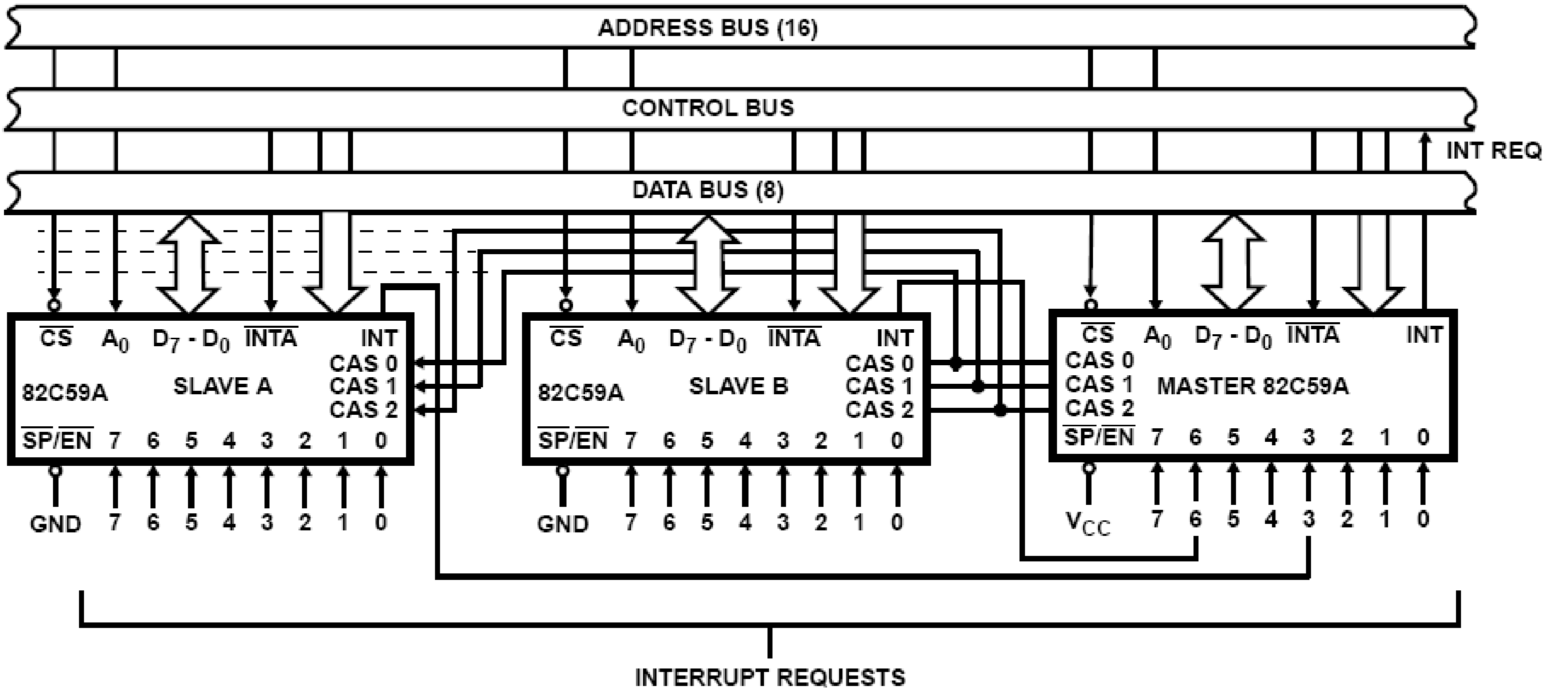
Cascading Multiple 8259As (Non-Buffered Mode)


IR0 must be individual interrupt request (Number of slaves < 8)

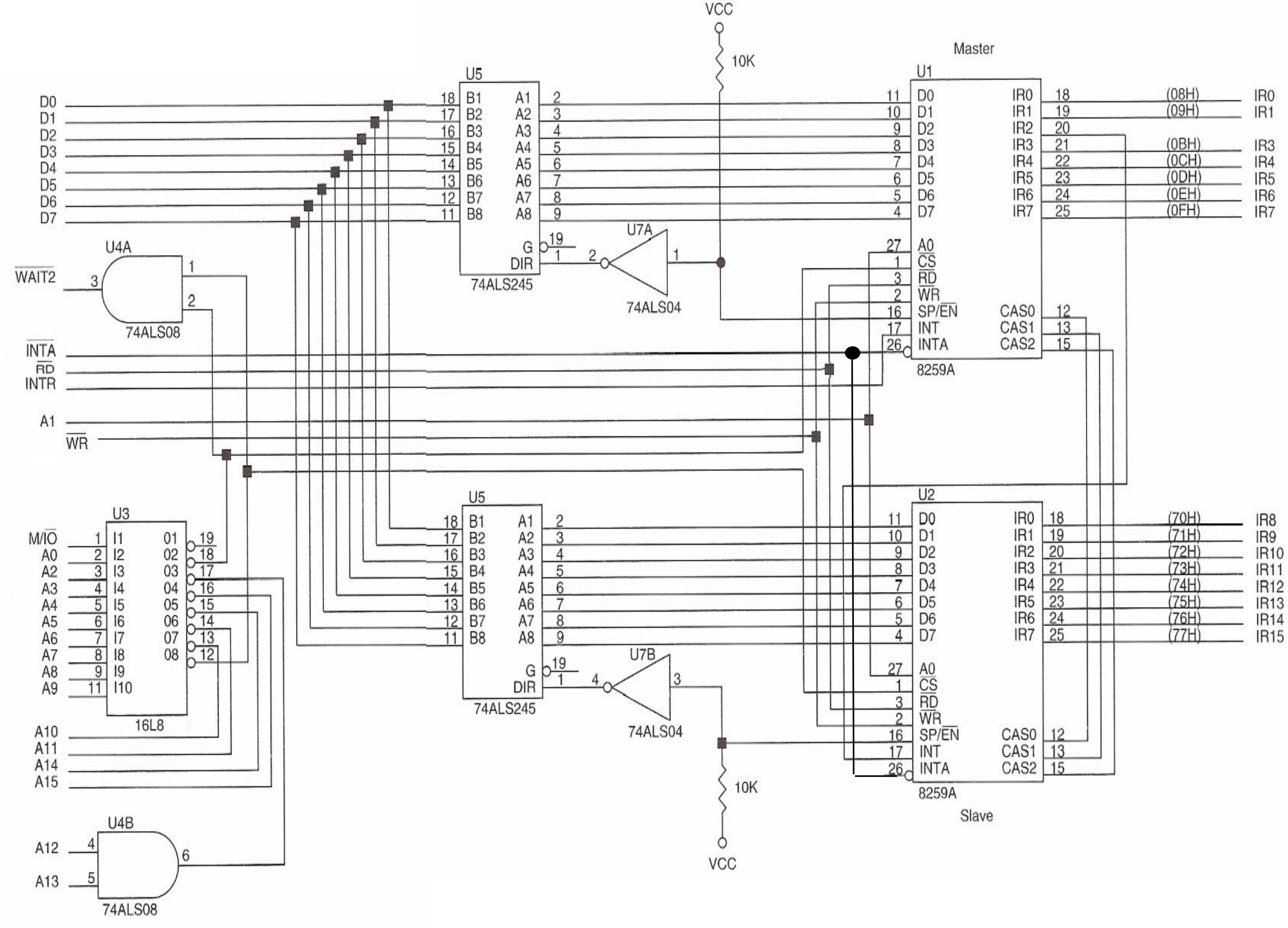
Programming the Cascading Multiple 8259As (Non-Buffered Mode)

```
        MOV     DX, MASTER_PIC_PORT0
        MOV     AL, 00010001H
        OUT     DX, AL
        INC     DX
        MOV     AL, 0B0H                ; Master’s interrupt vector =0B0H
        OUT     DX, AL
        MOV     AL, 10000001B           ; IR1 and IR7 of master have slaves
        OUT     DX, AL
        MOV     AL, 00000001B           ; Normal EOI, Non-Buffered Mode
        OUT     DX, AL
; ---------------------------------------------------------------------------------------------------------------------------------
        MOV     DX, SLAVE1_PIC_PORT0
        MOV     AL, 00010001H
        OUT     DX, AL
        INC     DX
        MOV     AL, 28H                 ; Slaves1’s interrupt vector =28H
        OUT     DX, AL
        MOV     AL, 00000001B           ; Slave’s ID = 1
        OUT     DX, AL
        MOV     AL, 00000001B           ; Normal EOI, Non-Buffered Mode
        OUT     DX, AL
; ---------------------------------------------------------------------------------------------------------------------------------
        MOV     DX, SLAVE7_PIC_PORT0
        MOV     AL, 00010001H
        OUT     DX, AL
        INC     DX
        MOV     AL, 58H                 ; Slaves1’s interrupt vector =28H
        OUT     DX, AL
        MOV     AL, 00000111B           ; Slave’s ID = 7
        OUT     DX, AL
        MOV     AL, 00000001B           ; Normal EOI, Non-Buffered Mode
        OUT     DX, AL
```

Cascading the 8259A



Cascading Multiple 8259As



8259A: Buffered Mode

- The **Buffered Mode** is useful in large systems where buffering is required on data bus.
- In **Buffered Mode**, whenever the 8259A's data bus output is enabled, its SP/EN will goes low. Therefore, this signal is used to enable the direction of (additional) bidirectional data buffers.
- Note that the pull-up on SP/EN, it is used to enable data transfer to the 8259A for its initial programming.
- In **Buffered Mode**, the user can program each individual 8259A as a master or a slave (**ICW4**).

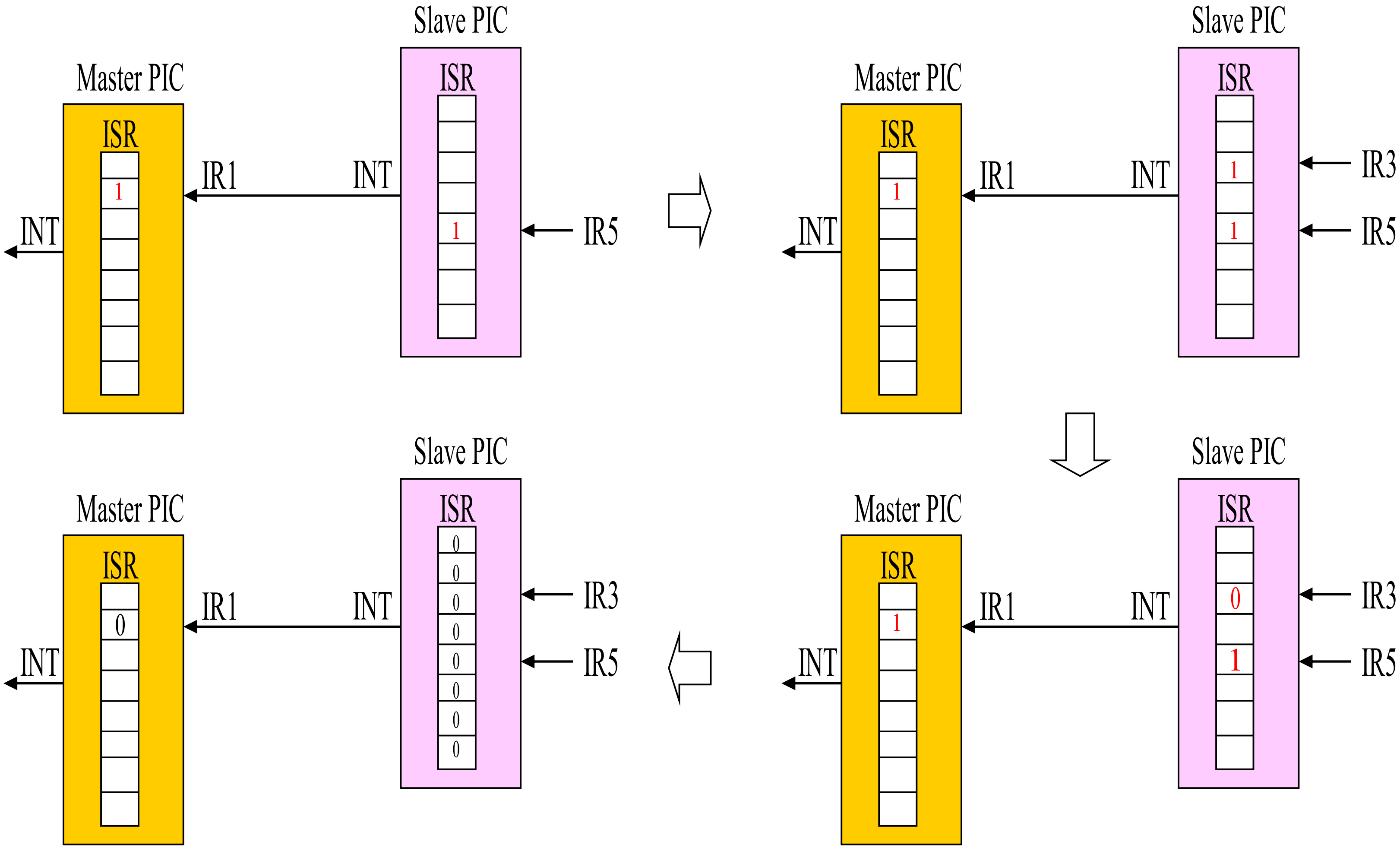
8259A: Special Fully Nested Mode

- In the cascade mode, if a slave receives a higher priority interrupt request than the one which is in service (through the same slave), it won't be recognized by the master. This is because the master's **ISR** bit is set, ignoring all requests of equal or lower priority.

- The **special fully nested mode** is programmed in the master only. In this mode the master will ignore only those requests of lower priority than the set **ISR** bit and will respond to all requests of equal or higher priority. Thus if a slave receives a higher priority request than the one in service, it will be recognized.

- The software must determine if any other slave interrupts are still in service before issuing an **EOI** command to the master. This is done by resetting the appropriate slave **ISR** bit with an **EOI** and then reading its **ISR**. If the **ISR** contains all zeros, there are not any interrupts from the slave in service and an **EOI** command can be sent to the master. If the **ISR** is no all zeros, an **EOI** should not be sent to the master.

8259A: Special Fully Nested Mode



Interrupt Service Routine of the Cascading Multiple 8259As (Special Fully Nested Mode)

```
IR_SR               SEGMENT  AT   0FE00H
                    ASSUME  CS:IR_SR, SS:STACK

                    ORG         0500H
First               LABEL       FAR
                    PUSH         DX
                    PUSH        AX

; SERVICE ROUTINE FOR IR
                    MOV         DX, SLAVE_PIC_PORT0                 ; Issuing Non-Specific EOI to slave
                    MOV         AL, 20H
                    OUT         DX, AL

                    MOV         AL, 00001011B                       ; Select ISR by using OCW3
                                OUT         DX, AL
                    IN          AL, DX                              ; Reading ISR
                    TEST        AL, 11111111B                       ; Testing ISR
                    JNZ         NOT_EOI_MASTER
                    MOV         DX, MASTER_PIC_PORT0                ; Issuing Non-Specific EOI to master
                    MOV         AL, 20H
                    OUT         DX, AL

NOT_EOI_MASTER:     POP         AX
                    POP         DX
                    IRET
NMI_SR              ENDS
```