CHAPTER 5

INTERRUPT OPERATIONS

INTRODUCTION

- An **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.
- Alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.
- The processor responds by suspending its current activities, saving its state, and executing a function called an *interrupt handler* (or an interrupt service routine, ISR) to deal with the event.
- This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.
- Used for data transfer between the peripheral and the microprocessor.
- Particularly useful when interfacing i/o devices that provide or require data at relatively low data transfer rate.
- These are ***asynchronous in nature***, i.e. It can be initiated at any time without reference of the system clock.

The most class of interrupts

- **Programs:**
  - Occurs during program execution, such as overflow, divide by zero, trying to execute illegal machine instruction and referencing memory outside a user’s allowed space.
- **Timers:**
  - Generated by a timer within the processor to perform certain function on regular basis.
- **I/O:**
  - Generated by I/O controller to signal general completion of task or to signal error.
- **Hardware failure:**
  - Power failure, or memory parity error.

Polling versus interrupt

Polling:

- Whereas, in polling, CPU steadily checks whether the device needs attention.
- Whereas it isn’t a hardware mechanism, its a protocol.
- The device is serviced by CPU.
- CPU polls the devices at regular interval.
- CPU has to wait and check whether a device needs servicing which wastes lots of CPU cycles.
- Command ready bit is used as indication for indicating that device requires servicing.
- ***Example:***
  - Constantly keep on opening the door to check whether anybody has come.

Interrupt

- In interrupt, the device notices the CPU that it requires its attention.
- Hardware mechanism.
- Interrupt handler services the device.
- Interrupt-request line indicates that device needs servicing.
- CPU is disturbed only when a device needs servicing, which saves CPU cycles.
- An interrupt can occur at any time.
- **E.g.**:
  - Let the bell ring then open the door to check who has come.

Interrupt structures

- Two ways of servicing multiple interrupts:
  1. Polled interrupts and
  2. Daisy chain (vectored) interrupts.

1. **Polled interrupts:**
   - Polling is to ask one by one.
   - Are handled using software and are slower.
   - The MPU identifies the interrupt generating device by executing special program which checks the devices.
   - The devices are placed according to priority.
   - In order to service an interrupt, the processor checks the starting with highest priority device. One the processor determines the source; it branches to the ISR for that device.

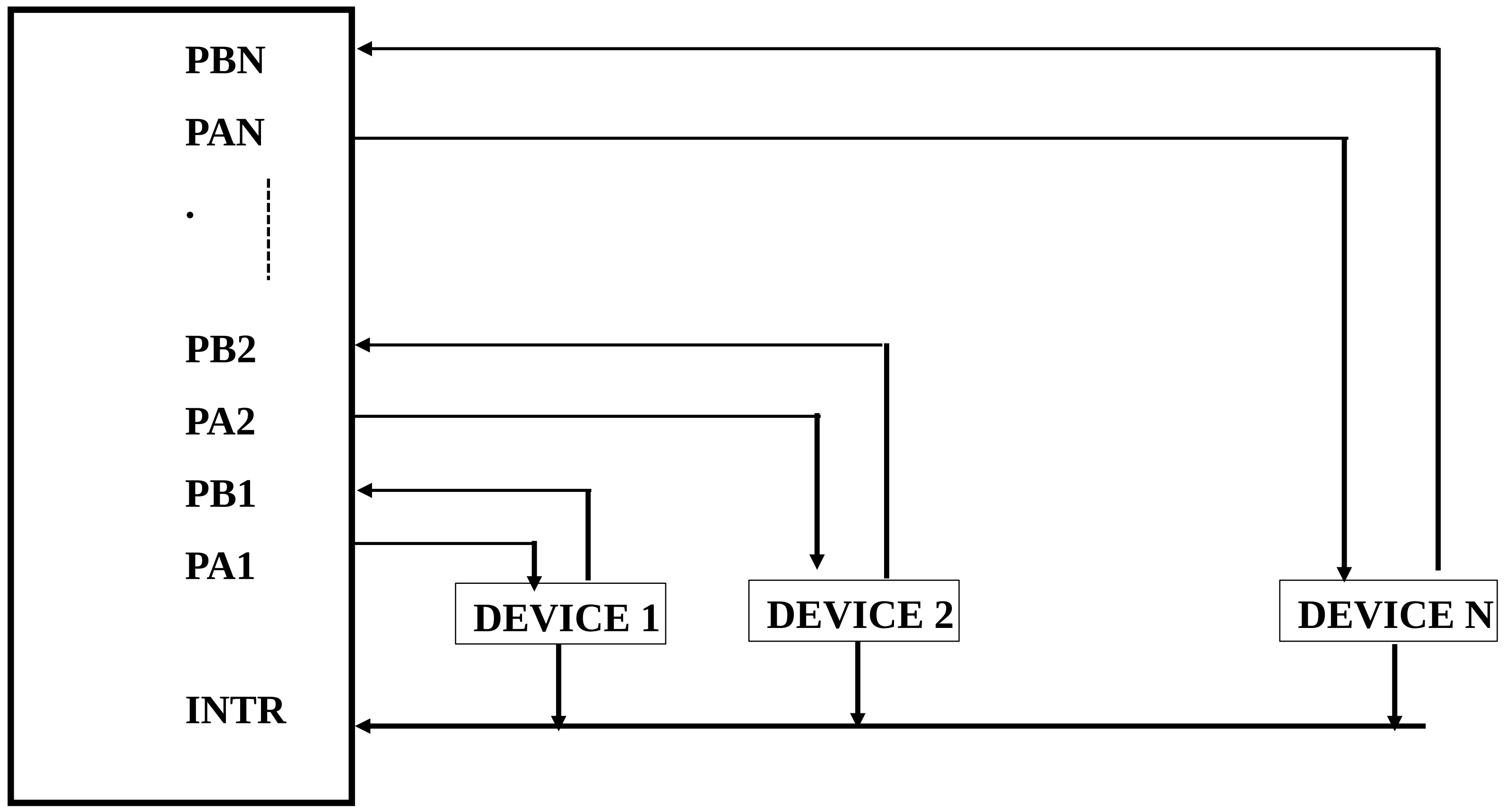


Fig: polled interrupt

2. Daisy chain (vectored) Interrupt

- Devices are connected in chain fashion as shown in figure below:
- Suppose one or more devices interrupt at a time, the MPU saves its current status and generates $\overline{INTA}$ signal to the highest priority device, (here device 1).
- If the device has interrupted the MPU, it accepts the $\overline{INTA}$ signal from MPU, otherwise it passes the signal to the next device until the signal get accepted by interrupting device.
- Once accepted, device provides a means to the MPU for finding the interrupt address vector using external hardware.
- With the help of hardware, t generates interrupt vector address which is used by MPU to execute ISR.

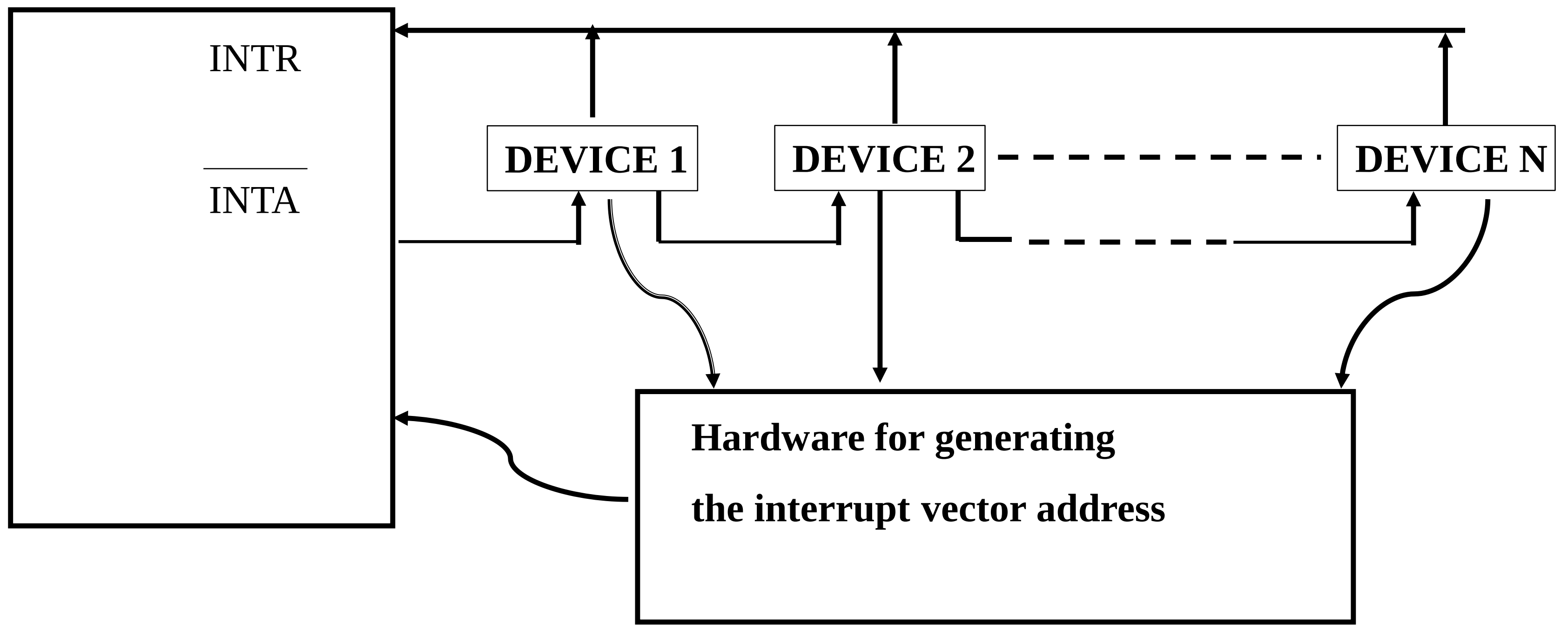


Fig: Vectored interrupt

Interrupt processing sequence

- The occurrence of interrupt triggers a number of events, both in processor hardware and in software.
- The interrupt driven I/O operation takes the following steps:
    1. The I/O unit issues an interrupt signal to the processor for exchange of data between them.
    2. The processor finishes execution of the current instruction before responding to the interrupt.
    3. The processor sends an acknowledgement signal to the device that it issued the interrupt.
    4. The processor transfers its control to the requested routine called “Interrupt Service Routine (ISR)” by saving the contents of program status word (PSW) and program counter (PC).
    5. The processor now loads the PC with the location of interrupt service routine and the fetches the instructions. The result is transferred to the interrupt handler program.
    6. When interrupt processing is completed, the saved register’s values are retrieved from the stack and restored to the register.
    7. Finally, it restores the PSW and PC values from the stack.

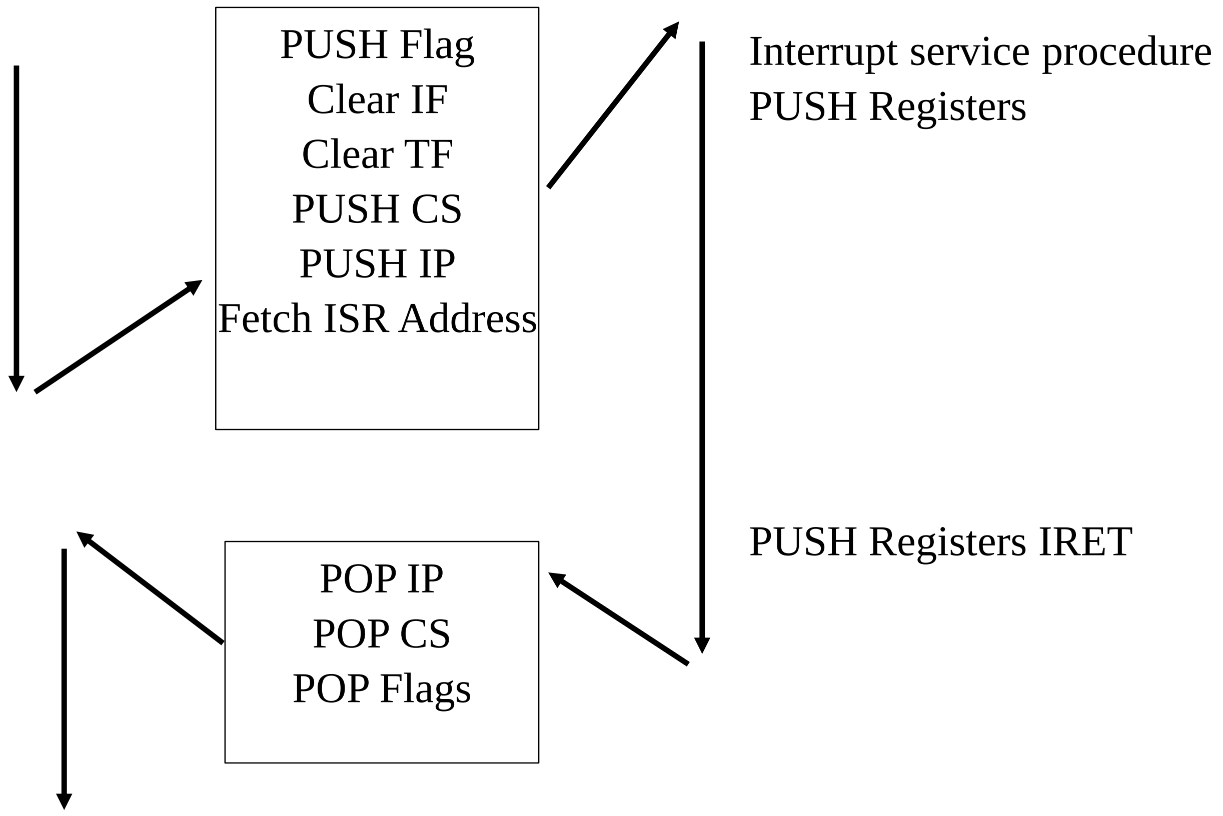


Fig: Interrupt processing

Multiple interrupts and priorities

- Two approaches to handle multiple interrupts:
    1. To disable interrupts while other interrupt is being served.
    2. Sequential processing of interrupts or prioritizing the interrupts

1) Sequential processing of interrupts:

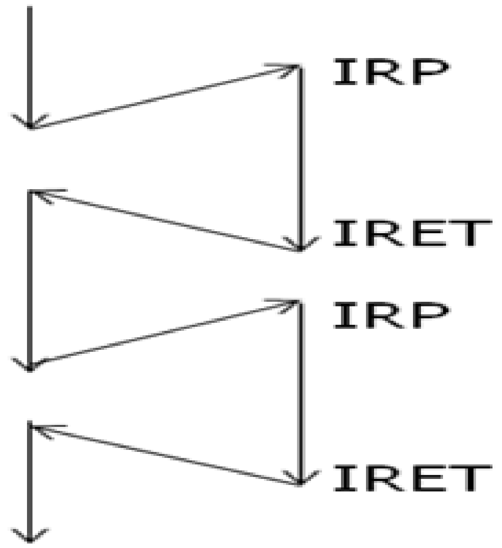


Fig: Sequential Interrupt Service

2) prioritizing the interrupts (Nested)

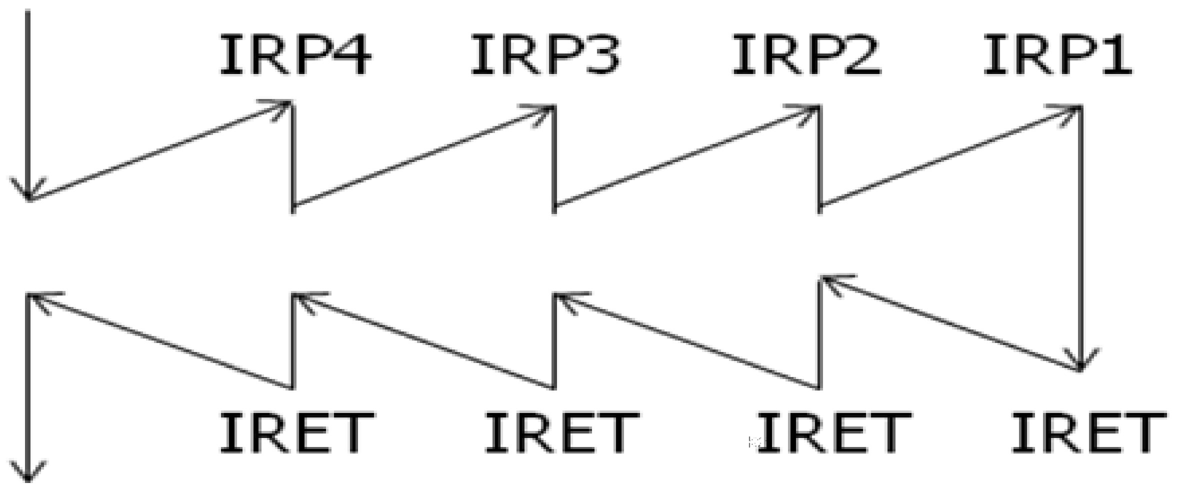


Fig: Priority wise Interrupt service

→ The drawback of sequential processing is that it does not take account of relative priority or time critical needs.

→ Type of interrupt processing that allows an interrupt of higher level to cause a lower priority interrupt pause until higher priority interrupt completes its function.

Interrupt Service Routine (ISR)

- A software routine invoked by an interrupt request from a hardware device.
- Handles the request and sends it to the CPU, interrupting the active process. When the ISR is complete, the process is resumed.
- Must perform very fast to avoid slowing down the operation of the devices and other lower priorities ISRs.
- ISR is responsible for doing the following things:
  - Saving the processor context
  - Acknowledging the interrupt
  - Restoring the processor context

Interrupt types

- Hardware and software
- Maskable and non-maskable
- Vectored and non-vectored
- External and internal

1. Hardware and software interrupts

- **Hardware interrupts:**
  - Interrupt source: external hardware
  - 5 types in **8085**: **INTR, RST7.5, RST6.5, RST5.5,TRAP**
  - 2 types in **8086: NMI,INTR**
- **Software interrupts:**
  - Mnemonics of microprocessor
  - 8 types in **8085**: RST0, RST1,……., RST7
  - 256 types in **8086: INT 00h,…, INT FFh**

2. Maskable and non-maskable interrupts

- **Maskable interrupts:**
  - Can be enabled or disabled by software; means we can enable or disable the interrupt by sending appropriate instruction like EI OR DI respectively.
  - INTR, RST 7.5, RST 6.5, and RST 5.5 are the examples of Maskable Interrupt OF 8085.
  - Level or edge triggered.
- **Non-Maskable interrupts:**
  - Cannot be enabled or disabled by sending any instruction.
  - TRAP interrupt is the non-maskable interrupt for 8085. It means that if an interrupt comes via TRAP, 8085 will have to recognize the interrupt we cannot mask it.
  - Used in critical power failure condition.
  - Both level and edged triggered.

3. Vectored and non-vectored Interrupt

- **Vector Interrupt:**
  - Processor knows the address of Interrupt. In other word processor knows the address of interrupt service routine.
  - The examples of vector interrupt are RST 7.5, RST 6.5, RST 5.5, TRAP.
- **Non-Vector Interrupt:**
  - Processor do not know the address of Interrupt.
  - It should be given externally.
  - The device should send the address of interrupt service routine to the processor for performing Interrupt.
  - The example of Non-vector interrupt is INTR.

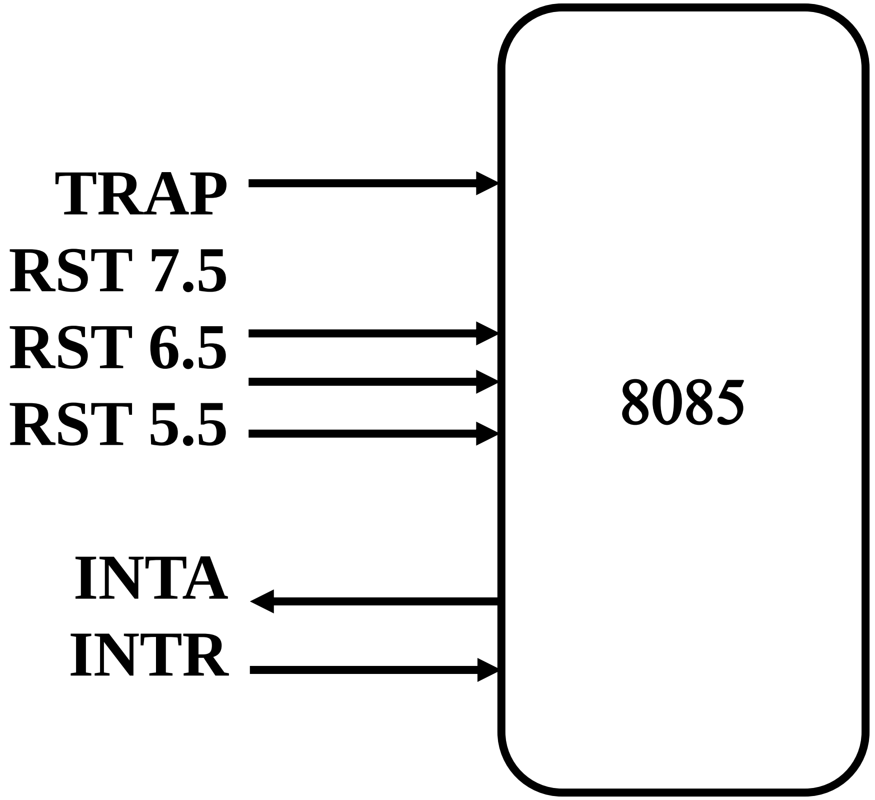
4. External and internal interrupts

- **External interrupts:**
  - Due to outside interference like from user, peripherals, from other hardware or networks.
- **Internal interrupts:**
  - An internal interrupt is a specific type of interrupt that is caused by instructions embedded in the execution instructions of a program or process.
  - Typically, internal interrupts resist changes by users, and happen "naturally" or "automatically" as a processor works through program instructions, rather than being caused by external events or network connections.

Interrupts in 8085

1. The interrupt process should be enabled by writing instruction EI in the main program. EI stands for enable interrupt, 1-byte instruction. DI (Disable Interrupt) 1-byte instruction, is used in complementary with EI which resets the flip-flop and disables the interrupt operation.
2. When MPU is executing a program, it routinely checks the INTR line during the execution of each instruction. INTR line will be raised if interrupt takes place.
3. If INTR line is high and the interrupt is enabled, the MPU completes current instruction, disable the interrupt enable flip-flop and issues $\overline{INTA}$ signal to device. The MPU will not be able to receive any other interrupt request until the flip-flop is enabled again.
4. The processor transfers its control to the requested routine called “Interrupt Service Routine (ISR)” by saving the contents of program status word (PSW) and program counter (PC) to the stack.
5. The service routine must include EI at its last to enable the interrupt process again. At the end of the ISR, RET instruction is initiated which retrieves the memory address of the stack where the program was interrupted and continue its execution.

Interrupt pins and priorities in 8085:



Types of 8085 interrupt

- Has multilevel interrupt system.
- **Software Interrupt:**
    - It is an instruction based Interrupt which is completely controlled by software.
    - That means programmer can use this instruction to execute interrupt in main program. There are eight software interrupts available in 8085 MP.
    - The vector address for these interrupts can be calculated as follows.

Interrupt number * 8 = vector address

For RST 5; 5 * 8 = 40 = 28H

Vector address for interrupt RST 5 is 0028H

Instruction	Corresponding hex code	Vector address
RST0	C7	0000H
RST1	CF	0008H
RST2	D7	0010H
RST3	DF	0018H
RST4	E7	0020H
RST5	EF	0028H
RST6	F7	0030H
RST7	FF	0038H

- **Hardware Interrupt:**
  - As name suggests it is interrupt which can get the interrupt request in hardware pin of microprocessor 8085.
  - There are mainly six dedicated pins available for interrupt purpose; TRAP, RST 7.5, RST 6.5, RST 5.5, INTR, INTA (It is not an Interrupt pin but it is used to send acknowledgement of the Interrupt request getting from other interrupt pin).

Name	Priority	Type	ISR location
TRAP	Highest	Vectored	0024H
RST7.5		Vectored	003CH
RST6.5	↓	Vectored	0034H
RST5.5		Vectored	003CH
INTR	lowest	Non-Vectored	

- **TRAP:**
  - Non-maskable interrupt, highest priority.
  - Need not to be enabled and cannot be disabled.
  - Level and edge sensitive, input should go high and stay high to be acknowledged.
  - When this interrupt is triggered, the program control is transferred to location 0024H without any external hardware.
  - The signal, which overrides the TRAP, is HOLD signal. (i.e., If the processor receives HOLD and TRAP at the same time then HOLD is recognized first and then TRAP is recognized).
  - Generally used for such critical events as power failure and emergency shutoff.
  - There are two ways to clear TRAP interrupt.
    1. By resetting microprocessor (External signal)
    2. By giving a high TRAP ACKNOWLEDGE (Internal signal)

- **RST7.5, RST6.5, RST5.5:**
  - These are maskable interrupts and can be enabled under program control with two instructions EI and SIM (Set Interrupt Mask).
  - RST7.5 is positive edge sensitive and can be triggered with short pulse.
  - RST6.5, RST5.5 are levelled sensitive, means that the triggering level should be on until the MPU completes the current instruction.
  - It is disabled by,

    1. ***DI instruction***
    2. ***System or processor reset.***
    3. ***After reorganization of interrupt.***
- **INTR:**
  - Maskable and Non-vectored interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.
  - Enabled by EI instruction and has lowest priority.
  - It is disabled by,

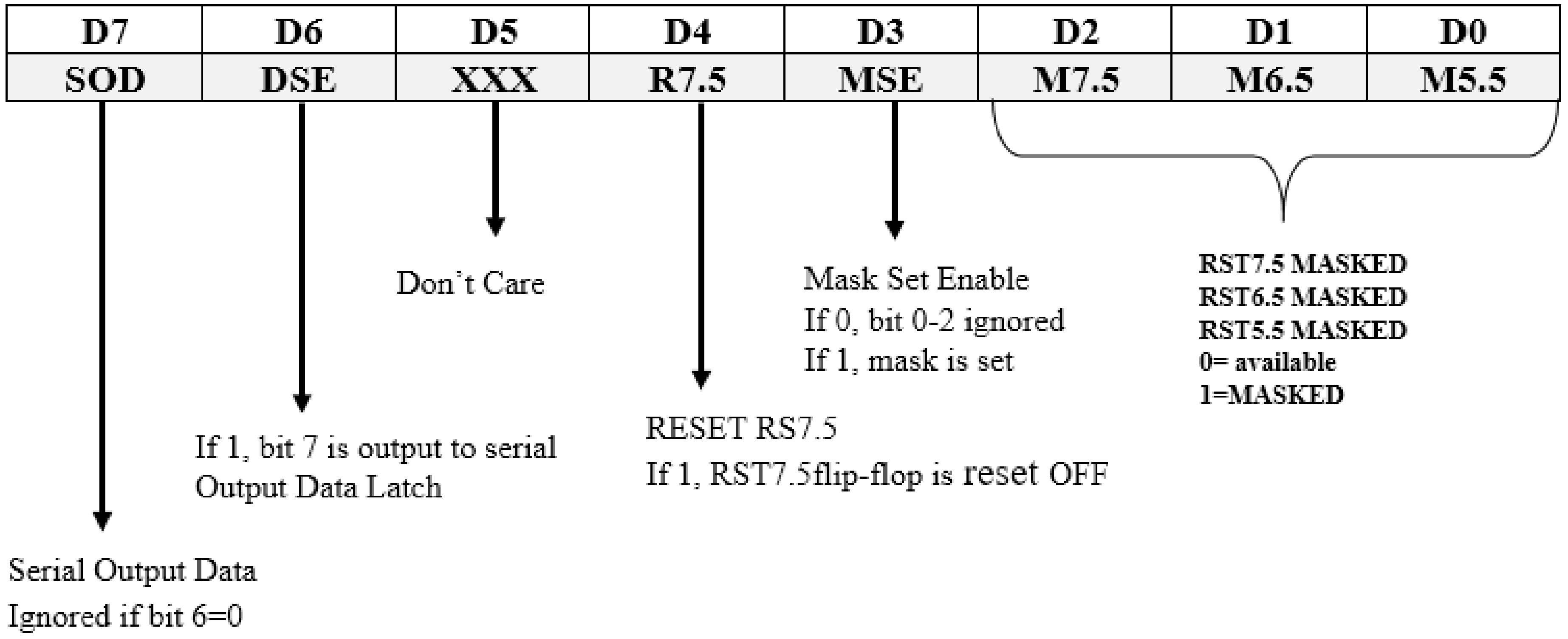
    1. ***DI, SIM instruction***
    2. ***System or processor reset.***
    3. ***After reorganization of interrupt.***

Interrupt Instructions:

1. **SIM (Set Interrupt Mask) instruction:**

- 1-byte instruction, & can be used for three different functions.
- One function is to set mask for RST 7.5, RST 6.5 and RST 5.5 interrupt.
- This instruction reads the content of accumulator & enables or disables the interrupt according to the contents of the accumulator.
- Bit D3 is the control bit and should=1 for bits D0, D1, D2 to be effective.
- Logic 0 on D0, D1, D2 will enable the corresponding interrupts, and logic 1 will disable the interrupt.
- The second function is to reset RST7.5 flip-flop. Bit D4 is additional control for RST7.5.
- If D4 = 1, RST7.5 is reset. This is used to override interrupts (or ignored) RST7.5 without servicing it.

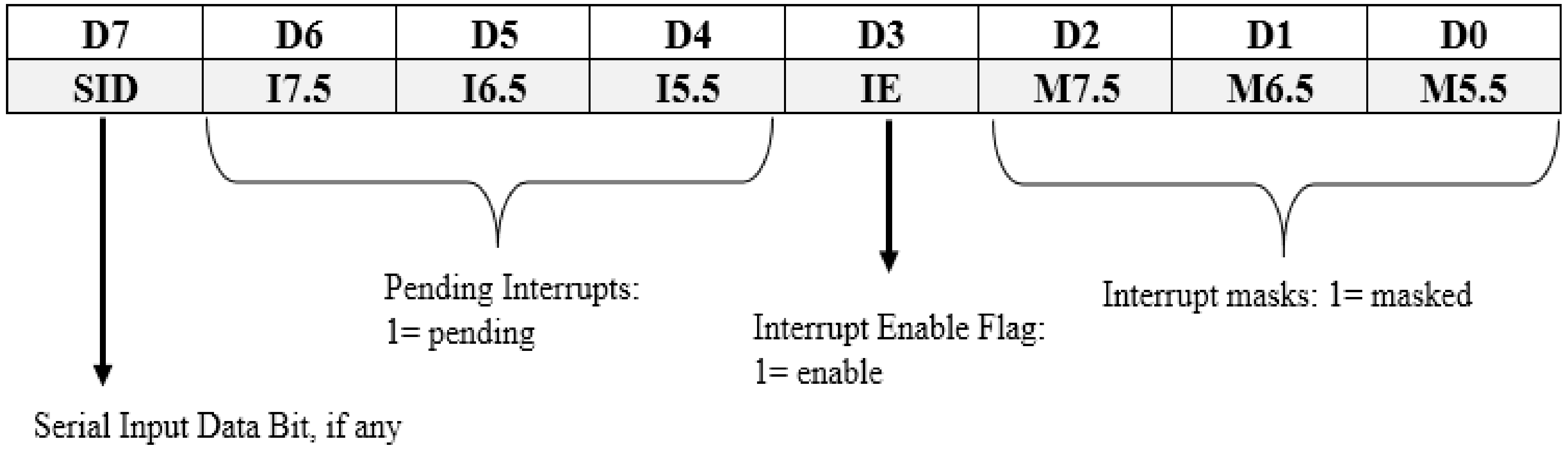
- The third function is to implement serial I/O.
- Bits D7 and D6 of the accumulator are used for serial I/O and do not affect the interrupts.
- Bit D6 =1 enables the serial I/O and bit D7 is used to transmit bits.



D7	D6	D5	D4	D3	D2	D1	D0
SOD	DSE	XXX	R7.5	MSE	M7.5	M6.5	M5.5

2. RIM (Read Interrupt mask) instruction:

- One byte instruction.
- To read interrupt masks.
- This instruction loads the accumulator with 8-bits including the current status of the interrupt masks.
- Bit D4, D5, and D6 identify the pending interrupts.
- To receive serial data bit D7 is used to receive serial data.



D7	D6	D5	D4	D3	D2	D1	D0
SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5

Interrupt Processing in 8086

- Interrupts in 8086 may come from three sources:
    1. Hardware: NMI or INTR
    2. Software
- The 8086 interrupt process can be described as:
    1. At the end of each instruction cycle, the 8086 checks to see if any interrupts have been requested.
    2. If there are any, the 8086 pushes the flag registers in stack.
    3. The 8086 disables the INTR input by clearing the interrupt flag (IF) and resets the trap flag (TF).
    4. The content of the current code segment register and instruction pointer are pushed on the stack.
    5. Then, the program control is transmitted to the ISR. An IRET instruction at the end of ISR returns the execution to the main program by retrieving the contents from stack which were pushed before.

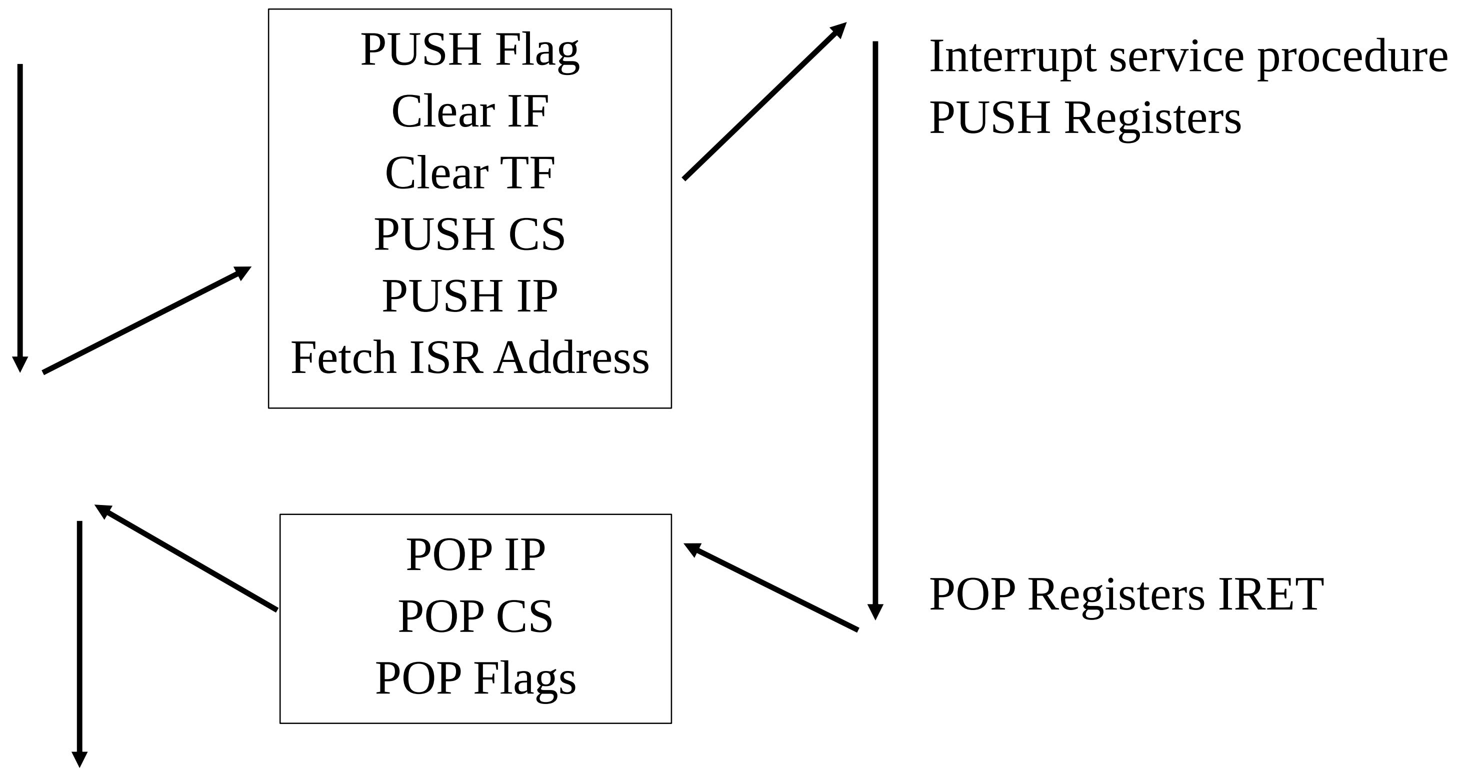


Fig: 8086 interrupt response

8086 Interrupt Types

- **Hardware (external) interrupts:**
  1. NMI (Non-Maskable Interrupt):
     1. Single pin non disabled
     2. Highest priority
     3. Generates TYPE-2 interrupt after execution
     4. IP located at **00008H** and CS is loaded from **0000AH**
  2. INTR (Interrupt request):
     1. Provides a single interrupt request and activated by I/O port.
     2. Can be masked or delayed
     3. Level triggered
     4. Can receive any interrupt type, so any IP and CS can get loaded.
     5. INTA is response pin

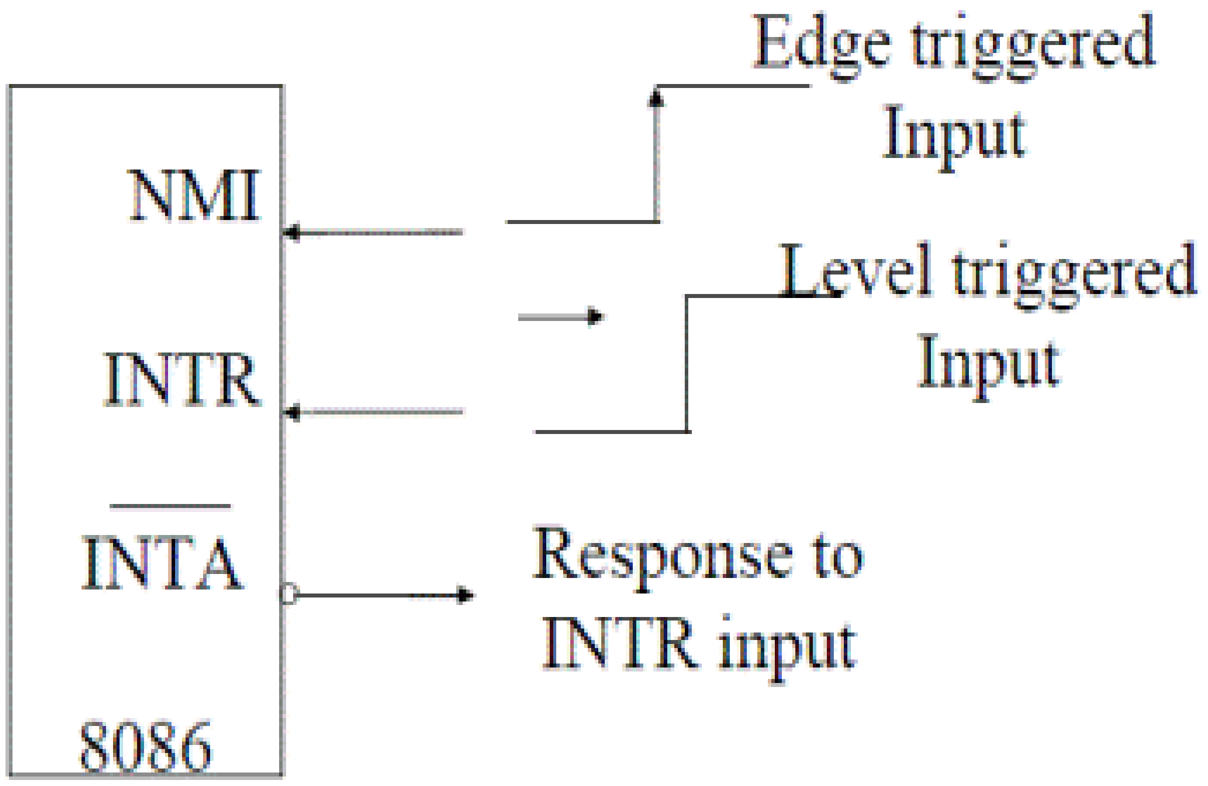


Fig: 8086 hardware interrupts

Software Interrupts (Internal Interrupts)

- Some instructions are inserted at the desired position into the program to create interrupts.
- 256 software interrupts
- INT format type ranges from INT 00h to INT FFh.
- Address ranges from 00000h to 003FFh.
- It is ***2-byte*** instruction. ***First byt***e provides the ***op-code*** and the *second byte* provides the ***interrupt type number***.
- IP is loaded from (type x 04h) and CS is loaded from the next address given by (type x 04h+02h).

- Some important software interrupts are:
  - ✓*TYPE 0* corresponds to division by zero(0).
  - ✓*TYPE 1* is used for single step execution for debugging of program.
  - ✓*TYPE 2* represents NMI and is used in power failure conditions.
  - ✓*TYPE 3* represents a break-point interrupt.
  - ✓*TYPE 4* is the overflow interrupt.
- **8086 Interrupt Priorities:**

Interrupt	Priority
Divide Error, INT(n),	HIGHEST
INT0	↓
NMI	
INTR	
Single step	LOWEST

Interrupt Vector Table (IVT)

- Feature of Intel 8086 family.
- An interrupt vector is a 4-byte long stored in the first 1024 byte locations of memory (00000h-003FFh).
- 256 different interrupt vectors.
- Each vector contains the address of the ISR.
- Each vector contains the IP and CS that forms the address of the ISR.
- The first 2-byte contains IP and last 2-byte contains CS.
- The IVT is a 1024 byte sized table that contains the address of ISR.
- The purpose od IVT is to hold the vector that redirects the processor to the right place when interrupt arrives.

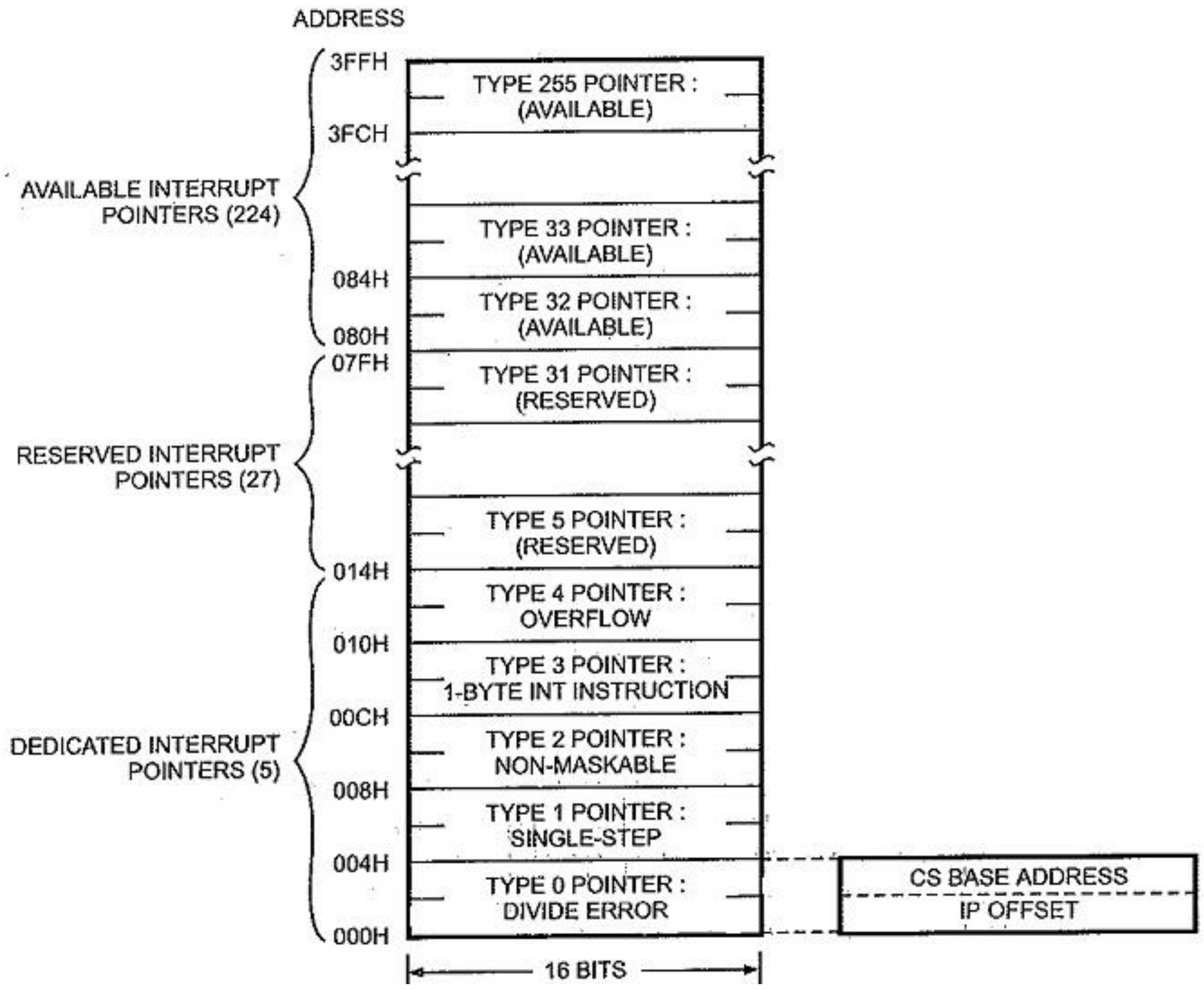

ADDRESS
3FFH
3FCH
084H
080H
07FH
014H
010H
00CH
008H
004H
000H
AVAILABLE INTERRUPT POINTERS (224)
RESERVED INTERRUPT POINTERS (27)
DEDICATED INTERRUPT POINTERS (5)
TYPE 255 POINTER : (AVAILABLE)
TYPE 33 POINTER : (AVAILABLE)
TYPE 32 POINTER : (AVAILABLE)
TYPE 31 POINTER : (RESERVED)
TYPE 5 POINTER : (RESERVED)
TYPE 4 POINTER : OVERFLOW
TYPE 3 POINTER : 1-BYTE INT INSTRUCTION
TYPE 2 POINTER : NON-MASKABLE
TYPE 1 POINTER : SINGLE-STEP
TYPE 0 POINTER : DIVIDE ERROR
CS BASE ADDRESS
IP OFFSET
16 BITS

Fig: IVT Structure (Organization)

- The interrupt number is used as an index into the table to get the address of the ISR.
- When 8086 responds to any interrupt type, it automatically multiplies the type by 4 to produce the desired address in the IVT.
- It then goes to that address in the table to get the starting address of the ISR.
- The lowest 5-types (TYPE-0 to TYPE-4) are dedicated to specific interrupts such as divide by zero-interrupt, the single-step interrupt, the non-maskable interrupt, breakpoint interrupt and the overflow interrupt.
- Interrupt TYPE-5 to TYPE-31 are reserved by Intel for use in more complex microprocessors.
- The upper 224 interrupt types from 32 to 255 are available for user to use in hardware and software interrupts.

DOS & BIOS interrupts:

- Dos interrupts services link applications with os services such as opening file, reading, writing content using certain functions of INT 4 H.
- BIOS interrupts control the screen disk controller and keyboard operation using INT 10H, 13H,16H etc.

Priority interrupt controller (PIC)

- The 8259A is a programmable interrupt-managing device, specifically designed for use with the interrupt signals (INTR/INT) of the Intel microprocessor 8085 and 8086.

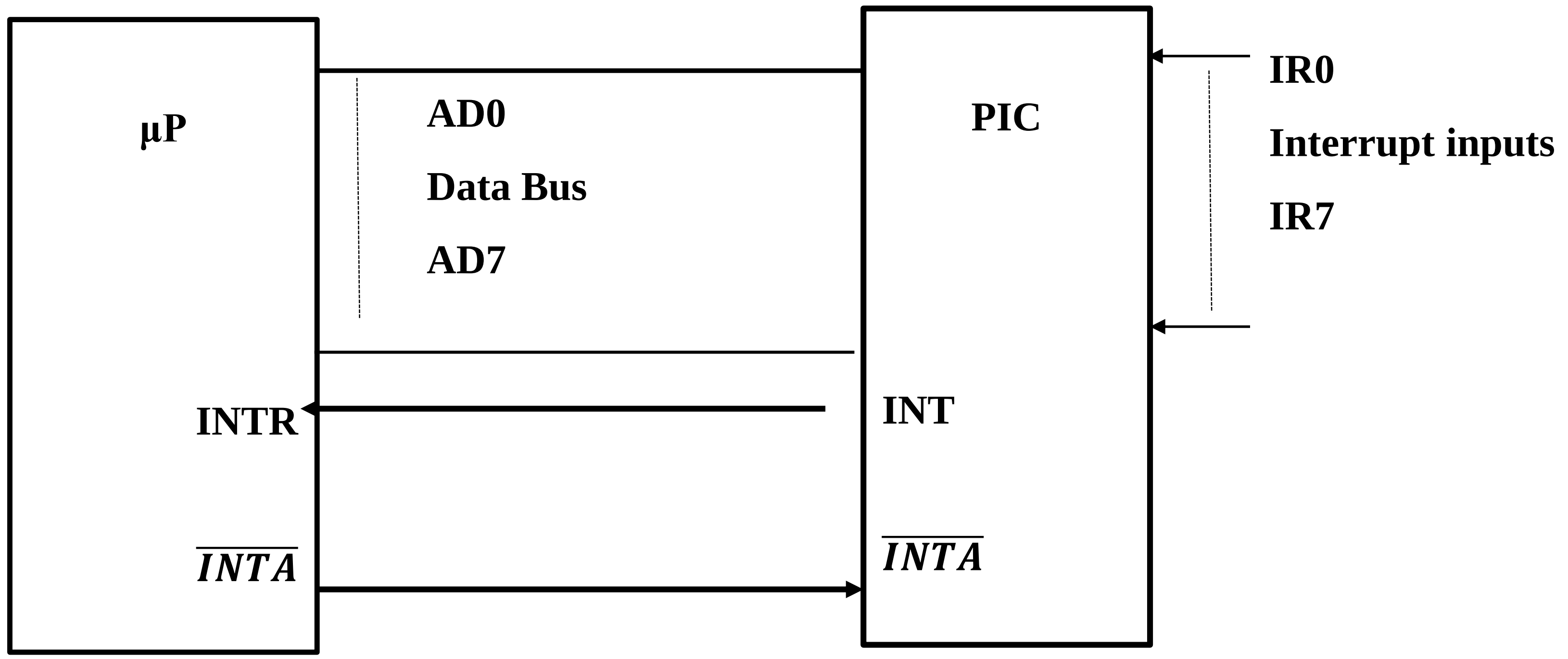


Fig: Multiple Interrupts using PIC

8259 Interrupt Controller

- Block diagram includes:
  - Control logic,
  - Registers for interrupt requests,
  - Priority resolver,
  - Cascade logic, and data bus.
  - The registers manage interrupt requests; the priority resolver determines their priority; cascade logic is used to connect additional 8259A devices.

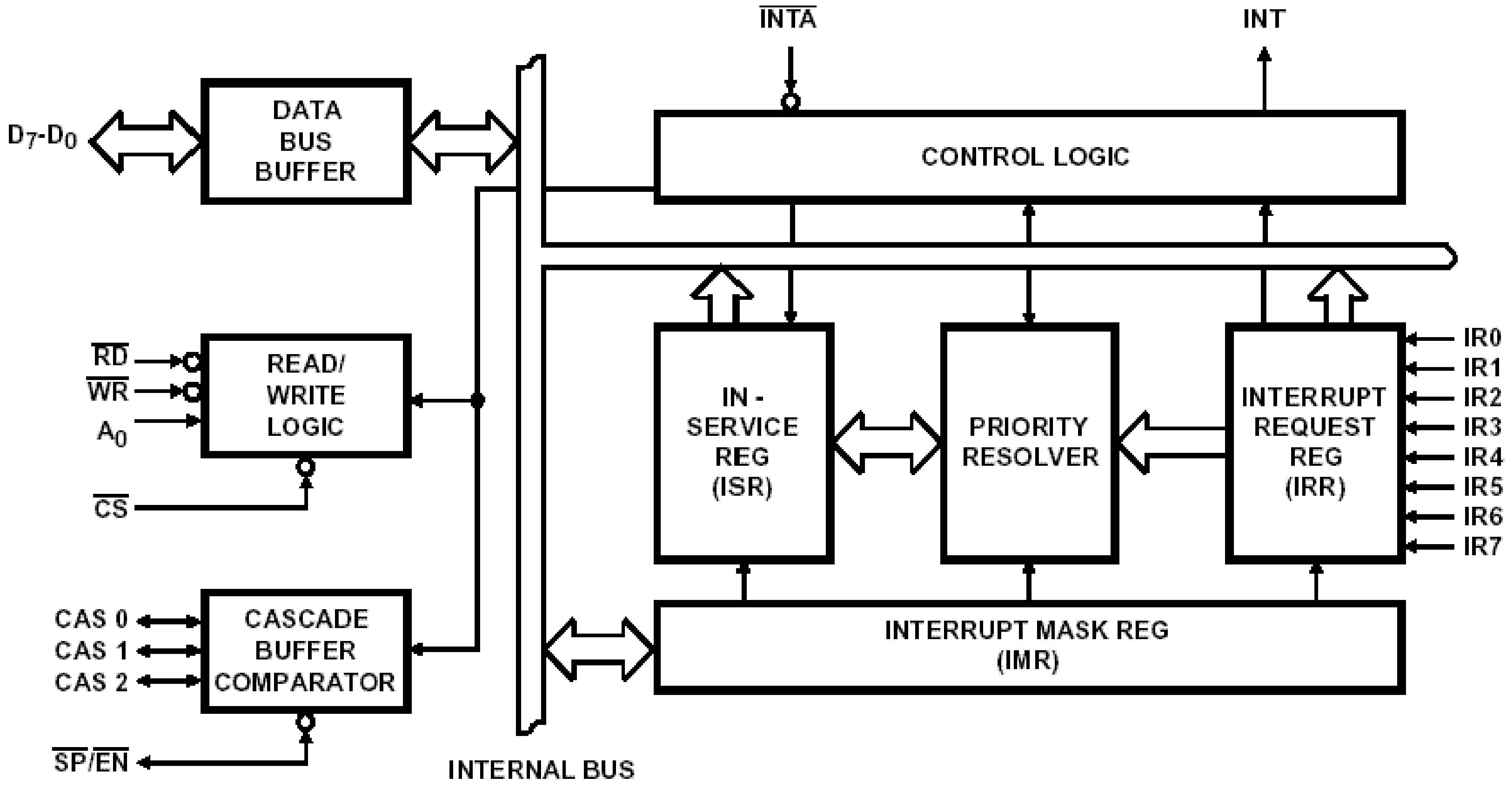


Fig: Block Diagram of 8259A PIC

The operation of 8259A:

i. One or more interrupt request lines go high requesting the service.
ii. The 8259A resolves the priorities and sends an INT signal to the MP.
iii. The MP acknowledges the interrupt by sending INTA(bar).
iv. After the INTA(bar) has been received, the op-code for the call instruction (CDH) is placed on the data bus.
v. Because of the CALL instruction, the MP sends two more INTA(bar) signals.
vi. At the first INTA(bar), the 8259A places the low-order 8-bit address on the data bus and at the second INTA(bar), it places the high-order 8-bit address of the interrupt vector. This completes the 3- byte CALL instruction.
vii. The program sequence of the MP is transferred to the memory location specified by the CALL instruction.

Priority modes

1. **Fully Nested mode**
   - IR0 has the highest priority and the following IR1, IR2, IR3…… etc. have the decreasing priorities.
2. **Automatic rotation mode**
   - First priority changes to the last after its service.
3. **Specific rotation mode**
   - This is user selectable or programmable, which means priority can be selected by programming.

Features

i. It manages 8 interrupt requests.
ii. It can vector an interrupt request anywhere in the memory map through program control without additional hardware for restart instructions. However, all 8 requests are spaced at the interval of either 4 locations or 8 locations.
iii. It can solve 8 levels of interrupt priorities in a variety of modes.
iv. With cascading additional 8259A devices, the priority scheme can be expanded to 64 levels.
v. The 8259A has the abilities such as reading the status and changing the interrupt mode during a program execution.
vi. It can mask each interrupt request individually.
vii. It can be set up to work with either the 8085 MP mode or the 8086/8088 MP mode.