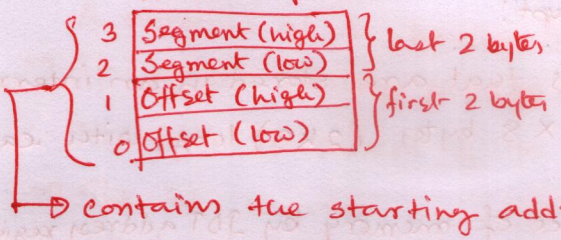
# Interrupt



- An interrupt is a hardware-initiated procedure that interrupts whatever program is currently executing.
- Useful when interfacing with I/O devices that provide or require data at relatively low data transfer rates (for example the case of keyboard example in the last lecture)
- Two H/W pins for requesting interrupts: INTR and NMI
  One H/W pin for acknowledging interrupt requested by INTR: INTA
  S/W interrupts: INT, INT0, INT3, and BOUND [INT0 and BOUND ⇒ conditional]
  Two flag bits: IF (Interrupt flag) and TF (Trap flag)
  A special return instruct: IRET (or IRETD in 80386, 80486, and ~~Pentium~~ Pentium 4)
- Interrupt vector: contains the (segment and offset) address of the interrupt service procedure

Interrupt vector table: Located in the first 1024 bytes of memory at address 000000H - 0003FFH. It contains 256 different 4 bytes int. vectors.

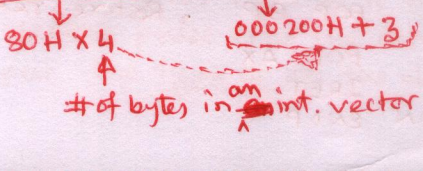
An interrupt vector:




⇒ First 32 interrupt vectors: Reserved
⇒ Other 224 interrupt vectors: User int. vectors

- Examples of dedicated interrupts:
  ⇒ Type 0: Divide error - Division overflow of divide by zero
  ⇒ Type 1: Single-Step or Trap - Occurs after execution of each instruction if TF flag is set. Upon accepting this int., TF is cleared to execute the int. service procedure at full speed.
  ⇒ Type 2: NMI - If 1 is placed on the NMI input pin; cannot be disabled
  ⇒ Type 3: One-Byte interrupt - A special one byte instruct (INT 3); often used to store a breakpoint in a program for debugging
  ⇒ Type 4: Overflow - INTO instruct interrupts if an overflow condition exists as reflected by OF (Overflow flag)
  ⇒ Type 5: BOUND - compares a register with boundary stored in memory. If 1st word in mem ≤ reg ≤ 2nd word in memory, no interrupt occurs. Otherwise, an interrupt occurs.

- INT instruction: INT n instruction calls the interrupt service procedure that begins at the address represented in vector number n.
  ⇒ Ex: INT 80H or INT 128 calls int. service procedure whose address is stored in vector type number 80H (000200H - 000203H)



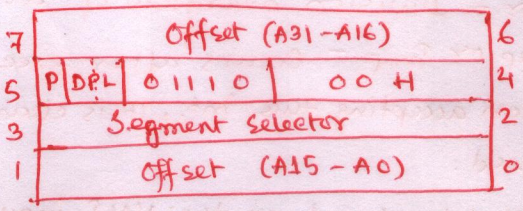
⇒ Each INT instruction is stored in 2 bytes of memory - first byte for opcode, and the second byte for int. type number.

⇒ Only excep$^n$: INT 3 - 1-byte instruc$^n$, used as breakpoint interrupt; It is 1 byte as it is easy to insert a 1-byte instruc$^n$ into a program

⇒ IRET: Removes six bytes from the stack - 2 for IP, 2 for cs, and 2 for flags
→ Return address for an interrupt: Next instruc$^n$ in the program
↳ " " " BOUND " : BOUND " itself, NOT the next instruc$^n$ (excep$^n$)
↳ similar excep$^n$ applies for INT type # 0, 5, 6, 7, 8, 10, 11, 12, and 13

- Operation of a Real Mode Interrupt:
  1. Push flag registers in stack
  2. Clear both IF and TF
  3. Push cs (code segment register) in stack
  4. Push IP (instruc$^n$ pointer) in stack
  5. Fetch the interrupt vector contents, and place into both IP and cs so that the next instruc$^n$ executes at the int. service procedure addressed by the vector.

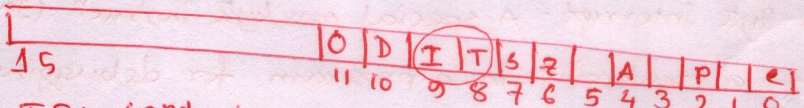
- Operation of a Protected Mode Interrupt:
  - Uses a set of 256 interrupt descriptors that are stored in an interrupt descriptor table (IDT). The IDT is 256 x 8 bytes (2KB) long, with each descriptor containing 8 bytes.
  - The IDT can be located at any place of memory by IDT address register (IDTR)
  - An entry in IDT:




- P bit: Present
- DPL bits: Privilege level of the interrupt

  - Difference with Real Mode: MP accesses IDT instead of Int. Vector Table (used in Real Mode)

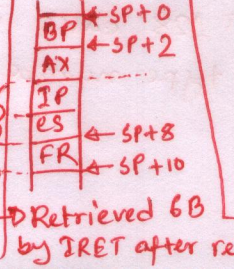
- Flag Register (FR):




- Clearing IF and TF: (2nd step in operation in Real Mode Int.)
  - Set or clear IF: STI and CLI instructions
  - Set or clear TF: No special instruc$^n$ (Interrupt service procedures are shown below)

```
TRON PROC NEAR
    PUSH AX ; Save Reg
    PUSH BP
    MOV BP, SP ; Get SP
    MOV AX, [BP+8] ; Get flags from stack**
    OR AH, 1 ; Set TF
    MOV [BP+8], AX ; Save flags
    POP BP ; Restore reg
    POP AX
    IRET
TRON ENDP
```

```
TROFF PROC NEAR
    PUSH AX
    PUSH BP
    MOV BP, SP
    MOV AX, [BP+8]
    AND AH, FEH ; clear TF
    MOV [BP+8], AX
    POP BP
    POP AX
    IRET
TROFF ENDP
```



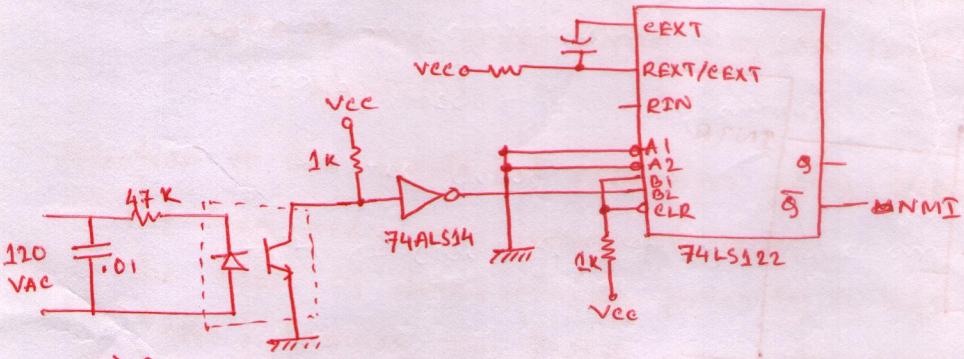

** Retrieved 6B by IRET after returning

# Hardware Interrupts

- NMI and INTR

→ NMI: (+)ve edge triggered (0-to-1, ⌐); After a positive edge, NMI must retain 1 until it is recognized by the µp

→ NMI is used for parity errors and other major system faults such as power failure (below).




→ An optical isolator provides isolation from the AC power line.

→ Output of the isolator is shapped by a Schmitt-trigger inverter to feed in 74LS122 monostable multivibrator. As long as AC power is applied, Q remains at 1 having $\overline{Q}$ at 0.

→ If power fails, 74LS122 no longer receives trigger pulses from 74ALS14. It makes $\overline{Q}$ to go 1, interrupting the µp by NMI pin

→ The int. service procedure (not shown in figure) stores contents of all internal registers and other data into a battery-backed-up memory.

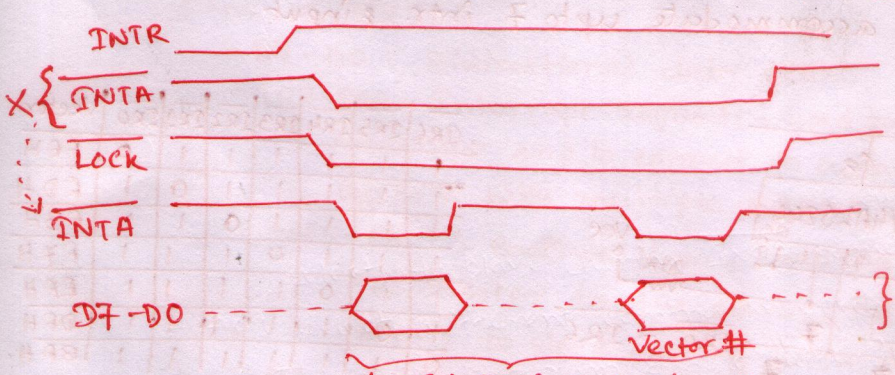
→ It is assumed that the system power supply has a large enough filter capacitor to provide energy for at least 75 ms after the AC power ceases.

- INTR and $\overline{\text{INTA}}$

- INTR is level-sensitive (must be held 1 until recognized

- INTR is set by an external event and cleared inside the interrupt service procedure. It is automatically disabled once it is accepted by the µp and re-enabled by the IRET instruction at the end of the intr. service proc.

- The µp responds to the INTR by pulsing the $\overline{\text{INTA}}$ output for receiving an intr. vector type number on the data bus D7-D0.

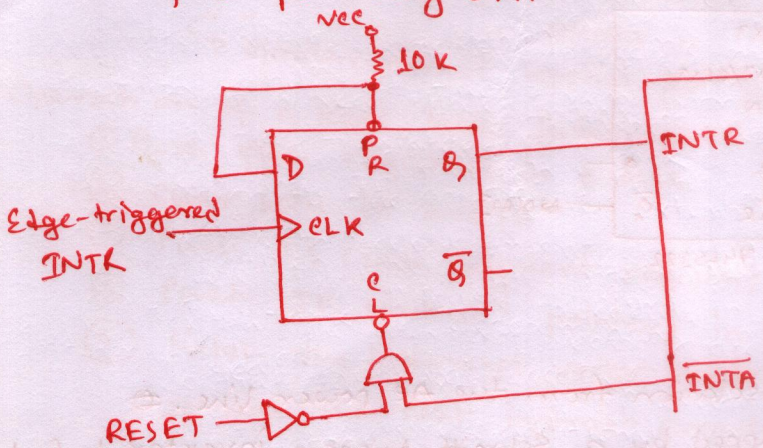



used to Lock peripheral off the system

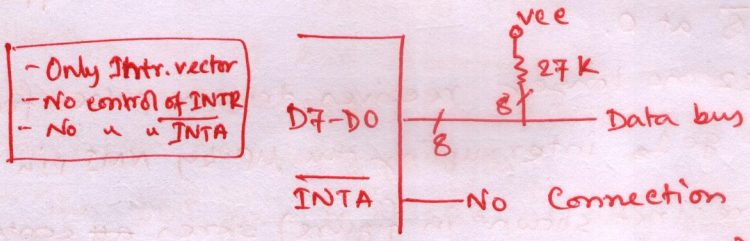
(**) The first pulse signals external ckt that the INTR has been acknowledged, and to send its type # to the µp. The second pulse tells the external ckt to put the type # on the data bus.

1 byte of vector type# placed on the data bus in response to $\overline{\text{INTA}}$ pulses (NOT 2 bytes type #)

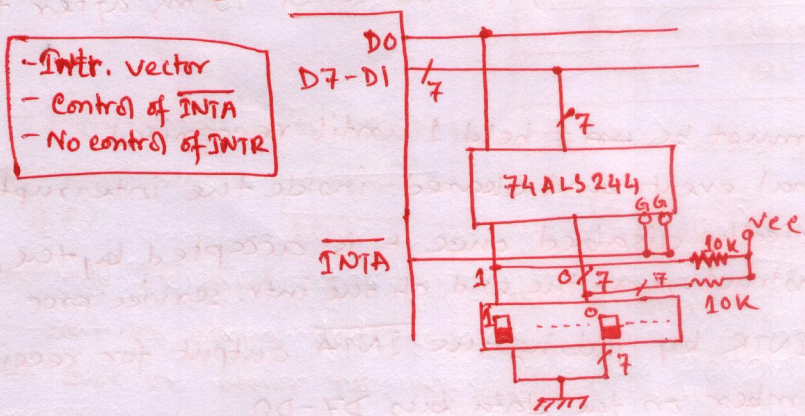
- Making the INTR input Edge-triggered:
  - → INTR is converted to edge-triggered input by using a D-FF.
  - → Clock serves the edge-triggered INTR input, and clear input (bottom of FF) is used to clear the request when $\overline{INTA}$ is output by the µP.
  - → RESET initially clears the FF so that no interrupt is requested during first powering on.



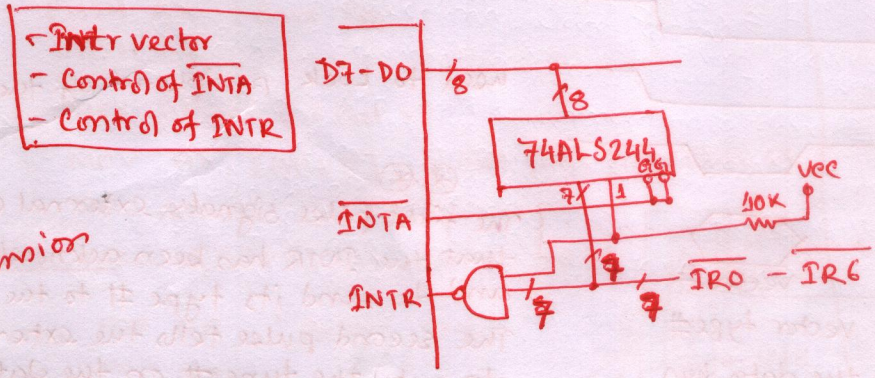

- A simple ckt to apply a fixed intr. vector type number FFH (always)




- Using a 3-state buffer (≡ 74ALS244) for $\overline{INTA}$ to place a vector type number 80H (1 intr. type #) in response to $\overline{INTA}$ pulse




- Using 74ALS244 to expand to accommodate upto 7 intr. ip input

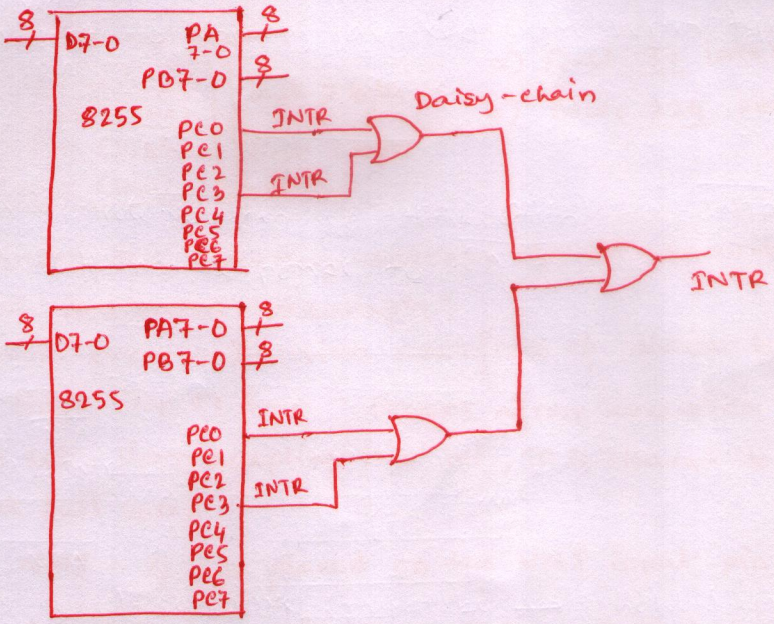



Expansion

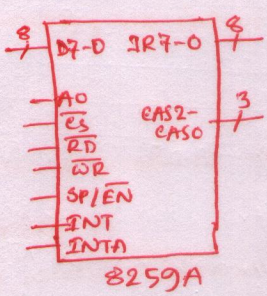
| IR6 | IR5 | IR4 | IR3 | IR2 | IR1 | IR0 | Vector |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | FEH |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | FDH |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | FBH |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | F7H |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | EFH |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | DFH |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | BFH |

- If two or more intr. requests inputs are activated simultaneously, a new vector is generated. For example, if $\overline{IR1}$ and $\overline{IR0}$ are activated simultaneously, the vector generated is FcH. Here, priority needs to be resolved at FcH (if $\overline{IR0}$ is the higher priority, then its vector address needs to be stored at FcH).
- Thus, $2^7$ (=128) locn's must be used to accommodate all possible cond^n's of the seven intr. request inputs.
  - Wasteful

- Solution to the wasteful extension using 74ALS244: Daisy-chained Interrupt
  - Requires only one interrupt vector
  - The task of determining priority is left to the intr. service proc.
  - Idea: Remove the INTR generation logic to s/w (in intr. service proc.)
    - Requires additional s/w execn time, but, much better in general




- 8259A (Programmable Interrupt controller): Enables further extension
  - Adds eight vectored priority encoder interrupts to the µp
  - Can be expanded to accept up to 64 interrupt requests without additional h/w. This expansion requires a master 8259A and eight 8259A slaves.
  - Pins of 8259A
    - D7-D0: Bidirectional data connn
    - IR7-IR0: Interrupt requests; used to request an interrupt and to connect to a slave ~~in~~ in a system with multiple 8259A
    - CAS2-CAS0: cascade lines; used as outputs from the master to the slaves. This bus acts like chip selects to slave during $\overline{INTA}$ sequences.
    - A0: Selects different command words within 8259A
    - SP/$\overline{EN}$: (Slave program/Enable buffer) - a dual funcn pin. In buffered mode, it controls the bus transciever. In non-buffered mode, it programs the device as a master (1) or a slave (0)



⇒ Command words :

① Initialization command words (ICW) : programmed before 8259A is able to function and these dictate basic operations of the 8259A.

- Four ICWs : ICW1, ICW2, ICW3 (Required only in cascading mode), and ICW4 (used in 8086 – Pentium 4 ; NOT used in 8085).

② Operation command words (OCW) : used to direct the operation of 8259A once it is programmed with ICWs.

- Three OCWs.

→ Details of CWs : self study (have an idea on them)