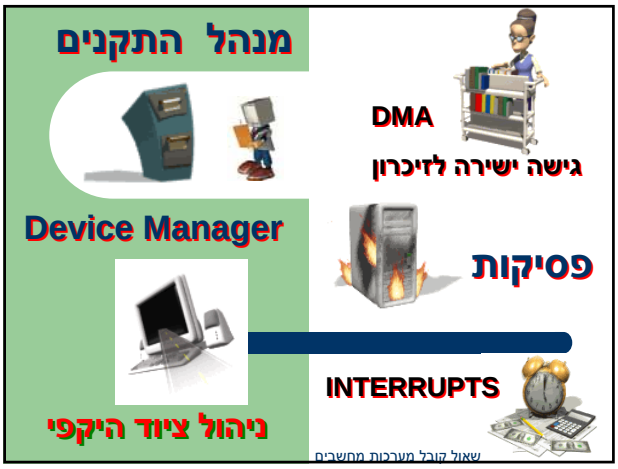
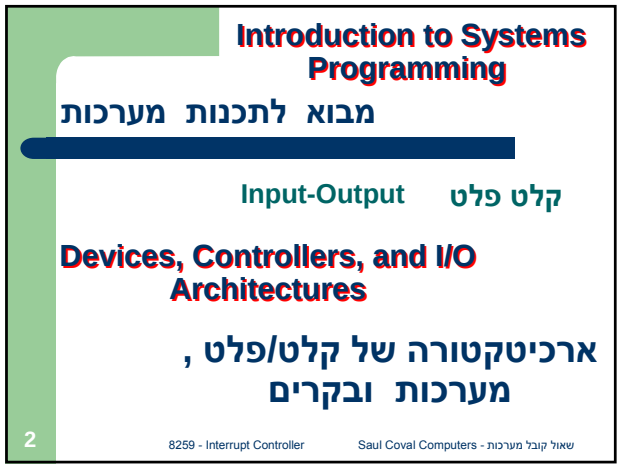
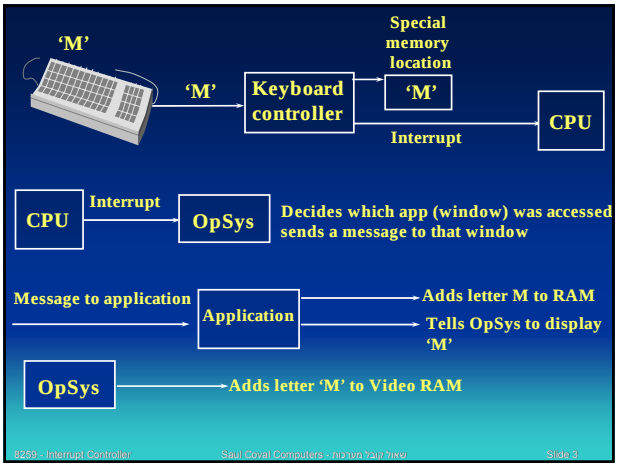






Why are interrupts important

- Interrupts let you use the operating system (run your programs, manage your files, access your peripherals etc.)
- Interrupts help peripherals "talk" to your microprocessor
- Interrupts help you measure time and control the timing of certain tasks in your microprocessors

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 4

Interrupts from a pedagogical perspective

- By learning interrupts you learn important concepts such as:
  - Concurrency: how your processor manages to service interrupts while your program doesn't know anything about them and how multiple interrupts are serviced at the same time
  - Preemptability and priorities, how can a low priority-task be preempted by a high-priority task
  - Scheduling: how can we assure that both low and high-priority tasks get the service they deserve from the processor

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 5

The INT and IRET instructions

- Syntax: INT imm8
- imm8 is an interrupt vector from 0 to 255
- INT does the following:
  - Pushes flag register (pushf)
  - Pushes return CS and IP
  - Far jumps to [0000:(4*imm8)]
  - Usually clears the interrupt flag disabling the interrupt system
- IRET is to INT what RET is to CALL
  - Pops flag register
  - Performs a far return

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 6

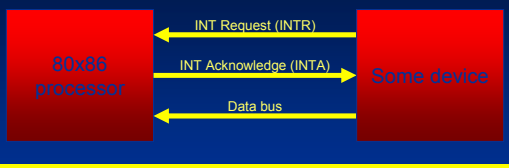
Things to notice

- The interrupt vector table is just a big permanently located jump table
- The values of the jump table are pointers to code provided by bios, hardware, the operating system or YOU!
- Interrupt service routines preserve the flags – the state of the microprocessor before the INT should be completely unaltered by the ISR and your program must return to normal operation.

Hardware interrupts

- Alert the processor of some hardware situation that needs the processor's attention
  - A key has been pressed
  - A timer has expired
  - A network packet has arrived
- Same software calling protocol
- Additional level of complexity with the interrupt "call" not coming from your program code
- Can happen at any time during the execution of your program, invocations of ISRs for hardware interrupts are *asynchronous*

The 80x86 interrupt interface

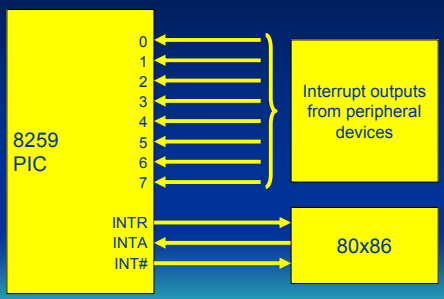


- Device generates request signal
- Device supplies interrupt vector number on data bus
- Processor completes the execution of current instruction and executes ISR corresponding to the interrupt vector number on the data bus
- ISR upon completion acknowledges the interrupt by asserting the INTA signal

It is not that simple…

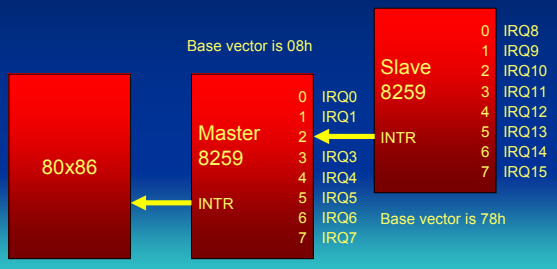
- What if we want to connect more than one devices to the processor ?
- What happens if multiple devices generate multiple interrupts at the same time ?
- We need a way to share the two interrupt lines among multiple devices
- 8259 Programmable Interrupt Controller
- The 8259 PIC operates as an arbiter for interrupts triggered by multiple devices
- One 8259 serves up to 8 devices, but multiple 8259 chips can be "cascaded" to serve up to 64 devices

The 8259 PIC



Interrupt vectors and the 8259 PIC

Master-Slave Configuration



The 8259 PIC

- PIC is very complex to program, fortunately the BIOS does most of the work needed
- Programmed with the I/O address 20h-21h (master) and 0A0h-0A1h (slave)
- I/O instructions yet to be discussed…
  - in reads from an I/O address
  - out writes to an I/O address
- Consider them as two registers the status register and the interrupt mask register

The 8259 PIC

- The mask register is addressed from 21h
- It lets you enable/disable specific hardware interrupts
- Counterintuitive: a 0 ENABLES an interrupt and a 1 DISABLES the interrupt
- Never load a value immediately to the mask register
- Always read the previous value and use and/or instructions to set the new mask

```
in al, 21h  ; this one reads the value of the mask register
and al, 0efh ; this zeroes out bit 4 i.e. IRQ4
out 21h, al   ; this actually disables the interrupt in IRQ4
```


The 8259 PIC

- When an interrupt occurs and the processor starts executing the ISR all further interrupts from the same device are blocked until the ISR issues an end of interrupt instruction

```
mov al, 20h
out 20h, al
```

- You must end exactly one interrupt!
  - Not sending one will block all interrupts from the save device
  - Sending two or more means that you might accidentally acknowledge the end of a pending interrupt!
- Two more registers track pending interrupts received at the PIC and interrupt priorities
- You must be careful when you're patching existing ISR's (because the end instruction sequence may already be included in the ISR)

The 8259 PIC

- IRQ mapping
  - Interrupt vectors 8 through 0Fh map to IRQ0-IRQ7
  - Interrupt vectors 70h-77h map to IRQ8-IRQ15

Typical IRQ assignments

- IRQ 0: Timer (triggered 18.2/second)
- IRQ 1: Keyboard (keypress)
- IRQ 2: Slave PIC
- IRQ 3: Serial Ports (Modem, network)
- IRQ 5: Sound card
- IRQ 6: Floppy (read/write completed)
- IRQ 8: Real-time Clock
- IRQ 12: Mouse
- IRQ 13: Math Co-Processor
- IRQ 14: IDE Hard-Drive Controller

Interrupt priority

- Lower interrupt vectors have higher priority
- Lower priority can't interrupt higher priority
- Higher priority can interrupt lower priority
  - ISR for INT 21h is running
    - Computer gets request from device attached to IRQ8 (INT 78h)
    - INT 21h procedure must finish before IRQ8 device can be serviced
  - ISR for INT 21h is running
    - Computer gets request from Timer 0 IRQ0 (INT 8h)
    - Code for INT 21h gets interrupted, ISR for timer runs immediately, INT21h finishes afterwards

Priority in the 8259

- 8259 supports several priority schemes
- On PC's the 8259 uses the simplest form of fixed priorities
- Each IRQ has a fixed priority
- Lower IRQs has higher priority
- The timer interrupt (IRQ0) has lower priority than any other IRQ
- If you really need higher priority than the timer (e.g. connecting a nuclear reactor to your microprocessor) it is possible to use a NMI (non-maskable interrupt)
- NMI has the highest priority among all hardware interrupts and cannot be disabled by the program

Interrupt enabling/disabling

- You can enable/disable all maskable hardware interrupts
- The CLI instruction disables all maskable hardware interrupts
- The STI instruction enables all maskable hardware interrupts
- Be very careful if you ever need to use them
  - Many deadlock scenarios!

The ugly details

- ISRs for hardware interrupts clear the interrupt flag at the beginning to disable interrupts. They may include a STI instruction if they want to enable interrupts before they finish
  - It's all about performance! Keeping interrupts blocked for long is a BAD IDEA
- ISRs for software interrupts do not disallow hardware interrupts automatically at the beginning. If an ISR for a software interrupt needs to do that it must issue a CLI instruction
  - This is what most ISRs do
  - Again for the sake of performance a STI instruction must be issued as soon as possible
  - Note that when interrupts are enabled the priority rule applies
- The CLI works only for maskable hardware interrupts
- Code enclosed between CLI/SCI is often called a *critical section*, an uninterruptible piece of code

Is there a way out of this mess ?

- In many critical section situations (e.g. patching the interrupt vector tables) DOS helps us ensure the required atomicity
- Convenient calls for
  - Safely getting the value of the interrupt vector from the interrupt vector table
  - Safely storing a new value to the interrupt vector table (patching the interrupt vector table)
- In all difficult situations always examine what if scenarios
  - What if a hardware interrupt occurs at different points of our ISR ?
  - Identify the points that need to be protected and protect them with CLI/STI

Servicing a hardware interrupt

- Complete current instruction
- Preserve current context
  - PUSHF Store flags to stack
  - Clear Trap Flag (TF) & Interrupt Flag (IF)
  - Store return address to stack
    PUSH CS, PUSH IP
- Identify Source
  - Read 8259 PIC status register
  - Determine which device (N) triggered the interrupt
- Activate ISR
  - Use N to index vector table
  - Read CS/IP from table
  - Jump to instruction
- Execute ISR
  - usually the handler immediately re-enables the interrupt system (to allow higher priority interrupts to occur) (STI instruction)
  - process the interrupt
- Indicate End-Of-Interrupt (EOI) to 8259 PIC

```
mov  al, 20h
out  20h, al
```

- Return (IRET)
  - POP IP (Far Return)
  - POP CS
  - POPF (Restore Flags)

Interrupt service routines

- Reasons for writing your own ISR's
  - to override the default ISR for internal hardware interrupts (e.g., division by zero need not terminate the program)
  - to chain your own ISR onto the default system ISR for a hardware device, so that both the system's actions and your own will occur on an interrupt (e.g., clock-tick interrupt, measure elapsed time)
  - to service interrupts not supported by the default device drivers (a new hardware device for which you may be writing a driver)
  - to provide communication between a program that terminates and stays resident (TSR) and other application software (maintain your ISRs)

Impact of interrupts on performance

- The frequency of occurrence and the latency of the ISR determine the impact of servicing interrupts to your program
- The latency of the ISR is non-negligible!
- You may not notice it but you may be interrupted several times while executing your program. The good thing is that you don't notice it!
- Always remember:
  - When the processor starts executing an ISR there might be other ISRs executing already
  - Your ISR may be interrupted by a higher-priority interrupt
  - Many devices expect low latency from your ISR (imagine what happens if you hit a key in the keyboard and wait for a minute!)
  - Even those devices with high latencies (e.g. the disk) are not allowed to block other activity in the processor for long

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 25

Bottom line

YOUR INTERRUPT SERVICE ROUTINES MUST BE SHORT AND ACHIEVE THEIR PURPOSE WITH THE MAXIMUM EFFICIENCY! NEVER BLOCK THE SYSTEM WITH YOUR ISRs

Interrupt Service Routines

- ISRs are meant to be short
  - keep the time that interrupts are disabled and the total length of the service routine to an absolute minimum
  - remember after interrupts are re-enabled (STI instruction), interrupts of the same or lower priority remain blocked if the interrupt was received through the 8259A PIC
- ISRs can be interrupted
- ISRs must be in memory
  - Option 1: Redefine interrupt only while your program is running
    - the default ISR will be restored when the executing program terminates
  - Option 2: Use DOS Terminate-and-Stay-Resident (TSR) command to load and leave program code permanently in memory

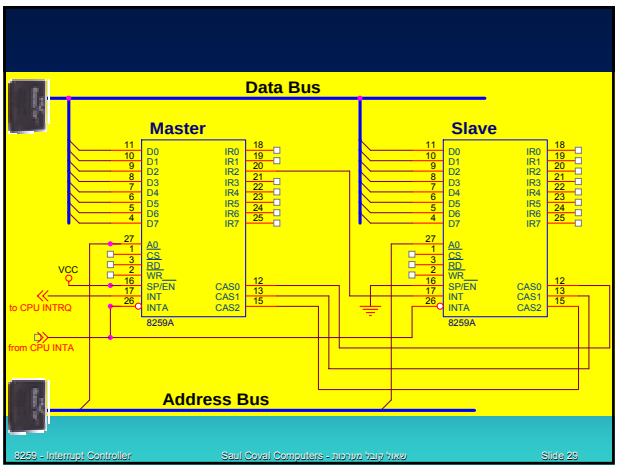
8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 27

Interrupt Driven I/O

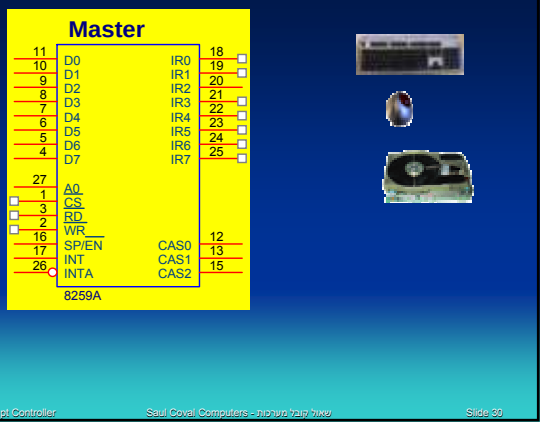
- Consider an I/O operation, where the CPU constantly tests a port (e.g., keyboard) to see if data is available
  - CPU polls the port if it has data available or can accept data
- Polled I/O is inherently inefficient
- Wastes CPU cycles until event occurs
- *Analogy:* Checking your watch every 30 seconds until your popcorn is done, or standing at the door until someone comes by

- Solution is to provide interrupt driven I/O
- Perform regular work until an event occurs
- Process event when it happens, then resume normal activities
- *Analogy:* Alarm clock, doorbell, telephone ring

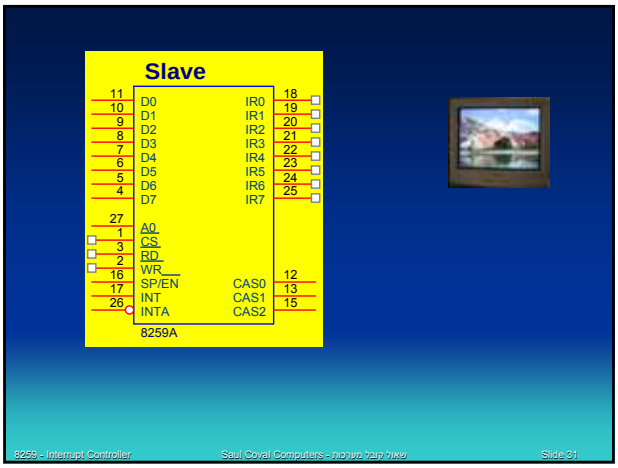
8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 28

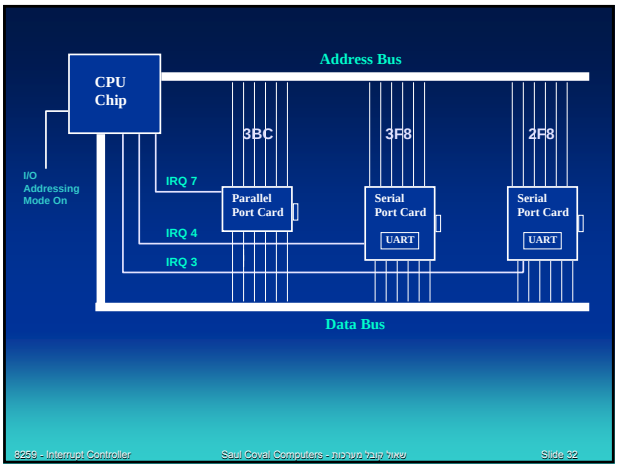


8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 29



8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 30





Interrupts and our everyday lives

- We will spend at least two lectures to explain how to measure and track time in your microprocessor and you will be wondering why don't we just look at our watches…
- But we will also learn that looking at your watch all the time is not a good thing to do, especially if you're a microprocessor…
- We all have priorities
  - E.g. you do your ECE291 homework and your girlfriend/boyfriend calls, there's a high-priority interrupt
  - While you talk to your girlfriend/boyfriend you get another incoming call from your mom, there's an interrupt that you decide how to handle
    - High priority: put the girlfriend/boyfriend on hold
    - Low priority: put your mom on hold or don't even bother to switch to the other line

Interrupts…seriously defined

- Triggers that cause the CPU to perform various tasks on demand
- Three types:
  - Software interrupts – initiated by the INT instruction in your program
  - Hardware interrupts – initiated by peripheral hardware
  - Exceptions – occur in response to error states in the processor or during debugging (trace, breakpoints etc.)
- Regardless of source, they are handled the same
  - Each interrupt has a unique interrupt number from 0 to 255. These are called interrupt vectors.
  - For each interrupt vector, there is an entry in the interrupt vector table.
  - The interrupt vector table is simply a jump table containing

Interrupt vectors

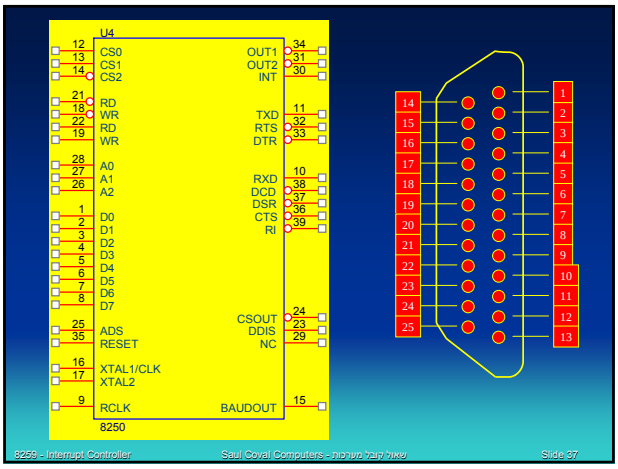
- The first 1024 bytes of memory (addresses 00000 – 003FF) always contain the interrupt vector table. Always. Never anything else.
- Each of the 256 vectors requires four bytes—two for segment, two for offset

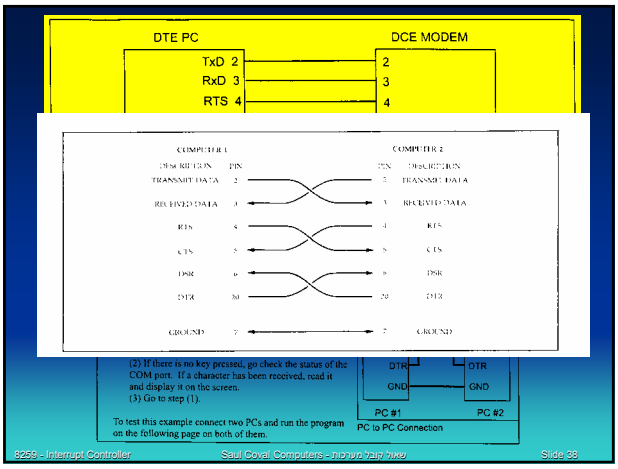
4 * x
00008
00004
00000

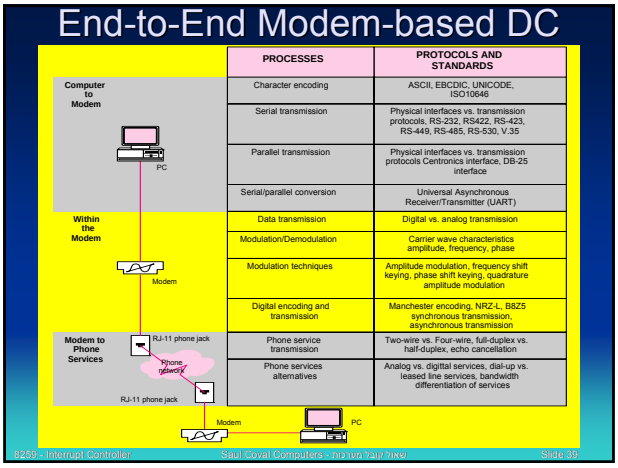
INT x
INT 2
INT 1
INT 0

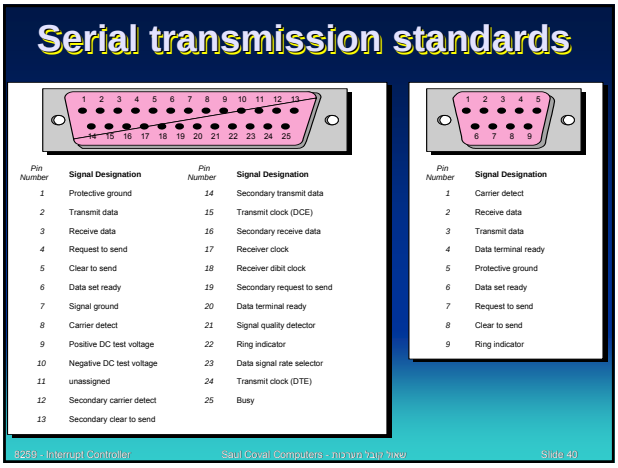
Software interrupts

- Essentially function calls using a different instruction to do the calling and different conventions
- Software interrupts give you access to "built-in" code in the BIOS, the operating system, or peripheral devices
- Software interrupts are triggered with the INT instruction

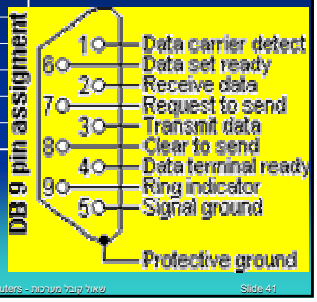


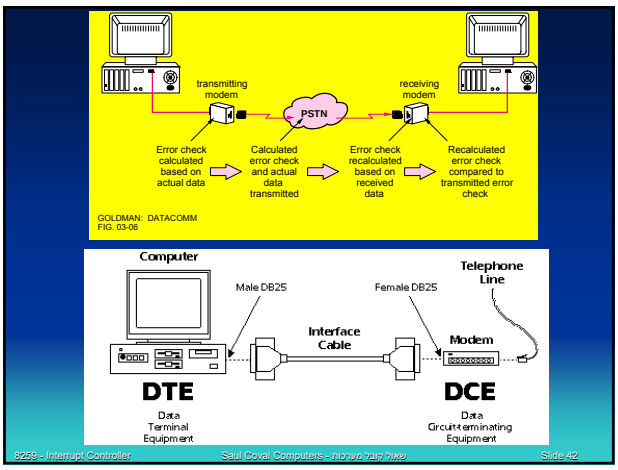






Pin	Signal	Description	I/O
1	CD	Carrier Detect	In
2	RD	Receive Data	In
3	TD	Transmit Data	Out
4	DTR	Data Terminal Ready	Out
5	SG	Signal Ground	-
6	DSR	Data Set Ready	In
7	RTS	Request to Send	Out
8	CTS	Clear to Send	In
9	RI	Ring Indicator	In



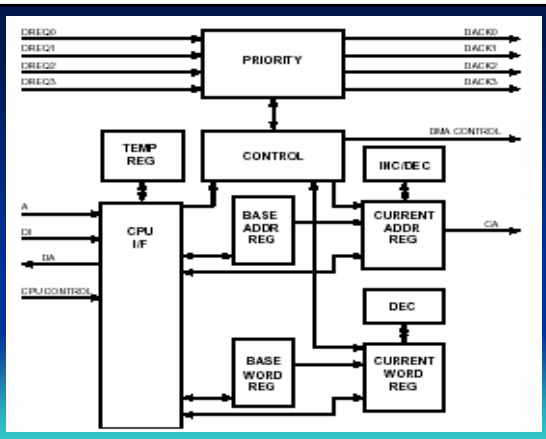


Serial Port Resources

- I/O addresses and IRQ

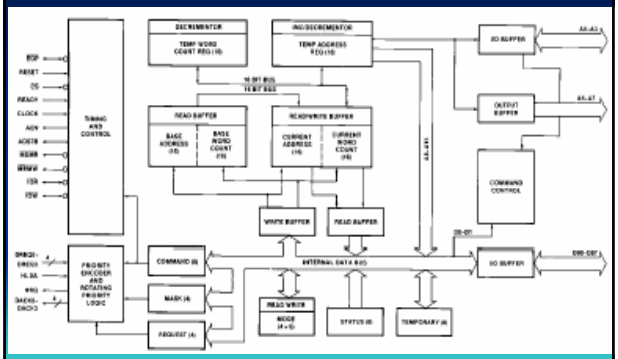
Com Port	I/O Address	IRQ
COM 1	3F8-3FF	4
COM 2	2F8-2FF	3
COM 3	3E8-3EF	4
COM 4	2E8-2EF	3

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 43

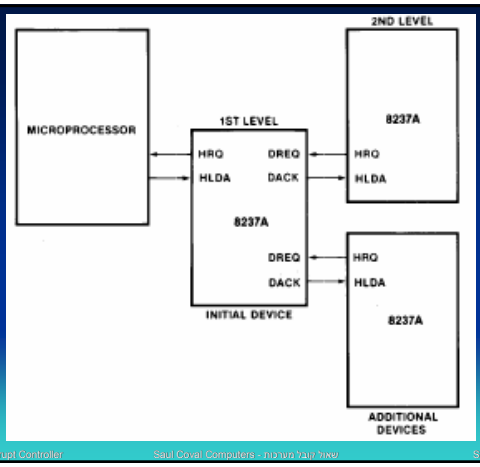


8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 44

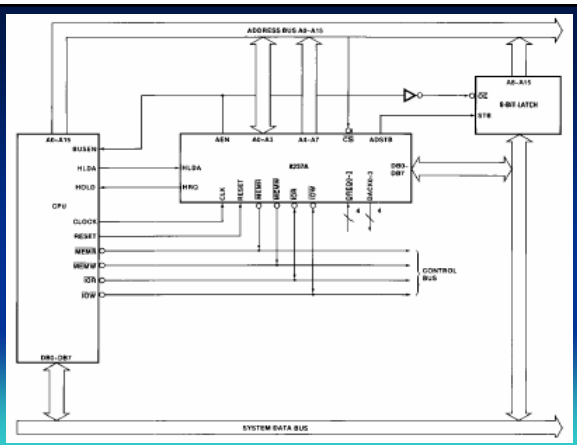
M8237A
4-CHANNEL DMA CONTROLLER



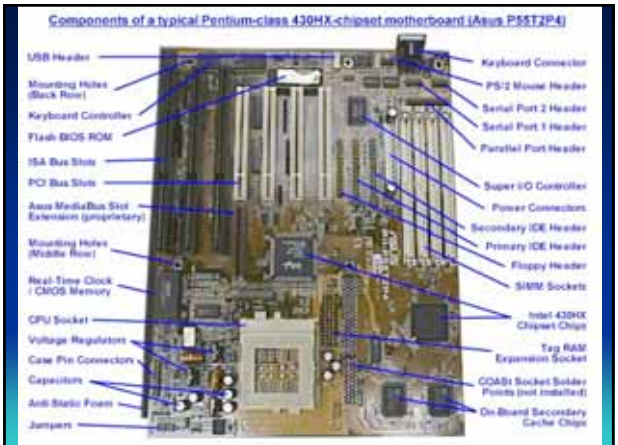
8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 45



8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 46



8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 47

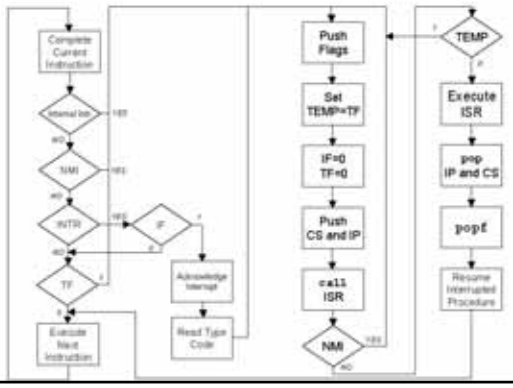


8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 48

Interrupts

- Hardware interrupts
  - Defined by CPU
  - Defined by board-level architecture
- Software interrupts
  - Convenient mechanism to call Operating System
  - defined by software
- Exceptions
  - in response to a condition encountered during execution of an instruction

What Happens During an Interrupt ?



Computer Interrupt

a *signal* indicating that an event needing immediate attention has occurred

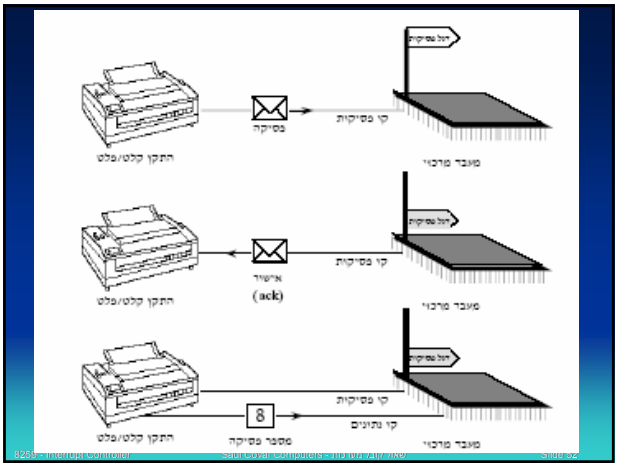
2 Types of Interrupts:

External - generated outside CPU by other hardware

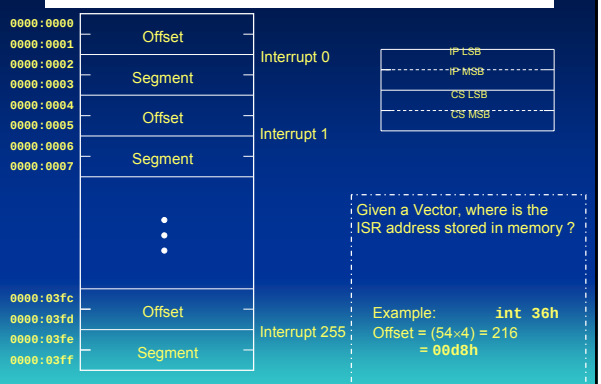
Internal - generated within CPU as a result of an instruction or operation
- x86 has internal interrupts: `int`, `into`, Divide Error and Single Step
- Trap generally means any processor generated interrupt
- in x86, Trap usually means the Single Step interrupt

x86 Interrupts:

1) Hardware Interrupt - External Uses INTR and NMI
2) Software Interrupt - Internal - from `int` or `into`
3) Processor Interrupt - Traps and 10 Software Interrupts (12 total)



IVT Format



Given a Vector, where is the ISR address stored in memory ?

Example: `int 36h`
Offset = (54×4) = 216
= `00d8h`

ISR address

Memory Address (hex)	Interrupt Function Pointer
003FC	INT 255
4 * x	INT x
00008	INT 2
00004	INT 1
00000	INT 0

Interrupt Vector Assignments

Type	Function	Comment
0	Divide Error	Processor - zero or overflow
1	Single Step (DEBUG)	Processor - TF=1
2	Nonmaskable Interrupt Pin	Processor - NMI Signal
3	Breakpoint	Processor - Similar to Sing Step
4	Arithmetic Overflow	Processor - into
5	Print Screen Key	BIOS - Key Depressed
6	Invalid Opcode	Processor - Invalid Opcode
7	Coprocessor Not Present	Processor - no FPU
8	Time Signal	BIOS - From RT Chip (AT - IRQ0)
9	Keyboard Service	BIOS - Gen Service (AT - IRQ1)
A - F	Originally Bus Ops (IBM PC)	BIOS - (AT - IRQ2-7)
10	Video Service Request	BIOS - Accesses Video Driver
11	Equipment Check	BIOS - Diagnostic
12	Memory Size	BIOS - DOS Memory
13	Disk Service Request	BIOS - Accesses Disk Driver
14	Serial Port Service Request	BIOS - Accesses Serial Port Drvr
15	Miscellaneous	BIOS - Cassette, etc.
16	Keyboard Service Request	BIOS - Accesses KB Driver

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 55

Interrupt Vector Assignments (cont)

Type	Function	Comment
17	Parallel Port LPT Service	BIOS - Printer Driver
18	ROM BASIC	BIOS - BASIC Interpreter in ROM
19	Reboot	BIOS - Bootstrap
1A	Clock Service	BIOS - Time of Day from BIOS
1B	Control-Break Handler	BIOS - Keyboard Break
1C	User Timer Service	BIOS - Timer Tick
1D	Pointer to Video Parm Table	BIOS - Video Initialization
1E	Pointer to Disk Parm Table	BIOS - Disk Subsystem Init.
1F	Pointer to Graphics Fonts	BIOS - CGA Graphics Fonts
20	Program Terminate	DOS - Clear Memory, etc.
21	Function Call	DOS - Transfer Control
22	Terminate Address	DOS - program Terminate handler
23	Control-C Handler	DOS - For OS Use
24	Fatal Error Handler	DOS - Critical Error
25	Absolute Disk Read	DOS - Disk Read
26	Absolute Disk Write	DOS - Disk Write
27	Terminate	DOS - TSR Usage
28	Idle Signal	DOS - Idle
2F	Print Spool	DOS - Cassette, etc.
70-77	Hardware Interrupts in AT Bios	DOS - (AT - IRQs 8-15)

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 56

AT – IRQ Definitions

IBM-AT (Advanced Technology) - Intel 80286)

Name	Interrupt Vector	Priority	Description
NMI	02	1	Memory Parity Error
IRQ0	08	2	Timer (Intel 8253 Chip 55 ms intervals)
IRQ1	09	3	Keyboard
IRQ2	0A	4	8259 PIC Slave or EGA/VGA Vert. Retrace
IRQ3	0B	13	Serial Port (COM2 or COM4)
IRQ4	0C	14	Serial Port (COM1 or COM3)
IRQ5	0D	15	Fixed Disk or LPT2 Request
IRQ6	0E	16	Floppy Disk Driver
IRQ7	0F	17	LPT1 Request
IRQ8	70	5	CMOS Real-Time Clock (RT Chip)
IRQ9	71	6	Re-directed to IRQ2
IRQ10	72	7	RESERVED
IRQ11	73	8	RESERVED
IRQ12	74	9	Mouse or other
IRQ13	75	10	Math Coprocessor (NPX)
IRQ14	76	11	Hard Disk
IRQ15	77	12	RESERVED

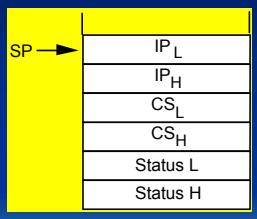
8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 57

Software Interrupts

- BIOS Calls
  - Keyboard -- INT 16H
  - Video I/O -- INT 10H
  - Serial I/O -- INT 14H
  - Printer I/O -- INT 17H
  - Time of day -- INT 1AH
  - Timer tick -- INT 1CH
- DOS Calls
  - Function request -- INT 21H

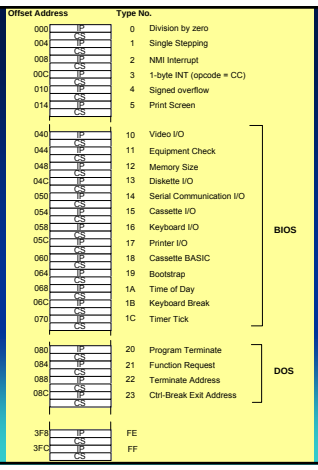
8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 58

Interrupt Processing



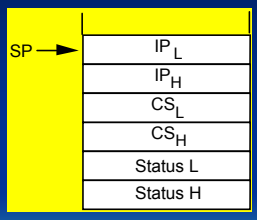
8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 59

Interrupt Vector Table



8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 60

Interrupt service routines end with an IRET instruction that pops the following values off the stack.



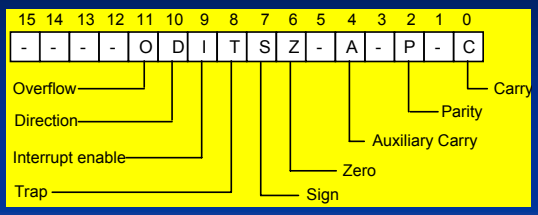
8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 61

Interrupt Vector Table

Offset address		Type number	
000	IP	0	Division by zero
	CS		
004	IP	1	Single Stepping
	CS		
008	IP	2	NMI Interrupt
	CS		
00C	IP	3	1-Byte INT (opcode = CC)
	CS		
010	IP	4	Signed overflow (INTO)
	CS		
014	IP	5	Print screen
	CS		

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 62

The Status Register



8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 63

Time of Day Interrupt: INT 1AH

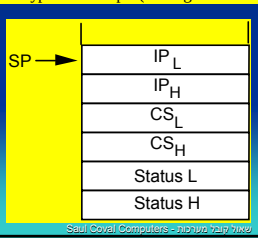
AH = 0	**Read the 32-bit counter clock setting** **Output: CX = high word of count** **DX = low word of count** **AL = 0 if count has not exceeded 24 hours since last read** **= 1 if 24 hours has passed**
AH = 1	**Set the 32-bit counter clock** **Input: CX = high word of count** **DX = low word of count**

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 64

Hardware Interrupts

Nonmaskable Interrupt, NMI

1) the status register, code segment register (*CS*), and instruction pointer (*IP*) are pushed on the stack (see Figure 14.1).
2) the program jumps to CS_2:IP_2 where the instruction pointer IP_2 is stored in locations 0000:0008 – 0000:0009 and the code segment CS_2 is stored in locations 0000:000A – 0000:000B. Thus a non-maskable interrupt is a Type 2 interrupt. (see Figure 11.17 in Chapter 11).



8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 65

Table 14.1
PC Hardware Interrupts

Interrupt Type No.	Interrupt Vector Hex Offset Address	Name
8	20	Timer Tick
9	24	Keyboard
A	28	Unused
B	2C	Reserved for COM2 Serial I/O
C	30	Reserved for COM1 Serial I/O
D	34	Unused
E	38	Disk I/O
F	3C	Reserved for Printer

8259 - Interrupt Controller Saul Coval Computers - שאול קובל מערכות Slide 66

Interrupt Vector Table

RESET

- Set Status, IP, DS, SS, and ES to 0000H
- Set CS to FFFFh
- Execution begins at FFFF:0000 = FFFF0
- (on Pentium, execution begins in *real mode* at FFFFFFF0)

Table 14.1
PC Hardware Interrupts

Interrupt Type No.	Interrupt Vector Hex Offset Address	Name
8	20	Timer Tick
9	24	Keyboard
A	28	Unused
B	2C	Reserved for COM2 Serial I/O
C	30	Reserved for COM1 Serial I/O
D	34	Unused
E	38	Disk I/O
F	3C	Reserved for Printer

Any Questions

Any Questions

Any Questions