# CS253 Architectures II

Lecture 9

Digital I/O

Charles Markham

## Input/Output

The 8086 uses a 20 bit bus to read and write to memory. In addition the 8086 allows for a 16 input/output address space. This is realised by addition of a control line on the CPU that selects memory or input/output read/write.

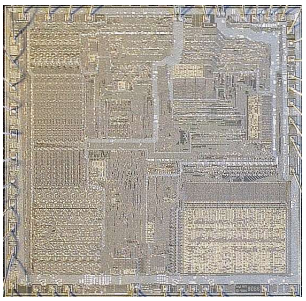
Locations in I/O space are called ports.

There are far fewer instructions to control I/O ports.

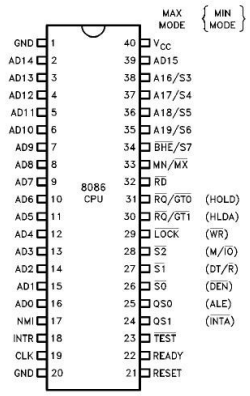
The 8086 has 16 bit I/O address space that can be mapped as 64K of 8 bit ports (Byte) or 32K of 16 ports (Word).

Some hardware (such as the screen) are mapped into memory rather than I/O space.

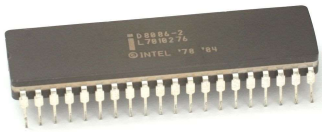
# Different views of the 8086



8086 die




8086 pinouts

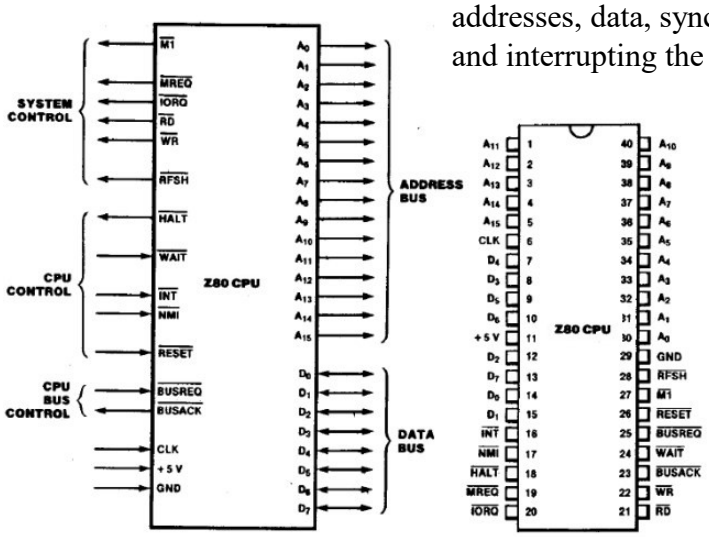


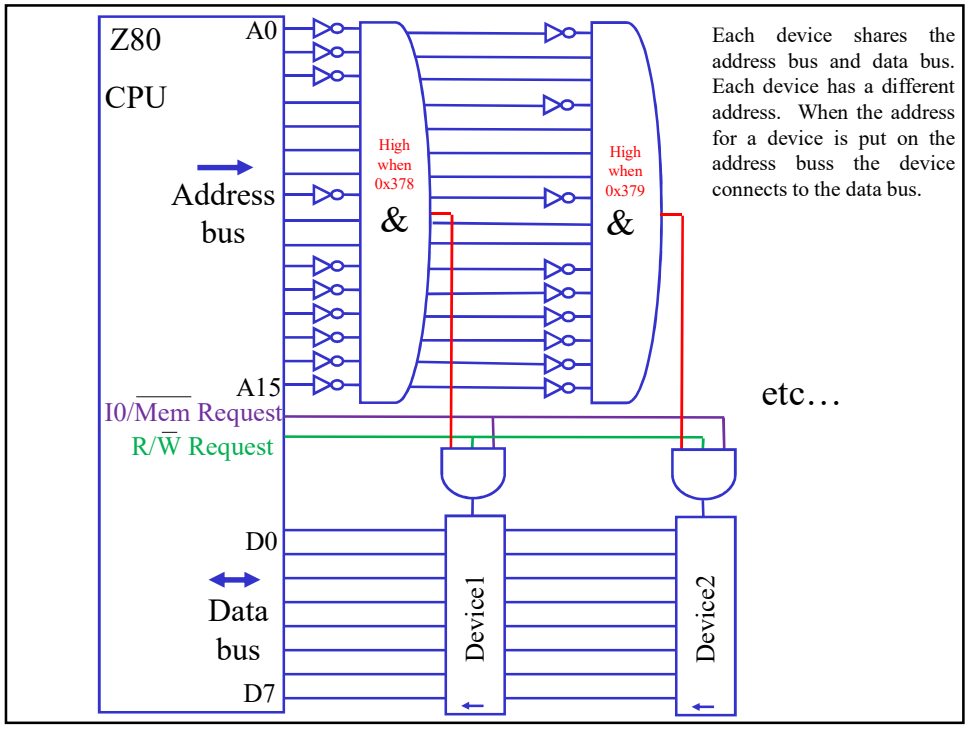
8086 chip

The 8086 is slightly more complicated as it uses a multiplexed address and data buses (the lines are shared), to make this work requires a four-clock memory access cycle which is slightly less efficient than separate data and address lines.

# The Z80A CPU

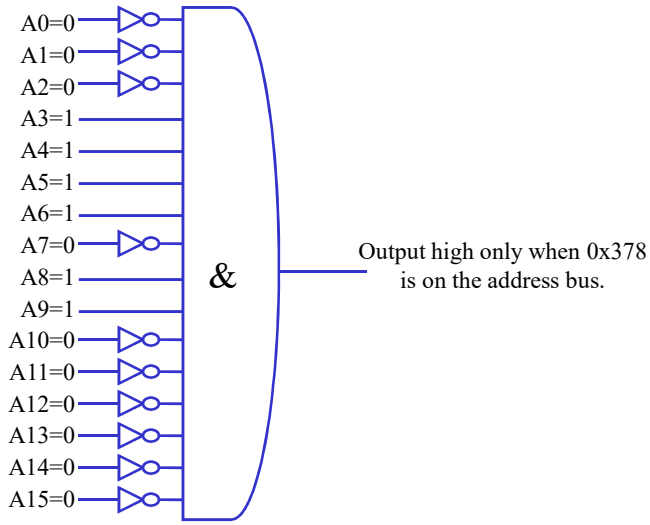
Diagram shows pinouts for a small CPU. The pinouts include connections for power, addresses, data, synchronising and interrupting the CPU.





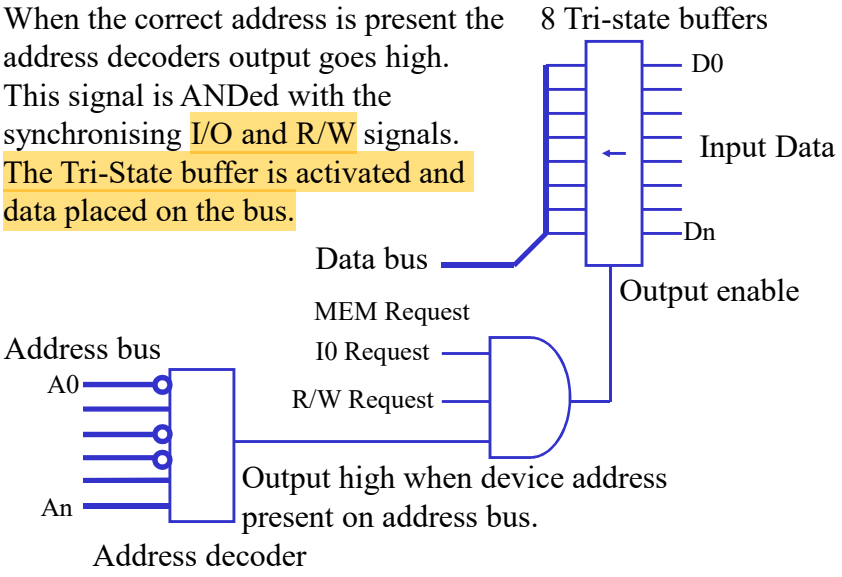

# Address decoder for port 0x378

0x378=(Binary) 0000,0011,0111,1000



# Input Ports

When the correct address is present the address decoders output goes high. This signal is ANDed with the synchronising I/O and R/W signals. The Tri-State buffer is activated and data placed on the bus.




# IN instruction

IN acc, port — Reads a byte,word or double word from the port to the accumulator. In immediate mode you can only address 256 ports, if you use dx to specify the port 64K ports are available.

Example: IN AL,300h

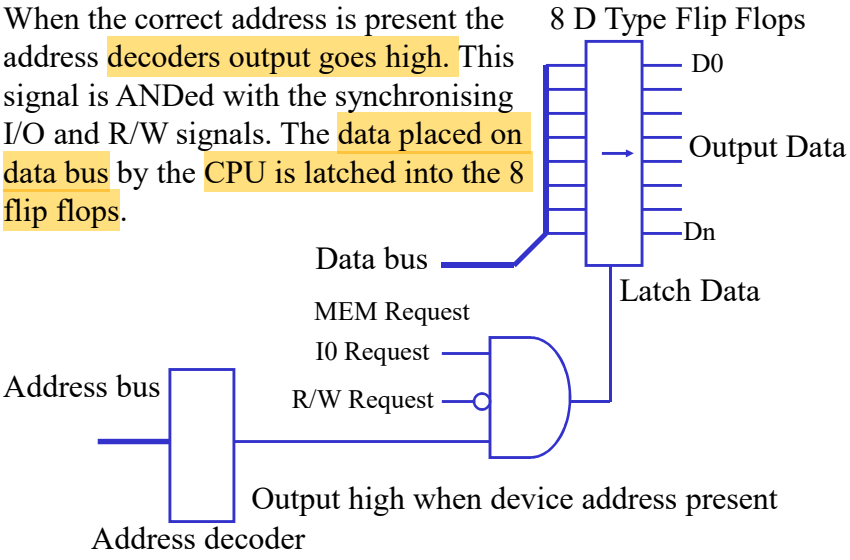
IN AX,DX

Note: The 8086 has an EFLAGS register that set the modes that processor is running in. Two bits in this register define the I/O privilege level, IOPL. The programs current privilege level CPL must be greater than IOPL for the instruction to execute. Solution is to IOPL=2

There is also a I/O permission bit map 8192bytes one bit per port..

# Output Ports

When the correct address is present the address decoders output goes high. This signal is ANDed with the synchronising I/O and R/W signals. The data placed on data bus by the CPU is latched into the 8 flip flops.




# OUT instruction

OUT acc, port — Writes a byte,word or double word from the port to the accumulator. In immediate mode you can only address 256 ports, if you use dx to specify the port 64K ports are available.

Example: OUT AL,300h

OUT AX,DX

Note: Again on 8086 access is restricted by IOPL and I/O permission bitmap.

# The interrupt controller

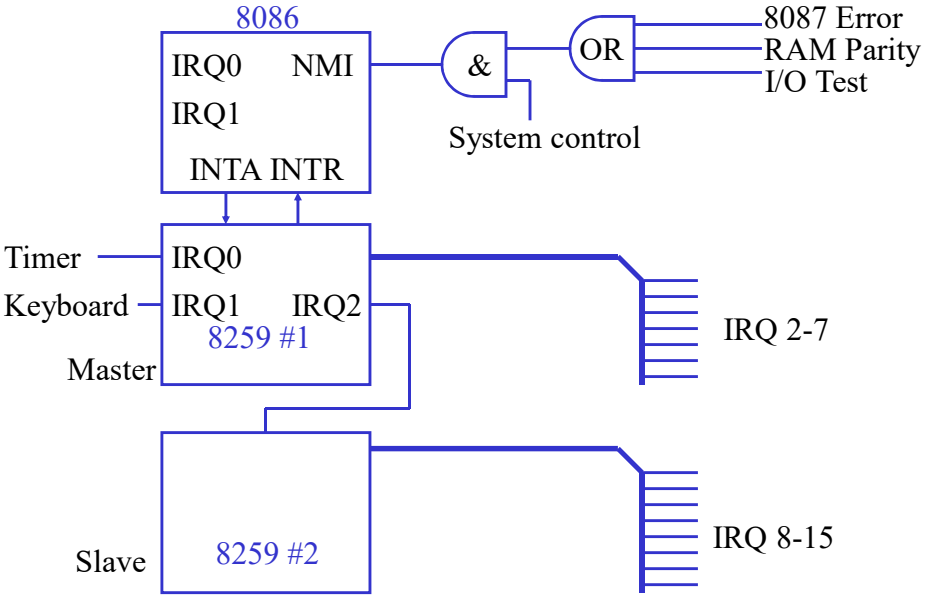
Many CPUs have a number of associated I/O chips that integrate all the logic required to realise a single device in one chip. The following are available for the 8086 cpu.

Intel 8237: direct memory access (DMA) controller
Intel 8251: universal synchronous/asynchronous receiver/transmitter at 19.2 kbit/s
Intel 8253: programmable interval timer, 3x 16-bit max 10 MHz
Intel 8255: programmable peripheral interface, 3x 8-bit I/O pins used for printer connection etc.
Intel 8259: programmable interrupt controller
Intel 8279: keyboard/display controller, scans a keyboard matrix and display matrix like 7-seg
Intel 8282/8283: 8-bit latch
Intel 8284: clock generator
Intel 8286/8287: bidirectional 8-bit driver.
Intel 8288: bus controller
Intel 8289: bus arbiter
Intel 8272A: floppy controller

The 8255 provides the circuitry for one eight bit input/output port (see previous slides).

The 8259 provides the hardware required to interface with the 8086 interrupts.

# Two lines, 15 Hardware Interrupts



# 8259A

The 8259A is a Priority Interrupt Controller, when an IRQ line is pulled low on the device, the interrupt number is placed on the PC bus. The interrupt vector table is looked up and execution jumps to the vector address.

Note: Interrupts 0-7 are not governed by the 8259.

When IRQ0 is brought low the vector table entry 8 is used for the jump. (IRQ1 table entry 9 etc). Entries offset by 20H.

# Typical IRQ designations

| IRQ | Description |
|---|---|
| 0 | Timer Tick |
| 1 | Keyboard |
| 2 | Second 8259A |
| 3 | COM2 |
| 4 | COM1 |
| 5 | LPT2 |
| 6 | Floppy Disk |
| 7 | Sound card |
| 8 | Clock |
| 9 | Redirected IRQ2 |
| 10 | Available |
| 11 | Available |
| 13 | Coprocessor |
| 14 | Hard Disk |
| 15 | Available |

## Some Well Known Port Addresses

Parallel Printer Latch: 0x378 normally output, LPT1 (LPT2 0x278)

Printer Control Latch: 0x37A normally output

Printer Status: 0x379 normally inputs

VDU Colour register: 0x3D9 Sets border colour in DOS text

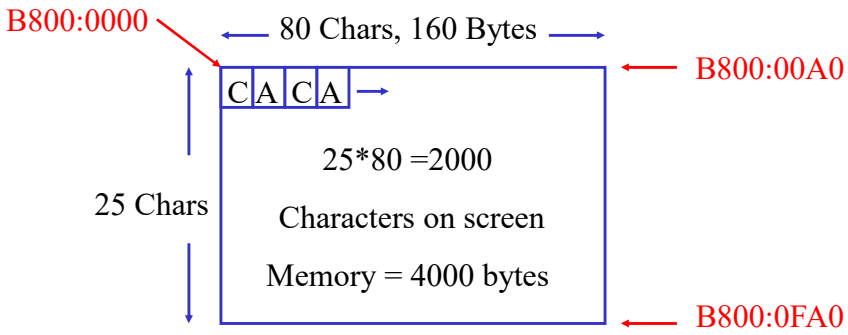
The Text Screen is memory mapped such that top left char is located in memory address B8000, the screen is 80 characters wide.

## Direct Access To Screen Memory

```
0017  B8 B800      mov   ax,0b800h    ; Segment base for screen
001A  1E           push  ds           ; Save current value of DS
001B  8E D8        mov   ds,ax        ; or try ES instead

001D  BB 0168      mov   bx,280       ; 1.5 lines of text
0020  B0 41        mov   al,'A'
0022  88 07        mov   [bx],al      ; Print an 'A'
0024  1F           pop   ds           ; restore the segment
```

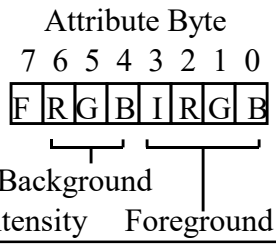
The above example writes directly to the screen memory, no interrupt service routine is called. It is very fast but hardware specific.

# Direct write to screen memory




Odd addresses contain the ASCII code (C).

Even addresses contain the attribute byte (A).




e.g. Attribute=12, Bright red

F:Flash, I:Intensity

# Time Slice (Polling) vs Pre-emptive (Interrupts)

Time Slice Scheduling: The operating system determines when to switch from one task to the next. In the simplest configurations time-slicing is used where each task is given a certain percentage of CPU time. In a round-robin implementation each task would be visited every 20mS. This makes it look to the user like all tasks a running simultaneously. The CPU is polling each piece of hardware.

In preemptive priority based scheduling a low priority task can be interrupted by a higher priority task. This allows the most important task to be done first. This is the method used by the 8259 in the PC. Hardware with a high priority can interrupt code running with a lower priority.

# Interrupts

Most advanced microprocessors allow normal linear program operation to be interrupted by some external signal or by a special instruction in the program.

When Interrupted the microprocessor,

- suspends execution of the main program,
- calls a procedure which services the interrupt (*interrupt handler* or *interrupt service routine*),
- returns control to the mainline program.

# Interrupts

The three types of Interrupt 8086

Hardware Interrupts

An external signal applied to the INTR (interrupt) pin or NMI (interrupt) pin of the processor causes an interrupt.

Exception Interrupts

Triggered by internal errors such as divide by zero, normally to print an error message.

Software Interrupts

Caused by the execution of an assembly language INT command. Simply a call to a function in BIOS.

# Interrupt Vectoring

The Interrupt Vector Table is located in RAM at 0000:0000 to 0000:03FF, i.e. the first 1024 bytes of RAM.

Each interrupt vector is described by 4 bytes, two for the Code Segment (CS) and two for the Instruction Pointer (IP).

A total of 256 Interrupt Vectors can be stored in the table.

Each Interrupt has a specific function.

Modern CPUs can position the vector table in other locations (see lab work)

# Part of The Interrupt Vector Table

| Int | Address | Contents | Function |
|---|---|---|---|
| 0 | 00-03 | 1B02:2389 | Divide by zero |
| 1 | 04-07 | 0070:06F4 | Single Step |
| 2 | 08-0B | 193D:0016 | NMI |
| 3 | 0C-0F | 0070:06F4 | Breakpoint |
| 4 | 10-13 | 0070:06F4 | Overflow |
| 5 | 14-17 | F000:FF54 | Print Screen |
| 6 | 18-1B | F000:DCF4 | Reserved |
| 7 | 1C-1F | F000:DCF4 | Reserved |
| 8 | 20-23 | 2810:1875 | Timer Tick |

Vector Address = 4*Interrupt Number

# Interrupt Processing Sequence

Hardware requests service routine by bringing IRQx low.

The 8259A signals to CPU via INTR line.

The CPU responds with Interrupt Acknowledge signal (INTA).

The Acknowledge signal causes the 8259A to place the interrupt number (0-255) on the data bus.

CPU reads the data.

CPU hardware related registers onto the stack, including current CS:IP,SP and Flags.

CPU multiplies Interrupt No.(Type) by 4 and looks up the jump address in the vector table. CS and IP are changed.

Execution of the interrupt service routine continues until IRET.

IRET registers to be restored to their former values.

CS and IP are restored and the main program continues.

# 8259A Initialisation

In practice the 8259A are already initialised at time of booting. On the PC they have been configured as follows.

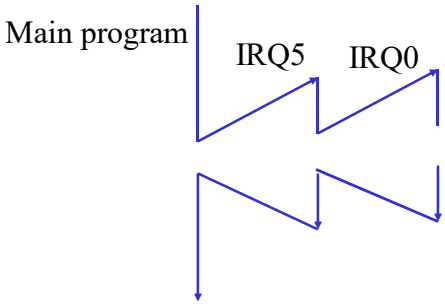
Base address for 8259A #1 is 20H, base address for 8259A #2 is A0H.

8259A#1 IRQ0=INT 08H, IRQ1=INT 09H,…,IRQ7=INT 0FH

8259A#2 IRQ8=INT 70H, IRQ9=INT 71H,…,IRQ15=INT 77H

# Interrupt Priority

Interrupt service routines can be nested. ISR with lower numbers have higher priority.




# 8259 Operational Command Words

Setting the interrupt mask register bit disables the interrupt.

The interrupt mask register is located at 21H and A1H.

Each bit of the 8 bit register corresponds to a specific IRQ line.

| | MSB | | | | | | | LSB | |
|---|---|---|---|---|---|---|---|---|---|
| 8259 #1, 21H | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | Disabled 7, 3, 1 |
| | IRQ7 | | | | | | | IRQ0 | |

```
MOV   AL,137
MOV   DX,021H
OUT   DX,AL
```

# Reset after interrupt

When an interrupt is called the controller notes that it is active and its priority. At the end of the interrupt you must tell the controller that the interrupt is completed. This is done by including a non-specific end of interrupt command at the end of the interrupt service routine before IRET.

Note IRET tells the microprocessor that the interrupt is finished.

```
EOI:    MOV   AL,020H
        MOV   DX,020H
        OUT   DX,AL
```

# Re-vectoring Interrupts

The timer interrupt 8, IRQ0 is activated every 50mS by hardware. This interrupt can be redirect to a different service routine of our own making.

WARNINGS:

If the service routine takes longer than 50mS to run then a second call to the interrupt will occur whilst the first is occuring. The isr should be quick and contain no delays.

To create a delay, just count the number of times the interrupt has been called. (e.g. 5 seconds = 100 interrupt calls)

When the desired number of interrupts have occurred then the isr will run a subroutine to execute the desired function.

Don't call other interrupt routines from an isr, direct write to memory.

# TSR's

Terminate and stay resident programs have very many uses. They were originally used in DOS based applications to provide some element of multitasking. Using TSR's it is possible to create pop up programs such as Sidekick or Spell checkers that run in the background of existing software. Since they remain resident they should be used with care. After developing a TSR do not do anything critical until you reboot your machine. Routines in TSRs are normally invoked by interrupts.

Operating systems such as Win 95 do things a little differently, they have an event manager that notifies each application when a relevant event has occurred. The relevant application then services the event. The event manager is different to an interrupt in that the request to service an event is put on a queue.

# CS253 Architectures II

Lecture 10

MASM Code for Interrupts

Charles Markham

# Start of a TSR

```
.MODEL   small
.STACK
.DATA          ; DS will may change when TSR operates
.CODE

.STARTUP
jmp ov1

dseg           db    '0',0     ; Store variables in code segment
cntr           db    0,0       ; Code segment remains resident
vtabip         dw    0         ; Vector table, IP
vtabcs         dw    0         ; Vector table, CS

ov1:
```

Jump over data stored in code segment.

These 2 lines are best placed later in the code so as to form part of jump.

# Save the existing vector address

4*8=32 =020h

```
push   es              ; Save existing value of ES.
mov    ax,0000h        ; Set ES (extra segment) to page 0
mov    es,ax           ; Note page 0 is vector address table
mov    bx,020h         ; Vector address for timer INT 8, IRQ0
mov    ax,es:[bx]      ; ax=[0000:0020], CS to jump to
mov    vtabip,ax       ; Store the CS, 1st two bytes of table
inc    bx
inc    bx
mov    ax,es:[bx]      ; ax=[0000:0020], IP to jump to
mov    vtabcs,ax       ; Store the IP, 2nd two bytes
pop    es              ; Restore the value of es
```

The vector table stores the address of the service routine for the timer. A IRQ0 causes the program to jump to the address stored in the vector table.

## Set vector address to your service routine

```
push    es              ; Change vector address to ISR routine
mov     ax,0000h        ; Vector address table segment
mov     es,ax
mov     bx,020h
mov     ax,OFFSET isr   ; Set IP of vector table to new ISR
mov     es:[bx],ax
inc     bx
inc     bx
mov     ax,cs           ; The ISR is in the same segment as the program.
mov     es:[bx],ax      ; New CS in vector table is the code segment of
pop     es              ; this program.
```

At this point the timer is no longer running the normal service routine, instead it jumps to the isr in our program. Note isr is just a label, nothing special. We still need the timer routine to work.

## Do something for a while and then restore the vector table.

```
call delay              ; Kill time for a while, isr will occur a few
                        ; times during this delay.
push es
mov ax,0000h
mov es,ax
mov bx,020h
mov ax,vtabip           ; Put the old vector table values back
mov es:[bx],ax
inc bx
inc bx
mov ax,vtabcs
mov es:[bx],ax
pop es
```

Note: With a TSR the code on this page is not included. If this is done then the interrupt will call the code contained in the TSR.

# Now just TSR

```
TSR     mov     ah,031h         ;TSR Entry code ah=31h,al=0h
        mov     al,00h
        mov     dx,1024         ;DX=no of bytes to remain resident
        int     021h            ;Terminate and stay resident.

        jmp     quit            ; Historic code, never reached

code:   Code for delay here
        …
isr:    Code for ISR here
        …

quit:   .EXIT                   ; Historic code, never reached
```

# A long delay (about 3 seconds)

```
delay:  push    cx              ; Long delay
        push    bx              ; Save all variable used

        mov     bx,25000        ; 500,000,000 loops in a few seconds !
dly1:   mov     cx,20000
dly2:   loop    dly2
        dec     bx
        jnz     dly1

        pop     bx              ; Restore variables used
        pop     cx
        ret
```

Remember: Do not make this part of isr, stack faults etc

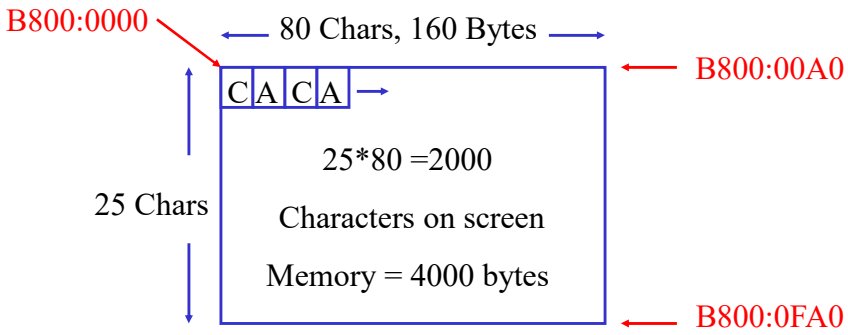
# Interrupt Service Routine

```
isr:    push    ax  ; Start of code you wish to call ever 50mS (18ms PC).
        push    bx  ; Interrupts don't store general registers, you must.
        push    cx
        push    dx
        push    di
        push    si
        push    bp
        push    es
        push    ds
```

# Only do something 1 in 5 interrupt calls, 1/4 Second.

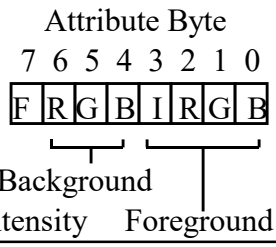
```
        mov     bx,OFFSET cntr
        mov     al,cs:[bx]          ; al=counter value
        inc     al                  ; increase al, al++
        cmp     al,5                ; if al!=5 jump
        jnz     skp
        mov     al,0                ; if al==5 then set al=0
skp:    mov     cs:[bx],al          ; counter value=al
        cmp     al,0
        jnz     ovr                 ; if al!=0 jump over code

        Code to do something ever 1/4 of a second (next slide)
        ….

ovr:    Code to end interrupt service routine (slide after next).
```

# Direct write to screen memory




Odd addresses contain the ASCII code (C).

Even addresses contain the attribute byte (A).




e.g. Attribute=12, Bright red

F:Flash, I:Intensity

# Code run ever 1/4 second

```
        mov   bx,OFFSET dseg
        mov   al,cs:[bx]      ; Load al with character code in [dseg]
        inc   al              ; Increase al by 1, next character.
        cmp   al,':'          ; If al equals character after 9
        jnz   skip
        mov   al,'0'          ; reset al=0

skip:   mov   cs:[bx],al      ; Set dl=[dseg], dseg=0,1,…,8,9,0,….
        mov   dl,al
        mov   ax,0b800h       ; Base address of screen memory
        mov   es,ax
        mov   bx,1680         ; Screen position (10,40)=2*(10*80+40)
        mov   al,dl           ; al=dl
        mov   es:[bx],al      ; Character changed on screen to ASCII al
```

# End of service routine

```
ovr:    mov    dx,020h  ; Non Specific end of interrupt command
        mov    al,020h  ; to sent 8259 #1 controller.
        out    dx,al

        pop    ds       ; Restore all registers previously save on
        pop    es       ; the stack.
        pop    bp
        pop    si
        pop    di
        pop    dx
        pop    cx
        pop    bx
        pop    ax;

        iret            ; Pop CS:IP from stack and return to main
                          normal main line code. Or call Timer
                          routine.
```

```
;JMP FAR VTABIP:VTABCS
        db      0EAh ;jump far
vtabip  dw 0    ; IP offset
vtabcs  dw 0    ; CS segment
```

# Other Interrupts that could be re-vectored.

The keyboard interrupt INT09 can be used to run code after a certain key is pressed.

Note: STI Turn maskable interrupts on, CLI disables interrupts.

These commands are sometimes added to the start and end of an isr to prevent it from being interrupted by higher priority interrupts.