The Keyboard

PC Keyboard : I/O Programmer's Model

- PC keyboard: interrupt driven
  - Cannot run in polled mode: no status port
  - Connected to IR1 of the PIC (details later), through 8255 Parallel Peripheral Interface (PPI)
    - 8255: our programming interface to the keyboard
  - Generates Hardware Interrupt 9

- 2 interrelated 8255 PPI ports:

Data Port (Port PA) :	I/O address 60H
Control Port (Port PB) :	I/O address 61H

PC Keyboard : I/O Programmer's Model

- The keyboard data port (Port A) has dual functionality :
  - Dual = Different values read from the same port!
  - Value read depends on the setting of Port B, Bit 7!
    - Port B, Bit 7 = 0 "Scan Code" read.
      (i.e. identify keystroke)
    - Port B, Bit 7 = 1 "Configuration switch data" is read
- In this course, we never use configuration data, so why don't we set Port B, Bit 7 = 0 and leave it there ?

PC Keyboard : Hardware Requirement

- Keyboard will not send next scan code until previous one “acknowledged”

- To acknowledge scan code:
  - Toggle PB bit 7 0 → 1 and then 1 → 0

- **CAREFUL**! All bits in PB have important values

1. Read Port B : PB_value
2. Force bit 7 = 1: PB_value OR 80H
3. Write modified value back to Port B
4. Write original value (with bit 7 = 0) back to Port B

- NB. The keyboard hardware is initialised when DOS boots

PC Keyboard : Scan Codes

- Scan code: code sent from keyboard whenever keys change state
  - Scan codes are NOT ASCII codes!!
  - The scan codes runs from 0 – 53H
    - e.g. “A” key scan code = 1EH

- Scan codes “make/break coded”
  - one code sent when key pressed (make)
  - different code sent when key released (break)
  - Only difference: most-significant bit
    - If MSBit = 0 → key pressed
    - If MSBit = 1 → key released
  - Example : Letter A
    - Make ‘A’ = 1EH (0001 1110b)
    - Break ‘A’ = 9EH (1001 1110b)

PC Keyboard : Multiple Key Combinations

- Multiple key combinations
  - <SHIFT> ‘A’
  - <CTRL><ALT><DEL>

- Software must manage multiple key combinations.
    - Left Shift key press, make code = 2AH
    - Right Shift key press, make code = 38H
    - Ctrl key press, make code = 1DH
    - Alt key press, make code = 3AH

- Keyboard software must track control keys for correct interpretation
  - Example: letter key pressed while one shift key was down?

    If yes: – how should scan code be interpreted?

Example : A Simple Keyboard Driver

- Requirements
  - prints uppercase char’s representing keys pressed
  - ALT, SHIFT, CTRL keys (and a few others) not managed
  - exit program by resetting
  - ISR ignores key released scan codes
  - uses lookup table to convert key released scan code to uppercase ASCII representation

Example : A Simple Keyboard Driver

- Program architecture
  - Duties divided between main program and keyboard ISR
    - Keyboard ISR gathers data as user enters keystrokes
    - Main prints the keystrokes
  - Data shared in variable `KEYBOARD_CHARACTER`
    - Variable initialised to 0FFh to represent “no data”
      - (0FFh is not an ASCII code for any key)
    - Keyboard ISR puts ASCII code in variable
    - Main program polls variable until valid data found
    - When main reads ASCII code, it must reset variable to “no data” value

How does it know when ?

Keyboard : Code Fragments

```
   LF            EQU    0AH
   CR            EQU    0DH

.data
    KEYBOARD_CHARACTER        DB      0FFH
    SCAN_TABLE     ; lookup table
       DB     0,0,'1234567890-=',8,0
       DB     'QWERTYUIOP[]',CR,0
       DB     'ASDFGHJKL;',0,0,0,0
       DB     'ZXCVBNM,./',0,0,0
       DB     '  ',0,0,0,0,0,0,0,0,0,0,0,0,0
       DB     '789-456+1230'
```

shared variable initialized to “no data” value

Use 0 for keys to ignore

Keyboard : Code Fragments

```
.code
  CLI        ; disable ints while installing ISR
  MOV AX , 0
  MOV ES , AX
  MOV DI , 09H*4
  MOV WORD PTR ES:[DI] , OFFSET  KISR
  MOV WORD PTR ES:[DI+2] , @code

  ; enable keyboard and timer interrupts @ PIC
  IN  AL, 21h
  AND AL , 0FCH
  OUT 21H , AL
  STI        ; let ints happen !
```

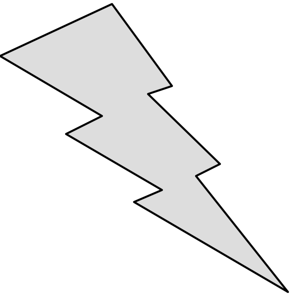
Keyboard ISR : Code Fragments

```
FOR_EVER:                         ; press reset to exit ☺
      CALL   GET_CHAR       ; returns ASCII in AL
      PUSH   AX             ; save char
      CALL   DISPLAY_CHAR   ; displays char in AL
      POP    AX             ; restore char
      CMP    AL , CR        ; check for Enter key
      JNZ    REPEAT_LOOP
      MOV    AL , LF        ; if Enter – do LF too !
      CALL   DISPLAY_CHAR
REPEAT_LOOP:
      JMP    FOR_EVER
```

- Exercise: Modify to exit if a particular char is found.

Keyboard ISR : Code Fragments

```
GET_CHAR PROC NEAR
      ; poll until char received from ISR
      ; check for “no data” value
      CMP KEYBOARD_CHARACTER, 0FFH
      JZ  GET_CHAR
```



Is this a critical region?
Should it be protected?

```
      ; get ASCII character
      MOV    AL , KEYBOARD_CHARACTER
      MOV    KEYBOARD_CHARACTER , 0FFH
      RET
GET_CHAR     ENDP
```

Keyboard : Code Fragments

```
KISR   PROC   FAR
; Standard ISR Setup(Save registers, initialise DS)

   IN AL , 60H         ; get scan code

; Acknowledge Keyboard : Toggle PB bit 7
   PUSH AX             ; save scan code
   IN   AL, 61H        ; read current PB value
   OR   AL, 80H        ; set bit 7
   OUT 61H, AL         ; write value back + bit 7=1
   AND AL, 7FH         ; clear bit 7–back to original
   OUT  61H , AL       ; write original value back
   POP  AX             ; restore scan code
```

Keyboard : Code Fragments

```
       TEST   AL , 80H                    ; ignore break codes
       JNZ    SEND_EOI

   ; Convert make code to ASCII
       LEA    BX , SCAN_TABLE
       XLAT
       CMP    AL , 0                      ; some keys ignored !
       JZ     SEND_EOI

   ; Put ASCII encoded value in shared variable
       MOV    KEYBOARD_CHARACTER , AL

   SEND_EOI:
       MOV    AL , 20H
       OUT    20H , AL
   ; Standard ISR exit code
   IRET
KISR   ENDP
```

The 5 Dedicated Interrupts (0..4)

- **Interrupt 0** (divide error)
  - Invoked by CPU after DIV or IDIV if the calculated quotient is larger than the destination
  - How big is the quotient if an attempt is made to divide by 0?

- **Interrupt 1** (single step)
  - Used by debuggers to support single stepping
  - TF flag set: CPU invokes this ISR after executing most instructions
    - TF cleared as part of INT execution (after flags are pushed)
    - Why is TF cleared ?
      - When ISR starts, processor no longer in single-step mode
      - Avoids an infinite loop!

The 5 Dedicated Interrupt (0..4)

- **Interrupt 2** (non-maskable interrupt)
  - Hardware interrupt which cannot be disabled.

- **Interrupt 3** (breakpoint interrupt)
  - A special version of the INT instruction encoded in one byte: CEH
  - Used to provide breakpoint capabilities for debuggers

- **Interrupt 4** (overflow interrupt) **INTO**
  - OF set when INTO instruction is executed: CPU invokes this ISR
  - Used in numeric libraries to trap overflow errors

- Higher processors (80186, 80286, etc.) have additional dedicated interrupts
  - IBM/Microsoft decided to use interrupts reserved by Intel for their own purposes. Caused problems when AT was released ☹