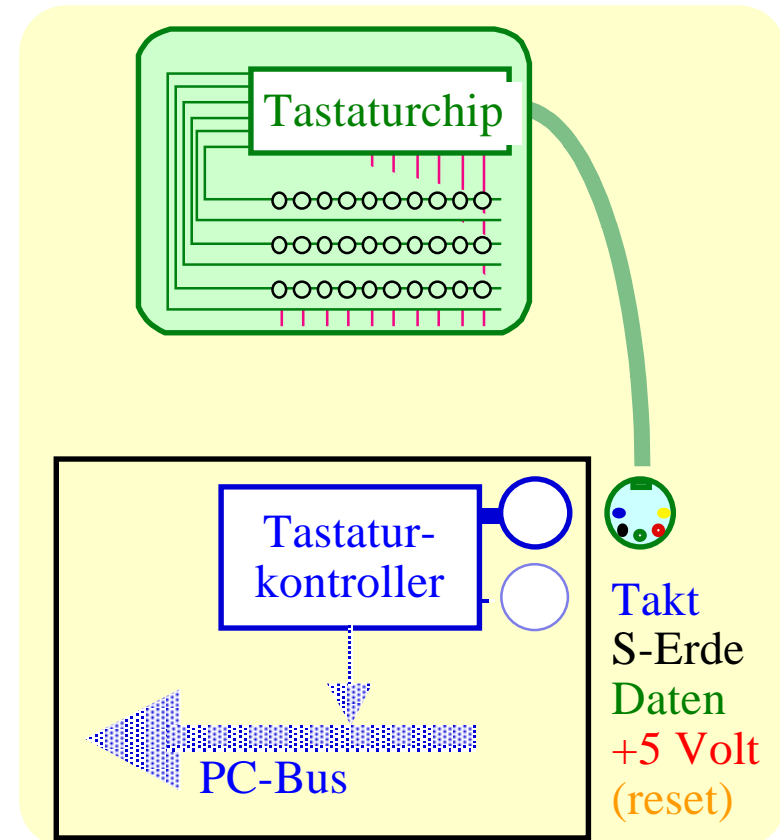


6.1.1 Umsetzung der Scancodes im Treiber

- Tastaturen liefern nur "Scancodes".
- Scancodes werden gemäss lokalem Alphabet in Zeichen übersetzt.
- Dauerumschaltung durch Treiber realisiert:
 - Caps-Lock, Num-Lock.
- Make-Break Arbeitsweise:
 - Make-Code für Tastendruck.
 - Break-Code beim Loslassen (= Make+128).
- Zusatzaufgaben der MF II -Tastatur:
 - Präfix \$E0 für alle zusätzlichen Tasten,
 - jedoch Präfix \$E1 für die "Pause"-Taste,
 - automatisches Num-Lock Präfix für Steuertasten zur Nachbildung der AT-Steuertasten.
- Ursache der Komplexität ist die Kompatibilität mit alten Tastaturen.
- Num-Lock und Präfix nur aus der Historie erklärbar.

6.2 Tastaturanschluss:

- Tastaturcontroller auf der Hauptplatine:
 - Interrupt IRQ#1,
 - 8255 beim PC/XT, später 8042/ 8741/ 8742,
 - Ansteuerung über E/A-Ports \$60 & \$64,
 - Zusatzeingang für PS/2 Maus ab PC/AT.
- Tastaturchip in der Tastatur:
 - Kommunikation mit dem Tastaturcontroller,
 - Abfrage der Tastatur über Scanmatrix,
 - Entprellen der Tasten.
- Scanmatrix:
 - Tasten an den Kreuzungspunkten,
 - "n-Key Rollover" feststellen ...
- 5-poliger DIN-Stecker:
 - fakultative Rücksetzung, Stromversorgung,
 - Schirmungserde, Signalerde,
 - Datenbits, Bittakt.



6.3 Beispiel "TippCode.pas"

- Scancodes mit Interrupt-Routine abholen.
- Break-Codes haben Vorzeichenbit gesetzt.
- Interrupt-Kontroller explizit zurücksetzen.
- IRET ist impliziert durch Schlüsselwort „interrupt“.
- Eingabeport ist \$60.
- Statusport \$64:
 - hier nicht berücksichtigt,
 - keine Fehlercodes,
 - keine PS/2 Maus,
 - ...

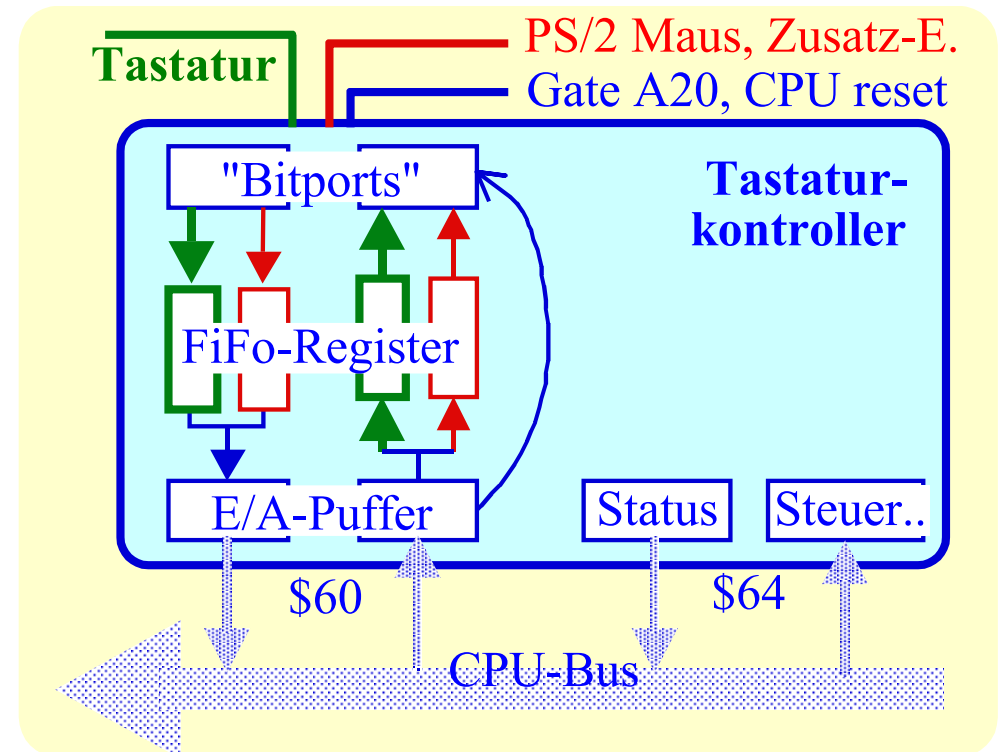
```
program TippCode ;(*$F+ .. grosse Sprünge *)
uses crt, dos;
var oldhandler: procedure; zlr: integer;

procedure MyHandler; interrupt; var taste: integer;
begin taste := port[ $60 ];
  if taste > 127 then taste := 128 - taste;
  port[ $20 ] := $20; (* End of Interrupt *)
  if ( zlr mod 10 ) = 0 then writeln;
  zlr := zlr + 1; write( taste : 5 );
end (* MyHandler *);

begin zlr := 0;
  GetIntVec( 9, addr( oldHandler ) );
  SetIntVec( 9, addr( MyHandler ) );
  repeat until zlr = 99 ;
  SetIntVec( 9, addr( oldHandler ) );
end .
```

6.4 Register im Tastaturkontroller

- Ein- und Ausgabepuffer am Port \$60:
 - Scancode von Tastatur einlesen,
 - Rückmeldecode von Tastatur einlesen,
 - Steuerbefehle an Tastatur schicken (•t).
- Ein-/Ausgabeports mit Bytezugriff:
 - für CPU über Portadressen zugänglich,
 - Ausgabepuffer für Daten zur CPU (\$60).
 - Eingabepuffer für Daten von CPU (\$60).
 - Steuerregister für Kontroller an Port \$64.
 - Statusregister des Kontrollers an Port \$64.
- "Bitports" ohne eigene E/A-Adresse:
 - indirekt über Befehl im Steuerregister bedienen,
 - teilweise mit den Steckern verbunden,
 - Ausgabe-Bitport in Richtung Gerät,
 - Eingabe-Bitport vom Gerät.



- Eingabe-Bitport:

- bitserielle Leitungen zum Chip hin von Tastatur, Zusatzeinheit und Hauptplatine:
- KBLK - Tastatur frei, Tastatur gesperrt (1,0),
- C/M - Monochrom-Display (veraltet),
- Bit#5..2 - reserviert,
- AUXD - Datenbit der Zusatzeinheit (Pegel),
- KBD - Datenbit der Tastatur (Pegel).

- Ausgabe-Bitport zur Tastatur etc.:

- bitserielle Leitungen zum Stecker typisch:
- KBD0 - Tastatur-Datenbit (Bit#7, Ausgabe, 1=Pull Data Low),
- KCLK - Tastaturtakt (1=Pull Clock Low),
- AUXB - Byte von Zusatzeinheit abholen bitte,
- OUTB - Byte von Tastatur abholen bitte,
- ACLK - ZE-Takt (Zusatzeinheit, 1=Pull Clock Low),
- AXD0 - ZE-Datenbit (Ausgabe, 1=Pull Data Low),
- GA20 - Gate für Adressleitung 20,
- SYSR - Prozessorreset.

- Steuerregister am Port[64] schreibbar:

- Bit#0..Bit#7 für Befehle an den Tastatur-Kontroller,

- Statusregister vom Port[64] lesbar:
 - PARE - Paritäts-Error (Bit#7),
 - TIM - Zeitüberschreitung für Antwort,
 - AUXB - Byte von Zusatzeinheit abholen bitte
 - KEYL - Tastatur frei, Tastatur gesperrt (1,0),
 - C/D - Befehlsbyte/ \neg Datenbyte geschrieben,
 - SYSF - Selbsttest ok,
 - INPB - Eingabepuffer beschäftigt, bitte warten,
 - OUTB - Byte von Tastatur abholen bitte.

- Befehle an den Tastatur-Kontroller:
 - werden ins Steuerregister geschrieben (\$64),
 - Befehlsparameter von CPU in Eingabepuffer,
 - z.B. Ausgabebitport, Tastaturausgabepuffer, Zusatzeinheit schreiben,
 - Bits am Eingabeport lesen ...

- Befehle an die Tastatur:
 - regulär über Ausgabepuffer an Port[60],
 - Wiederholrate & Verzögerung (\$F3),
 - Leuchtdioden schalten (\$ED),
 - Tastatur identifizieren (\$F2),
 - Tastatur reset (\$FF),
 - Echo (\$EE) ...

6.4.1 Programmfragment: LEDs bzw. Gate-A20 einschalten:

- Abwarten bis Tastaturbefehl bitweise über die Leitung geschickt wurde.
- LED-Befehl an den Chip in der Tastatur.
- Gate-A20 Befehl an den Kontrollerchip.

```
procedure WaitForKeyboardReady; (* INPB im Statusregister testen *)  
begin while ( port[ $64] and 2 ) <> 0 do (* busy loop *) end;
```

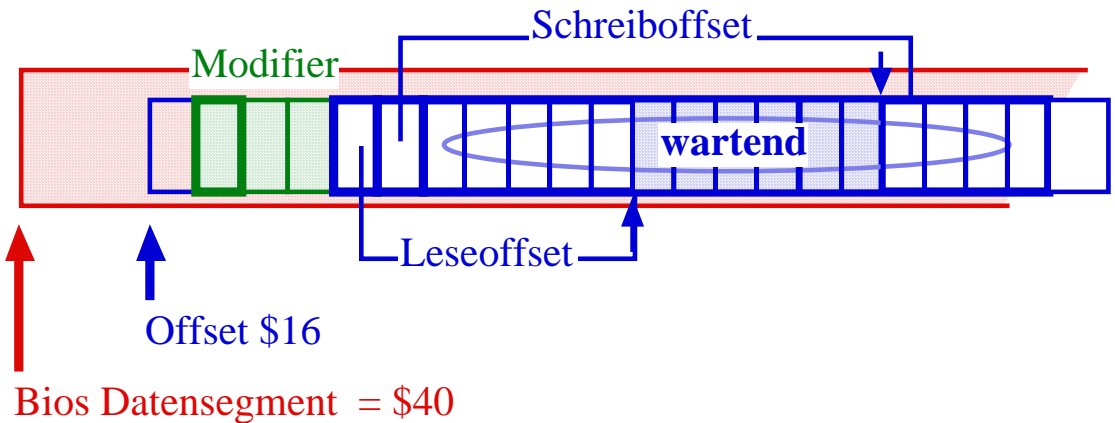
```
procedure SwitchLight( LEDSet: integer );  
begin     WaitForKeyboardReady; port[ $60] := $ed; { KB-Befehl }  
          WaitForKeyboardReady; port[ $60] := LEDSet; { Befehlsparam }  
end;
```

```
procedure enableA20;  
begin     WaitForKeyboardReady; port[$64]:=$D1; { schreibe Aux-Port }  
          WaitForKeyboardReady; port[$60]:=$DE; { enable A20 ... }  
          WaitForKeyboardReady  
end;
```

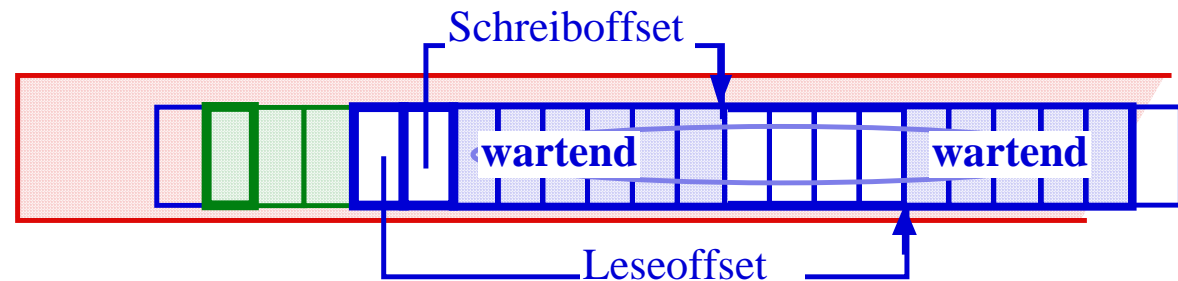
6.5 Tastatureingabe über Bios

6.5.1 Ringpuffer im Speicher

- Interrupt 1 ruft Tastaturtreiber.
- Gelesene Zeichen in den Puffer:
- Pufferorganisation:
 - Puffer für 15 Zeichen (32 Bytes),
 - Modifier enthalten die Umschaltzustände,
 - Scancode und ASCII paarweise gespeichert,
 - Lesezeiger und Schreibzeiger mit Wrap-around.



- Im Ringpuffer ergibt sich eventuell eine "Wrap-Around"-Situation für die Lese-& Schreibzeiger:



6.5.2 Beispiel: KeyBuffer.pas

- Darstellung des Warteschlangenfensters.
- Abholung der Zeichen absichtlich verzögert.
- Ausgabe des Programms, bei schneller Eingabe von 5 'W's:
 - jeweils 3 Zeilen pro Ausgabezyklus,
 - Sternchen zeigen wartende Zeichen an,
 - rechts: Lesezeiger, Schreibzeiger, Modifier#1,
 - jeweils 16 alte und neue Zeichen im Puffer,
 - Scancode von 'W' ist 17,
 - Scancodes der Tasten.

```

                                     *           54 56 32
- - - - -
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12

* *                                     * * * 56 34 96
W W - - - - - W W W
17 17 12 12 12 12 12 12 12 12 12 12 17 17 17
```

*	*													*	*	58	34	96
W	W	-	-	-	-	-	-	-	-	-	-	-	-	W	W	W		
17	17	12	12	12	12	12	12	12	12	12	12	12	12	17	17	17		

*	*														*	60	34	96
W	W	-	-	-	-	-	-	-	-	-	-	-	-	W	W	W		
17	17	12	12	12	12	12	12	12	12	12	12	12	12	17	17	17		

*	*																30	34	96
W	W	-	-	-	-	-	-	-	-	-	-	-	-	W	W	W			
17	17	12	12	12	12	12	12	12	12	12	12	12	12	17	17	17			

	*																	32	34	96
W	W	-	-	-	-	-	-	-	-	-	-	-	-	W	W	W				
17	17	12	12	12	12	12	12	12	12	12	12	12	12	17	17	17				

6.5.3 Quelltext von "Keybuffer.pas":

- Pascal Record über den Speicher legen.
- Struktur ausgeben jeweils bei neuem Zeichen.
- Ein Zeigerwert fingiert die Adresse.

```
program KeyBuffer; uses crt;
type BuffPtr = ^RingBuffer;
      RingBuffer = record ifiller, modif1, modif2, alt: char;
                  rd, wr: integer;
                  code: array[0..31] of char;
                  end;
var zlr : longint; oldWr : integer; key : char;

function wartet( rd, pos, wr: integer): boolean;
begin if wr>rd then wartet:= ( rd<= pos) and ( pos<wr)
      else wartet:= ( rd <= pos) or ( pos<wr); (* wrapping around *)
      if wr=rd then wartet:=false;
end;
procedure ShowBuff( var b: RingBuffer ); var iBuff : integer; ch : char;
```

```

begin with b do begin (* RingPuffer formattiert ausgeben *) writeln;
  for iBuff:=0 to 15 do (* Wartemaske *)
    if wartet( rd, 2*iBuff+30, wr) then write(' *') else write(' ');
  writeln( rd:4, wr:4, ord(modif1):4 );
  for iBuff:=0 to 15 do begin (*ASCII *)
    ch := code[ 2*iBuff ]; if ch<' ' then ch:=' '; (* filter *) write(ch:3);
  end; writeln;
  for iBuff:=0 to 15 do (* Scancode *) write( ord( code[ 2*iBuff+1 ] ):3 );
  key:= ReadKey; (* Rekursion, ~iteration *) writeln;
  if keypressed then ShowBuff( b );
end end;

procedure TestBuff( var b: RingBuffer );
begin if b.wr<>oldWr then ShowBuff( b ); oldWr := b.wr end;

begin repeat delay(10000) (* let it fill *);
  TestBuff( BuffPtr( $400000 +$16 )^);
until key='.';
end.

```

6.5.4 Bios-Aufruf über die DOS-Unit

- DOS-Unit in Turbo-Pascal erlaubt bequem BIOS- & DOS-Fkt. zu rufen.
- BIOS besorgt die Umsetzung der Scancodes auf den lokalen Zeichensatz.
- Register-Record als Hilfskonstruktion (Abbild der echten Register).
- Vollführt großen Umweg durch die Laufzeitumgebung und MS-DOS.
- Schwer mit dem Debugger zu testen.

```
program PascalBios; uses dos, crt;  
var reg : Registers;  
begin    write( 'Pascal Bios - Readkey :' );  
        reg. ah := 0; (* 0 = Zeichen_lesen *)  
        Intr( $16, reg );  
        writeln( ( integer( reg.ah ):4, chr( reg. al ) :3 );  
end.
```

6.5.5 Bios-Aufruf über Inline-Assembler

- Gleiche Bios-Funktion direkter gerufen.
- Immer noch langsamer als über Ports.
- Leicht mit dem Debugger zu testen.

```
program AsmBios;  
var scan: shortint; ascii: char;  
  
begin write('AsmBios - Readkey :');  
    asm    MOV    AH, 0  
          INT     $16  
          MOV    [scan], ah  
          MOV    [ascii], al  
    end ;  
    writeln( scan : 4, ascii : 3);  
end .
```

6.5.6 PS/2-Protokoll (Firmware)

- Takt vom Device.
- Datenleitung zur PS/2 Tastatur oder Maus ist bidirektional.
- Übertragung vom Gerät zum Host:
 - ansteigende Flanke,
 - erst testen ob ~Data,
 - "Request to send",
 - Host hat Vorrang.
- Übertragung vom Host zum Gerät:
 - "Request to send",
 - Takt vom Device.

