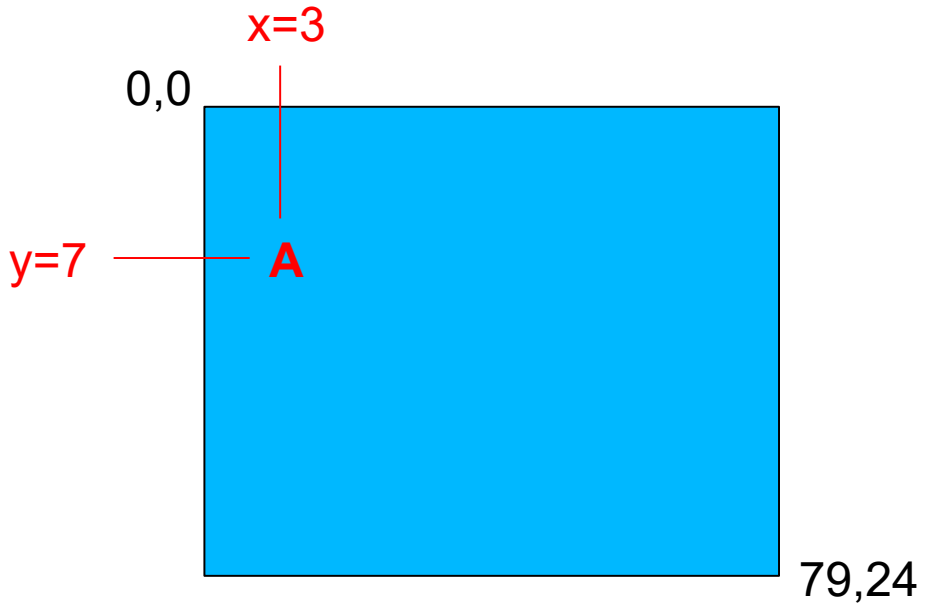
CGA-Programmierung

Ausgabestrom

- `print!` und `println!`
  - Standard-Makros für Ausgaben in Rust
  - Realisiert mithilfe eines Writers in `cga_print.rs`
  - Die Umwandlung von Zahlen nach Hexadezimal- oder Binäreformat erfolgt automatisch in `core::fmt`
  - Die Ausgabe des Zeichenstroms erfolgt mithilfe der Funktion `cga::print_byte(byte)` (an der aktuellen Text-Cursor-Position)

Text-Cursor

- Die Position ist ein 16 Bit Offset zur linken oberen Ecke
- Die Textauflösung ist 80x25 Zeichen
- Cursor-Position im Beispiel = (3,7)
- Offset = 7*80 + 3 = 563



Text-Cursor

- Der Text-Cursor wird in der CGA-Hardware verwaltet
- Er kann über bestimmte Port-Adressen gelesen / geschrieben werden
- Achtung: Port-Adresse != Speicheradressen
  - Unterscheidung durch CPU-Pin: M/IO
  - 16 Adressleitungen für Port-Adressen → 16-Bit
- Zugriff auf Port-Adressen erfolgt über Portbefehle: `in` / `out`
- Sind in der Vorgabe in `cpu.rs` in den Funktionen `inp` und `outb` gekapselt

Portadressen für den Text-Cursor

- Der 16 Bit Offset wird über ein Datenregister geschrieben / gelesen
- Das Datenregister kann aber nur ein Byte schreiben / lesen
- Die Auswahl von Cursor high/low erfolgt über ein Indexregister
- Es muss also erst über das Index-Register mitgeteilt werden, welches Byte high/low über das Datenregister geschrieben werden soll

Port	Register	Zugriffsart
3d4	Indexregister	nur schreiben
3d5	Datenregister	lesen und schreiben

Index	Register	Bedeutung
14	Cursor (high)	Zeichenoffset der Cursorposition
15	Cursor (low)	

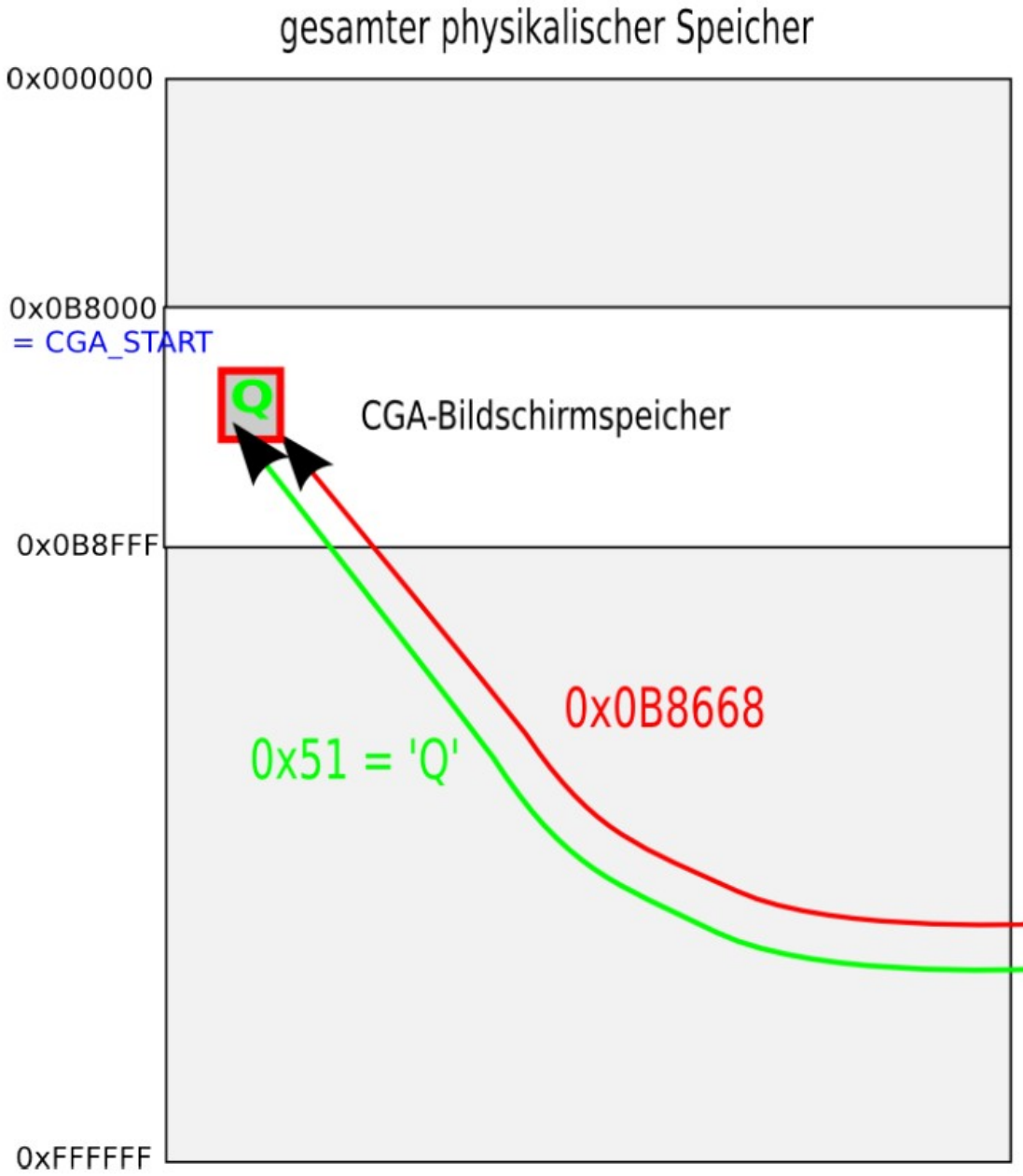
Anzeige von Zeichen

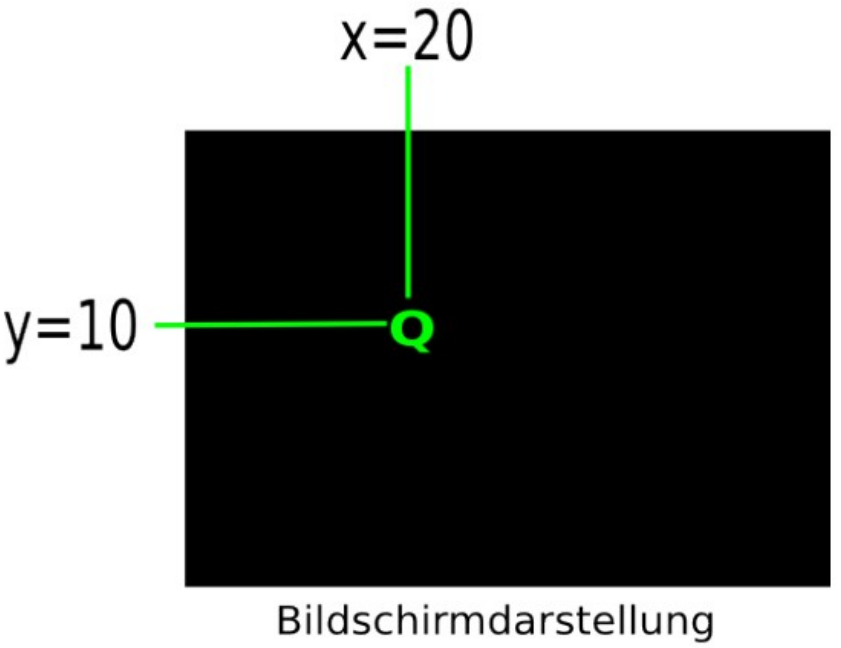
- Erfolgt in `cga.rs` in der Funktion `show(x,y,c,attrib)`
  - Die Ausgabe erfolgt durch schreiben an Speicheradressen (nicht Portadressen)
  - Ab der physikalischen Speicheradresse `0xb8000` ist der CGA-Speicher eingeblendet
  - Beispiel zur Anzeige des Buchstabens Q in der linken oberen Ecke

```
pub const CGA_START: u64 = 0xb8000;

unsafe {
   *(pos as *mut u8) = 'Q' as u8;
}
```

Anzeige von Zeichen






```
let x = 20;
let y = 20;
let pos = CGA_START + 2*(x + y*80);

unsafe {
  *(pos as *mut u8) = 'Q' as u8;
}
```

Anzeige von Zeichen

- Für die Anzeige eines Zeichens werden jeweils zwei Bytes verwendet
- Gerade Adressen: ASCII-Code
- Ungerade Adressen: Attribut-Code

```
pub const CGA_START: u64 = 0xb8000;

// ...

let pos = CGA_START + 2*(x + y*80);

unsafe {
   *(pos as *mut u8) = 'Q' as u8;
   *((pos+1) as *mut u8) = 0xF; // weiss auf schwarz
}
```

Attribut-Byte

- Zu jedem Zeichen können die Merkmale Vordergrundfarbe, Hintergrundfarbe und Blinken einzeln festgelegt werden.
- Für diese Attribute steht pro Zeichen ein Byte zur Verfügung

Darstellungsattribute	
Bits 0-3	Vordergrundfarbe
Bits 4-6	Hintergrundfarbe
Bit 7	Blinken

- Das Blinken funktioniert nur auf echter Hardware

Attribut-Byte

- Im CGA-Textmodus stehen die folgenden 16 Farben zur Verfügung:

Farbpalette			
0	Schwarz	8	Dunkelgrau
1	Blau	9	Hellblau
2	Grün	10	Hellgrün
3	Cyan	11	Hellcyan
4	Rot	12	Hellrot
5	Magenta	13	Hellmagenta
6	Braun	14	Gelb
7	Hellgrau	15	Weiß

- Da für die Hintergrundfarbe im Attributbyte nur drei Bits zur Verfügung stehen, können auch nur die ersten acht Farben zur Hintergrundfarbe gewählt werden (gilt nur für echte Hardware; In QEMU gibt es 16 Hintergrundfarben, dafür kein blinken).

Weiterführende Informationen

- Wer mehr zum Thema VGA-Grafikkarten-Programmierung lesen möchte, sei auf das FreeVGA-Projekt verwiesen:

  http://www.osdever.net/FreeVGA/home.htm

- Ist nicht notwendig für unsere Aufgabe!