Aufgabe 2 - Erweiterung um PIC und Interrupts

Rainer Müller

Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2014/2015

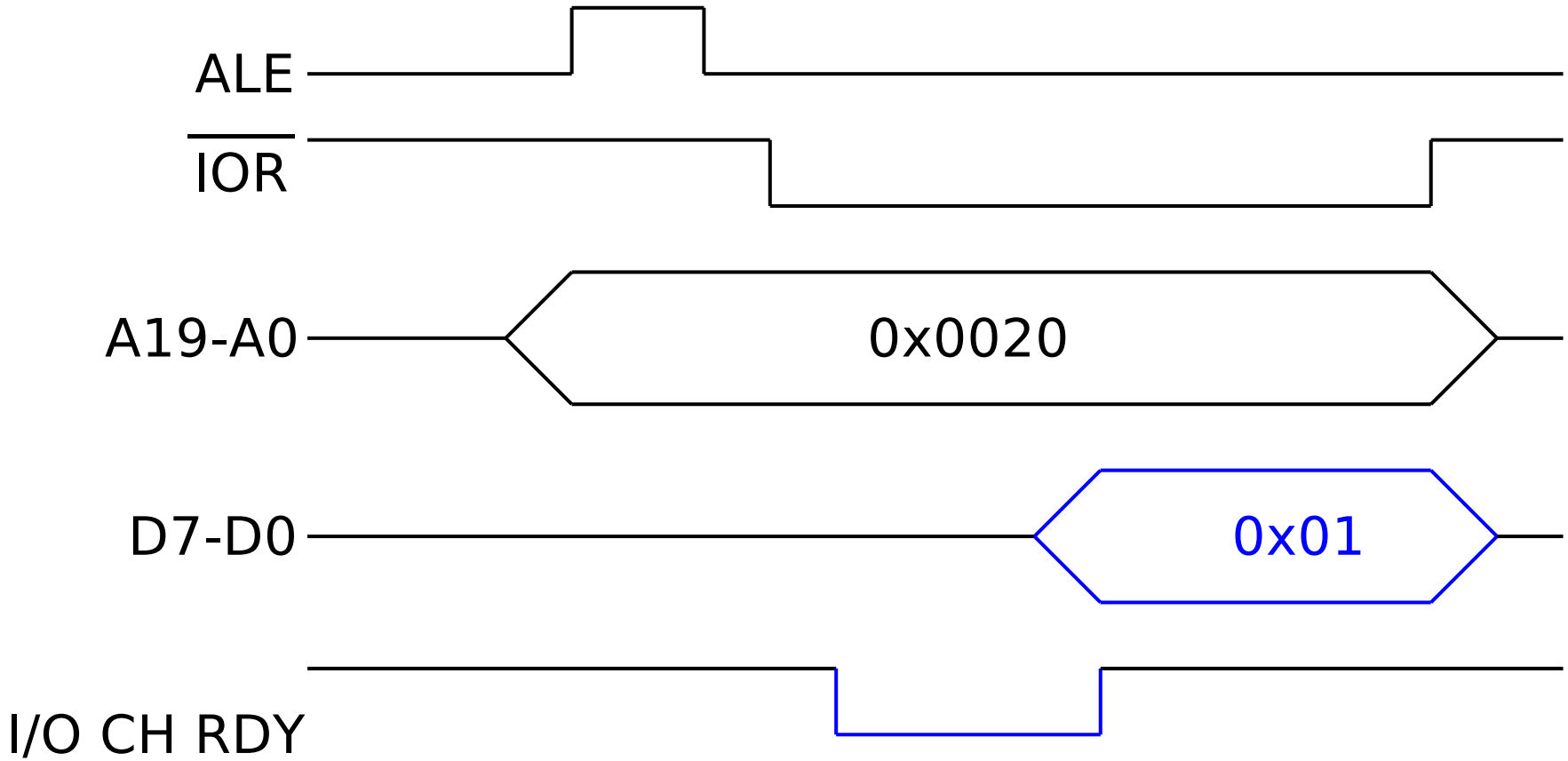
Busse - Überblick

- technische Sicht
  - „Kabelbündel“ zur Verbindung von Komponenten
  - physikalische Eigenschaften: Timing, „Tristate“, etc.
  - Arbitrierung
  - Topologie
- logische Sicht
  - ermöglicht Kommunikation mehrerer Komponenten
  - gezieltes Adressieren einzelner Komponenten
  - Senden oder Empfangen eines Datums/mehrerer Daten
  - ggf. weitere Kommunikationsaspekte: Speicher/IO, Menge der Übertragenen Informationen (1 Byte, 2 Byte), ...

ISA-Bus (auszugsweise)

- 20 Adressleitungen
- 8 Datenleitungen
- Kommunikationssteuerung:
  - MR: Lesen von Speicher
  - MW: Schreiben an Speicher
  - IOR: Lesen von IO
  - IOW: Schreiben an IO
  - ALE: Adresse gültig
  - I/O CH RDY: IO-Daten an Bus
- weitere Leitungen:
  - zur DMA-Steuerung
  - Reset
  - Masse, Versorgungspannungen: +5V, -5V, +12V, -12V
  - CLK: Taktsignal
- ...

Übertragungsbeispiel ISA-Bus (vereinfacht)



CPU liest von IO-Gerät mit Port `0x0020`.
Gerät antwortet mit Datum `0x01`.

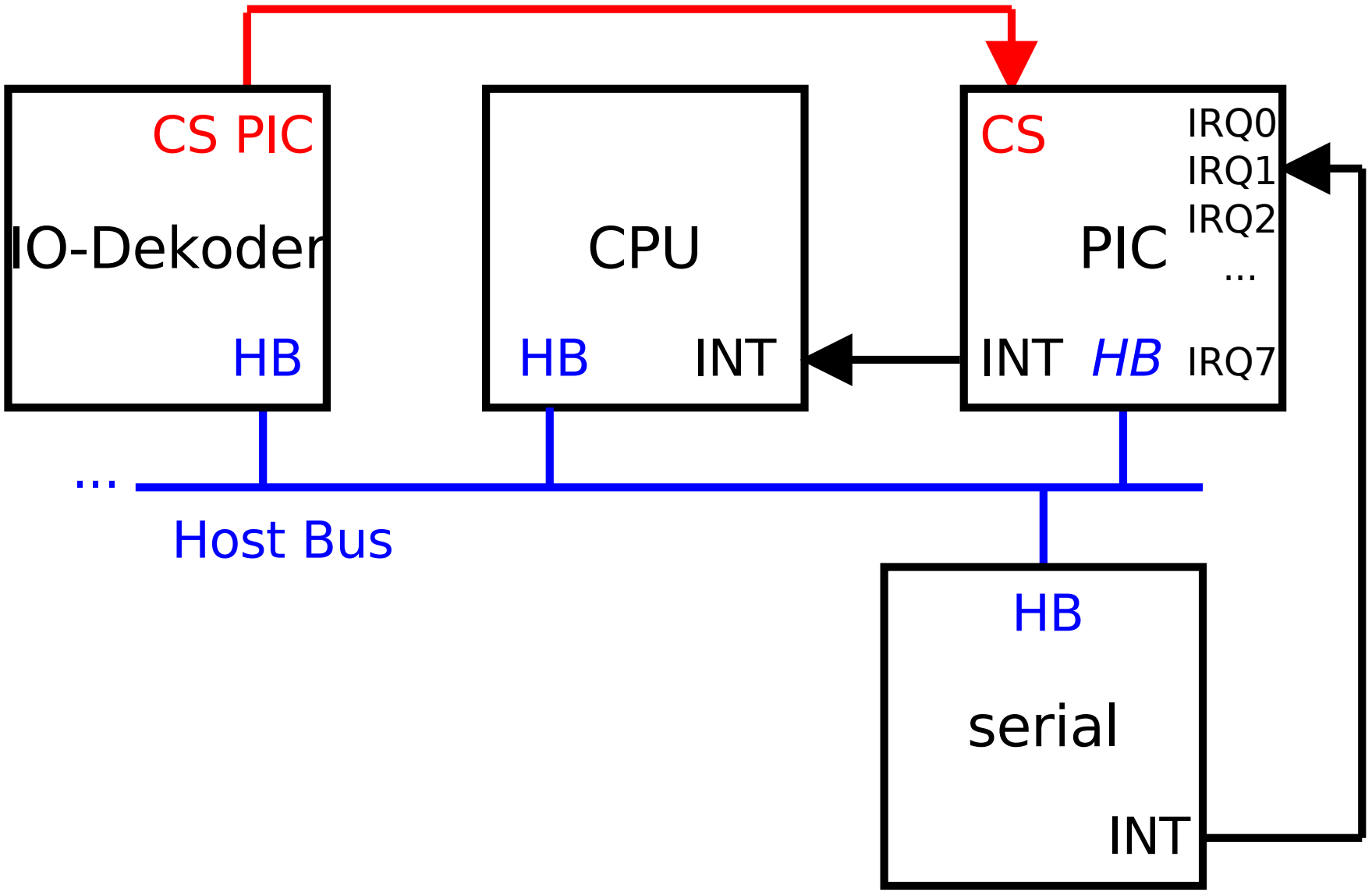
Relevanz für virtuelle Maschinen?

Was hiervon wird für virtuelle Maschinen benötigt?

Übersicht

- serielle Schnittstelle generiert Interrupts
- Intel PIC 8259a verwaltet Interrupts
  - Unterscheidung mehrerer Interruptquellen
  - Prioritäten
  - Ein-/Ausschalten von einzelnen Interrupts
  - „Zwischenspeicher“
  - Signalisieren der CPU
- IO-Dekoder übernimmt Adresssteuerung des PIC
- CPU empfängt Interruptsignal und führt Interruptbehandlung durch

Übersicht (2)


CS PIC
IO-Dekoder
HB
CPU
HB
INT
CS
IRQ0
IRQ1
IRQ2
...
PIC
INT
HB
IRQ7
...
Host Bus
HB
serial
INT

serielle Schnittstelle

- Nun auch Eingabe vom Gastsystem
- Eingabe: Lesen von Zeichen auf `stdin`
- Wenn Zeichen eingegeben $\rightarrow$ Interrupt
- weitere Eingabe dann blockiert
- Lesen von Adresse `0x00`
  - holt Zeichen ab und
  - setzt Interrupt zurück
  - ermöglicht Empfang von weiteren Zeichen

Intel PIC 8259a

- Dokumentation: `http://pdos.csail.mit.edu/6.828/2005/readings/hardware/8259A.pdf`
- 8 Interrupteingänge IRQ 0-7
- kann kaskadiert werden (hier nicht notwendig)
- diverse Priorisierungsvarianten
- Interrupts können einzeln maskiert werden
- Interrupts ausgelöst durch
  - steigende Flanke (Edge-triggered)
  - konstanten hohen Pegel (Level-triggered)
  - einstellbar nur chip-weit
  - später: einstellbar für einzelne Interrupteingänge

Intel PIC 8259a: Fully nested Mode

1. Anlegen eines Interrupts aktualisiert `irr`
2. Wenn nicht maskiert $\rightarrow$ Interrupt an CPU
3. CPU führt Interrupt Acknowledge Zyklus aus
4. Interrupt mit höchster Priorität wird in `isr` kopiert
   - Fully nested mode: Priorität gemäß Interrupt Nummer (wobei 0 „höchste“ Priorität)
5. Entsprechender Interrupt wird aus `irr` gelöscht
6. Interruptvektor wird an CPU übermittelt
7. Wenn „*automatic end of interrupt*“: CPU-Interruptleitung wird bei Ende des Interrupt Acknowledge Zyklus zurückgenommen

Ein Interrupt höherer Priorität kann einen Interrupt-Handler niedriger Priorität unterbrechen.

IO-Dekoder

- Intel PIC 8259a hat nur **eine** Adressleitung
  $\Rightarrow$ weiß nicht, ob er gemeint ist
- Chip-Select-Eingang signalisiert Daten für PIC
- IO-Dekoder steuert Chip-Select-Eingänge
- „großer Demultiplexer“
- IO-Adressbereich `0x0020-0x003f` setzt Chip-Select des PIC

Interruptbehandlung der CPU (Wiederholung)

1. CPU führt **Interrupt-Acknowledge-Zyklus** durch und liest Vektor vom Bus.
2. Zustandssicherung:
   - 2.1 EFLAGS → Stack
   - 2.2 CS → Stack
   - 2.3 EIP → Stack
3. bei Interrupts: `IF=0`
4. Eintrag aus Interrupttabelle wird angesprungen
5. *Behandlungsroutine wird ausgeführt*
6. *IRET*
7. `EFLAGS`, `CS` und `EIP` werden vom Stack geholt und der Zustand somit wiederhergestellt

Da fehlt doch was. . . ?

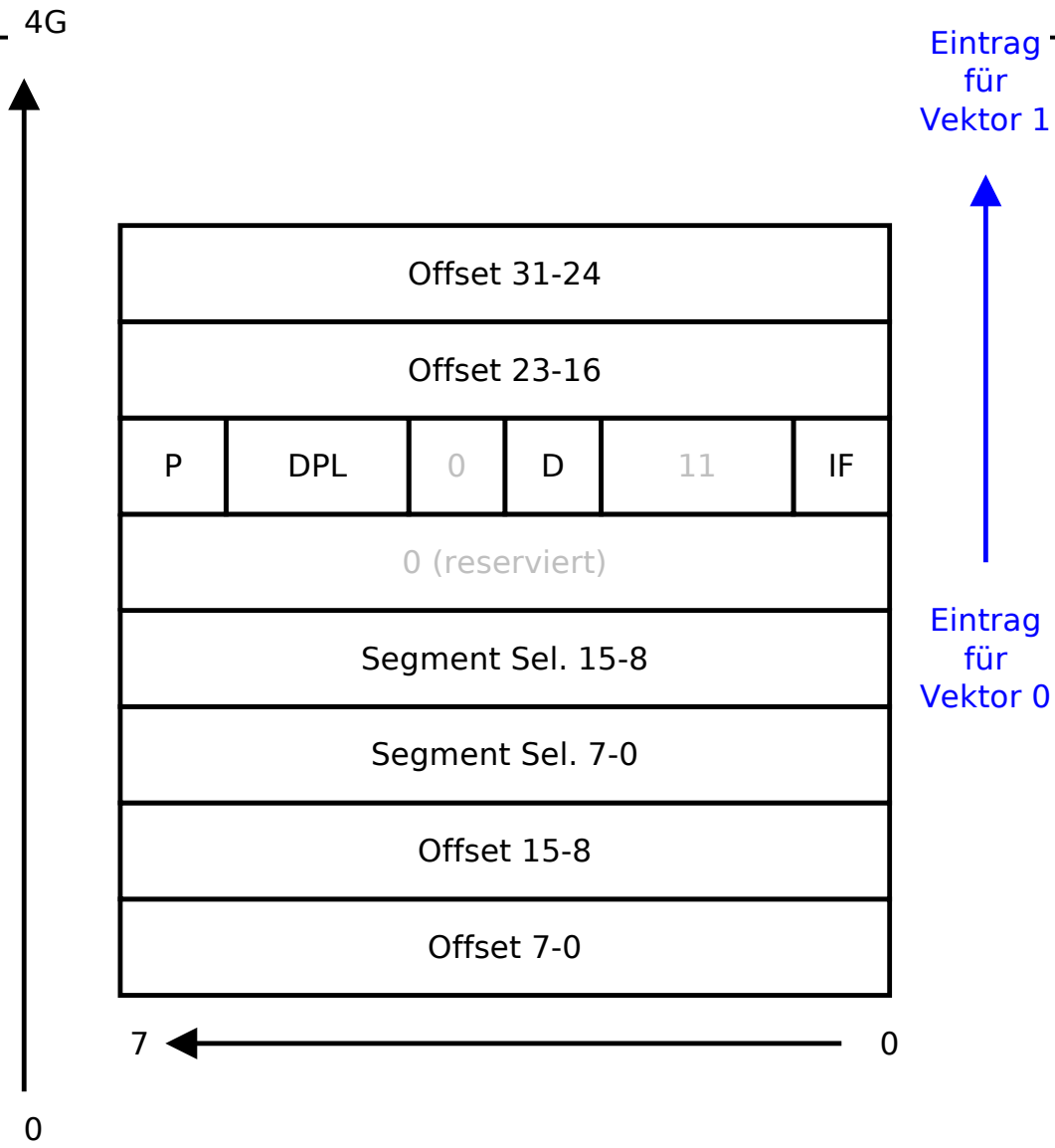
- Woher weiß denn der PIC, dass er einen Vektor auf den Bus legen soll?
- Wie findet die CPU denn die Interrupttabelle?

idtr

- Register `IDTR` enthält Zeiger auf IDT, sowie Größe der IDT in Bytes.
- `lidt` dient zum Laden des `IDTR`
- Aufbau Operand von `lidt`:

Offset	Inhalt	Größe
2	Pointer auf Anfang IDT	32 Bit
0	Größe der IDT	16 Bit

IDT



IF	Interrupts ausschalten (0)?
D	1: 32 Bit Ziel, 0: 16-Bit Ziel
DPL	maximaler Ring
P	Eintrag gültig?

Siehe auch:

„Intel 64 and IA-32 Architectures Software Developer's Manual“, Volume 1: Basic Architecture, Kapitel 5.11.