Grundbegriffe der Informatik

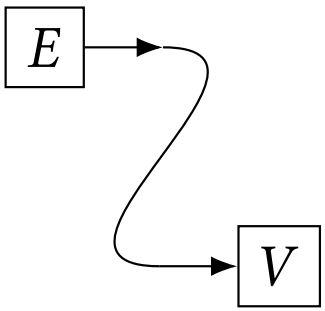
Kapitel 10: Prozessor

Thomas Worsch

KIT, Institut für Theoretische Informatik

Wintersemester 2015/2016

Drähte verbinden „Erzeuger“ und „Verbraucher“



für einen „Draht“ gibt es *drei* Möglichkeiten

- es wird eine 0 übertragen
- es wird eine 1 übertragen
- es wird nichts übertragen
  - manchmal durch Z repräsentiert

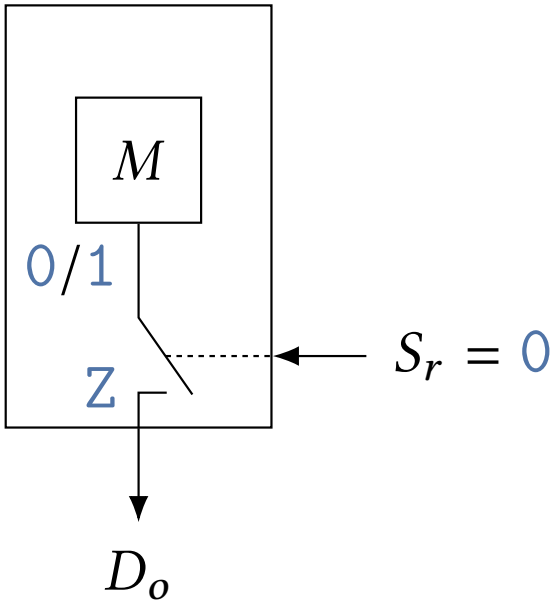
an jedem Drahtende

- *Erzeuger:* kann Bit schreiben (oder nichts tun) oder
- *Verbraucher:* kann Bit lesen (oder nichts tun)

Draht kann weder 0 noch 1 speichern

- wenn kein Erzeuger ein Bit „liefert“,
  wird nichts übertragen

„Erzeuger“ für ein Bit



Bestandteile

- interne Schaltung M
- u. U. Eingänge, weitere Steuerleitungen

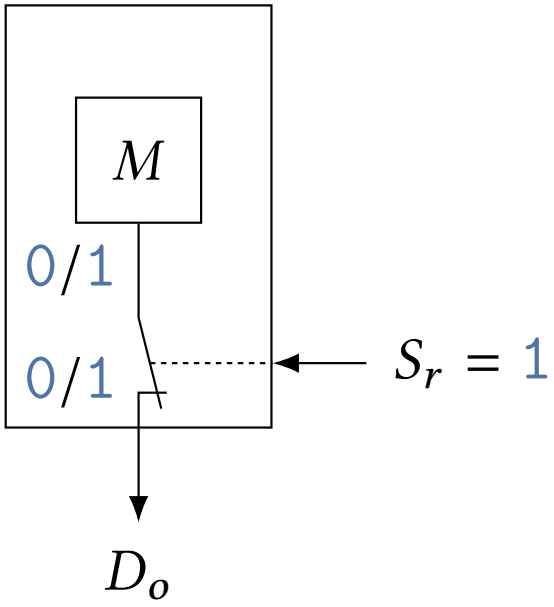
Steuerleitung S_r

- liefert ggf. Befehl zur „Ausgabe“: $S_r = 1$

Datenleitung D_o

- liefert ein Bit, falls $S_r = 1$ ist
- sonst nichts — „Z“

„Erzeuger“ für ein Bit



Bestandteile

- interne Schaltung M
- u. U. Eingänge, weitere Steuerleitungen

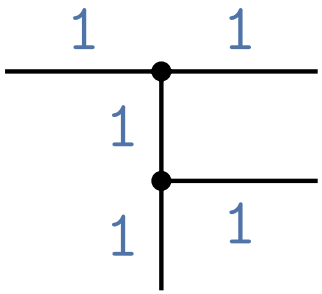
Steuerleitung S_r

- liefert ggf. Befehl zur „Ausgabe“: $S_r = 1$

Datenleitung D_o

- liefert ein Bit, falls $S_r = 1$ ist
- sonst nichts — „Z“

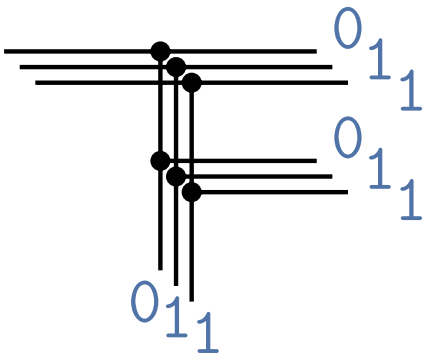
Drähte können mehrere Erzeugerausgänge mit Verbrauchern verbinden



Drähte kann man miteinander verbinden, so etwas nennt man einen *Bus*

- überall wird der gleiche Wert übertragen
- oder überall wird nichts übertragen

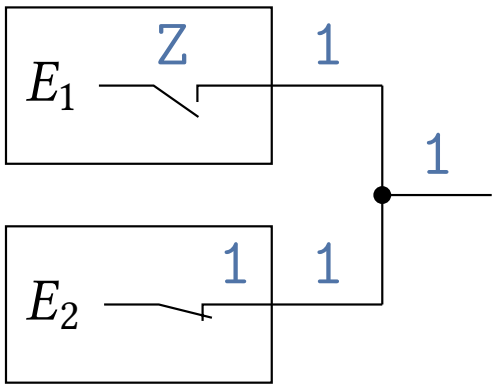
Drähte können mehrere Erzeugerausgänge mit Verbrauchern verbinden



Drähte kann man miteinander verbinden, so etwas nennt man einen *Bus*

- überall wird der gleiche Wert übertragen
- oder überall wird nichts übertragen

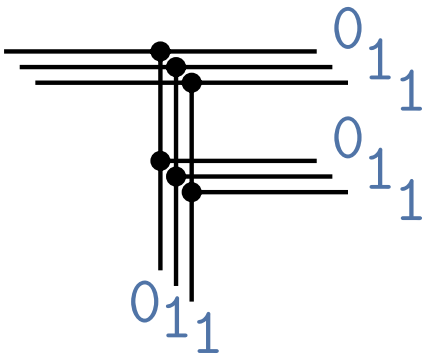
oft auch mehrere Leitungen „parallel“



Ausgänge mehrerer Erzeuger können miteinander verbunden sein

- erlaubt: maximal einer liefert ein Bit

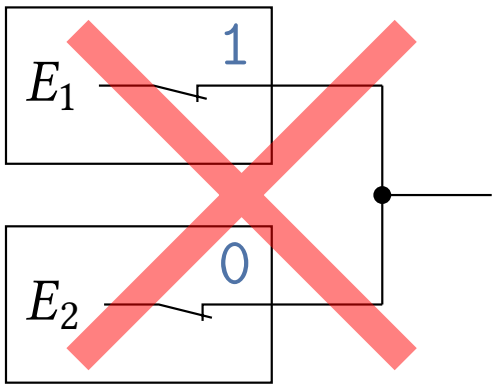
Drähte können mehrere Erzeugerausgänge mit Verbrauchern verbinden



Drähte kann man miteinander verbinden, so etwas nennt man einen *Bus*

- überall wird der gleiche Wert übertragen
- oder überall wird nichts übertragen

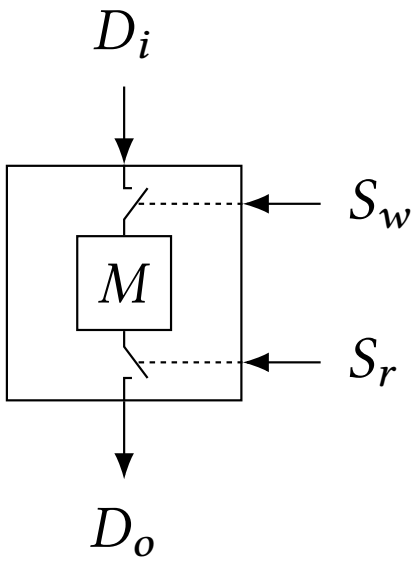
oft auch mehrere Leitungen „parallel“



Ausgänge mehrerer Erzeuger können miteinander verbunden sein

- erlaubt: maximal einer liefert ein Bit
- verboten: mehrere Erzeuger liefern gleichzeitig Bit auf gleichen Bus

Einfaches „Speicher-Element“ für ein Bit



Bestandteile

interner Speicher M

- *immer* entweder in Zustand 0 oder in Zustand 1

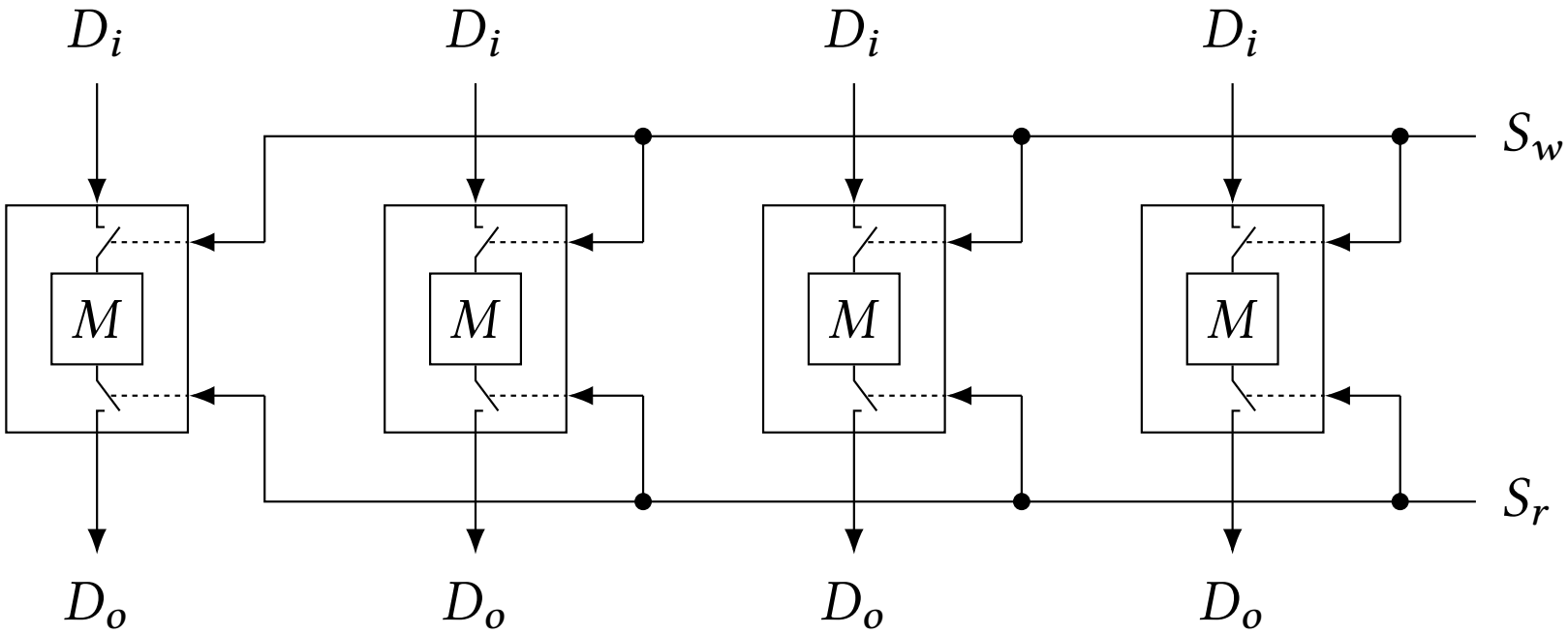
Steuerleitungen

- auf S_w wird Schreibbefehl geliefert
- auf S_r wird Lesebefehl geliefert

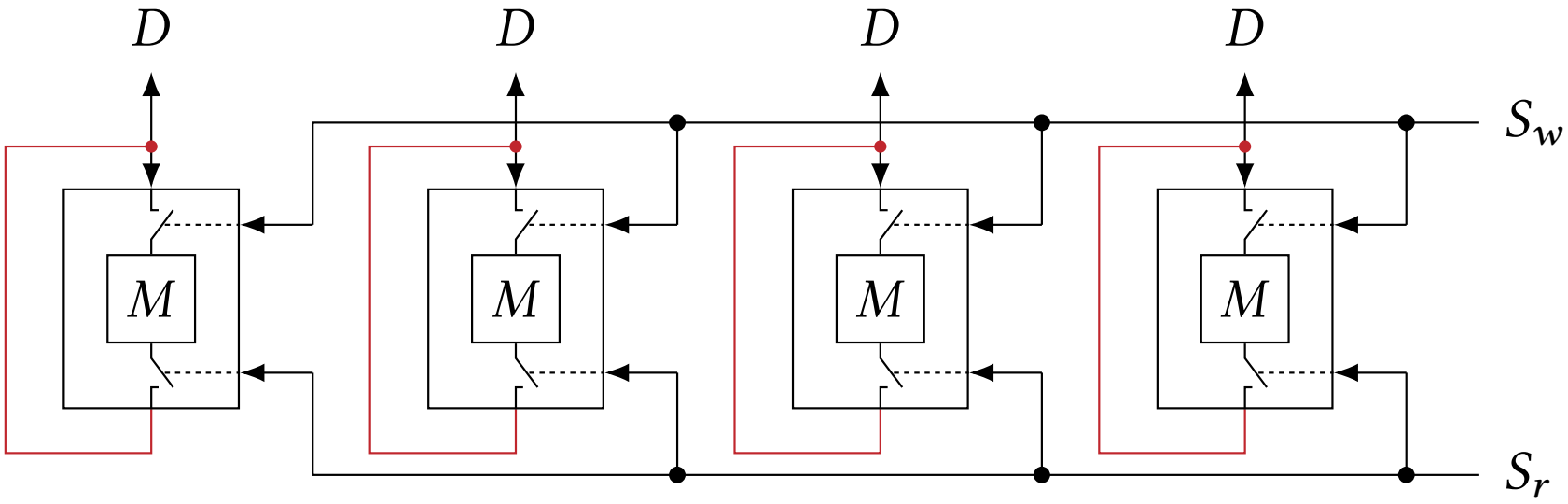
Datenleitungen

- auf D_i wird neu zu speicherndes Bit geliefert
- auf D_o wird gelesenes Bit geliefert

Register — mehrere Ein-Bit-Speicher nebeneinander



Register — mehrere Ein-Bit-Speicher nebeneinander



manchmal D_i und D_o miteinander verbunden

Hauptspeicher für die Mima

Größen

- Adressen: 20 bit
- Werte: 24 bit
  - sogenannte (Speicher-)Worte

Programmspeicher

- ein Maschinenbefehl je Wort
  - 4 bit Befehlscodierung und 20 bit Adresse/Wert oder
  - 8 bit Befehlscodierung, Rest irrelevant

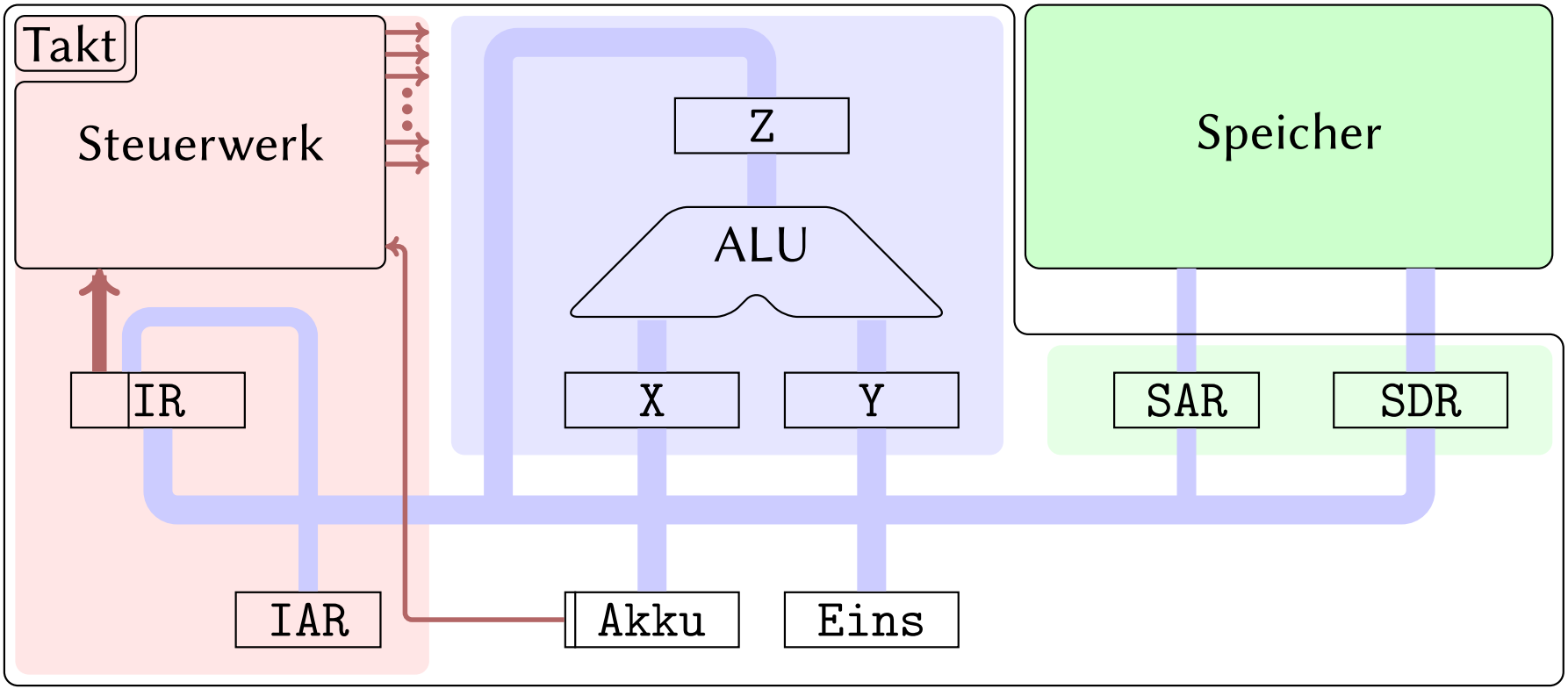
Datenspeicher

- Eingaben, Zwischenergebnisse, Ausgaben
- Zweierkomplementdarstellung

Trennung von Programm und Daten

- per Konvention, nicht erzwungen

die MIMA — ein idealisierter Prozessor



Steuersignale veranlassen Register und ALU,
auf einem Bus zu lesen bzw. zu schreiben
Meldesignale beeinflussen Arbeitsweise des Steuerwerks

MIMA-Befehle (1a) — Datentransport Akku ↔ Speicher

Notation

- MIMA: Folge von 24 Bits
  - z. B. 001000000000000000101010
- Mensch: symbolisch
  - STV 101010
  - STV 42
- $M(adr)$: Hauptspeicherinhalt an Adresse *adr*

MIMA-Befehle

- LDC *const* «*load constant*»
  - Akku ← *const*
- LDV *adr* «*load value from address*»
  - Akku ← $M(adr)$
- STV *adr* «*store value at address*»
  - $M(adr)$ ← Akku
- mehr später

MIMA-Befehle (1b) — Datentransport mit indirekter Adressierung

MIMA-Befehle

- `LDIV` *adr* «*load value indirect from address*»
  - `Akku` $\leftarrow M(M(adr))$
- `STIV` *adr* «*store value indirect at address*»
  - $M(M(adr)) \leftarrow$ `Akku`

vergleiche Vorlesung «Programmieren»

- Objekte
- Referenzen

MIMA-Befehle (2a) — für die ALU

arithmetisch/logische Befehle

- `ADD` *adr*
  - `Akku` ← `Akku` „+“ Speicher(*adr*)
  - für eine gewisse Interpretation von „+“
- «logische» Operationen
  - `Akku` ← `Akku` „op“ *Speicher*(*adr*)
  - eine `1` entspricht «wahr»
  - eine `0` entspricht «falsch»
  - Anwendung an allen 24 Stellen der Operanden separat
- `AND` *adr*
- `OR` *adr*
- `XOR` *adr*
  - exklusives Oder, d. h. Addition modulo 2

MIMA-Befehle (2b) — mehr für die ALU

einstellige Operationen auf `Akku`

- `NOT`
  - Invertierung der `Akku`-Bits — wie exklusives Oder mit $11\cdots11$
- `RAR` «*rotate accumulator right*»
  - Rotation der `Akku`-Bits nach rechts
    aus $x_{23}x_{22}x_{21}\cdots x_2x_1x_0$ wird $x_0x_{23}x_{22}x_{21}\cdots x_2x_1$

Vergleichsoperation

- `EQL` *adr* «*equal?*»
  - $\texttt{Akku} \leftarrow \begin{cases} 11\cdots11 & \text{falls } \texttt{Akku} = M(adr) \\ 00\cdots00 & \text{sonst} \end{cases}$

$$= \begin{cases} -1 & \text{falls } \texttt{Akku} = M(adr) \\ 0 & \text{sonst} \end{cases}$$

Sprünge ändern die normale Reihenfolge der Programmabarbeitung

unbedingter Sprung

- `JMP` *adr* «*jump*»
  - setze fort mit Befehl in Adresse *adr*
- Fortsetzung der Programmausführung an explizit angegebener Stelle

```
000101   LDC 111        «lädt 7 in den Akku»
000110   STV 111101     «speichert die 7 an Adresse 61»
000111   JMP 011000     «springt zum Befehl an Adresse 56»
   ⋮
011000   LDC 101011     «lädt 43 in den Akku»
011001   ADD 111101     «addiert Wert von Adresse 61»
                        «jetzt steht 50 im Akku»
```

Bedingte Sprünge — Ausführung hängen vom Inhalt des `Akku` ab

Zweierkomplement

- die *negativen* Zahlen haben höchstwertiges Bit gesetzt

bedingter Sprung

- `JMN` *adr* «*jump if negative*»
- setze fort mit Befehl in Adresse *adr*, falls in `Akku`
  - höchstwertiges Bit auf `1` ist
  - eine negative Zahl repräsentiert ist