Rechnerarchitektur

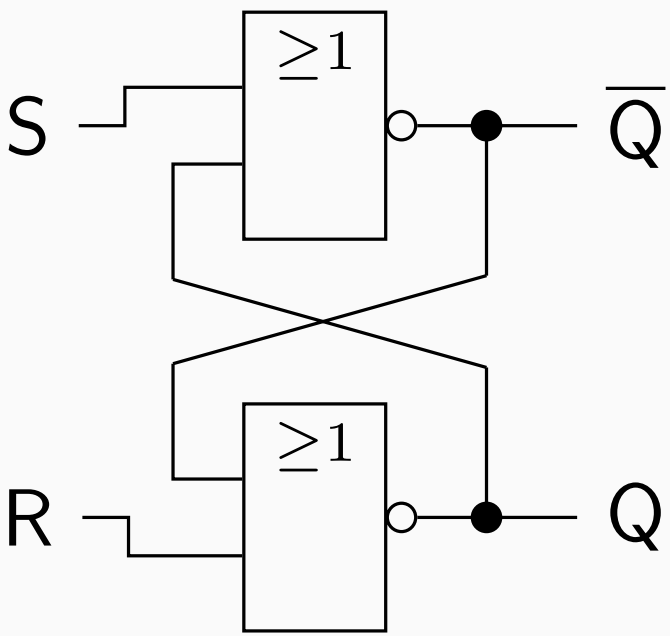
Einführung in die Informatik & Rechnerarchitektur (EIR1/EIF1)

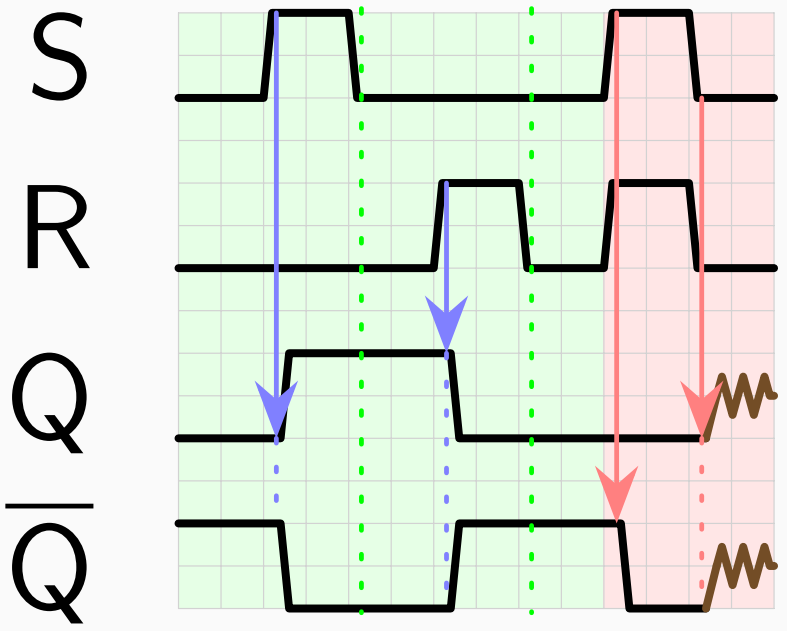
Erik Pitzer

SE & MBI – FH Hagenberg – WS 2025/26

Speicher

RS-Latch

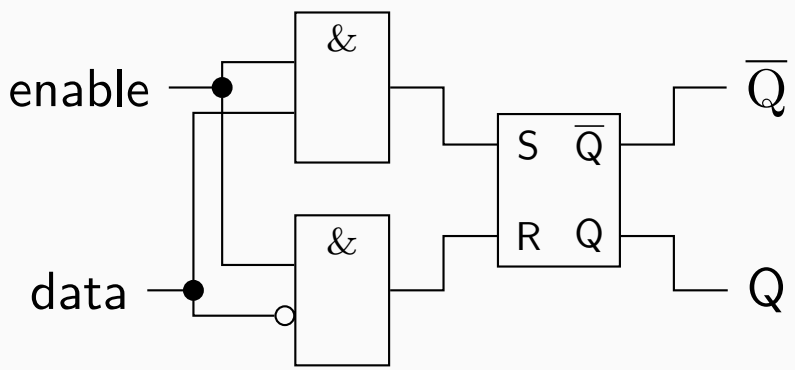


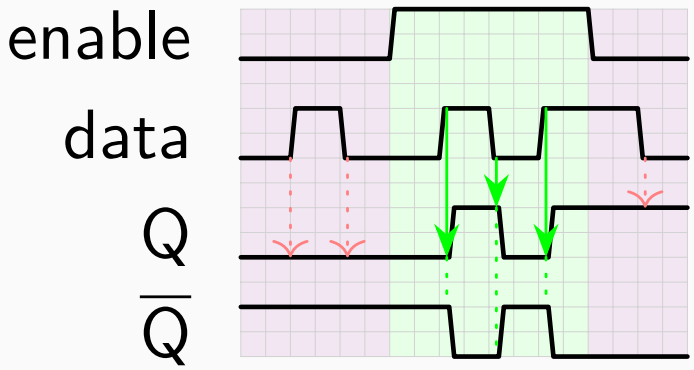


- falls R und S gleichzeitig von 1 auf 0 wechseln fängt das RS-Latch zu schwingen an
- `demos/memory.circ` `RS_Latch`

D-Latch

- Eingänge R und S werden nur noch gemeinsam geschaltet um ungültige Zustände zu verhindern

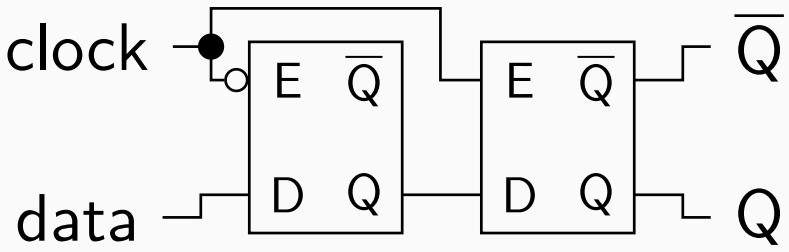


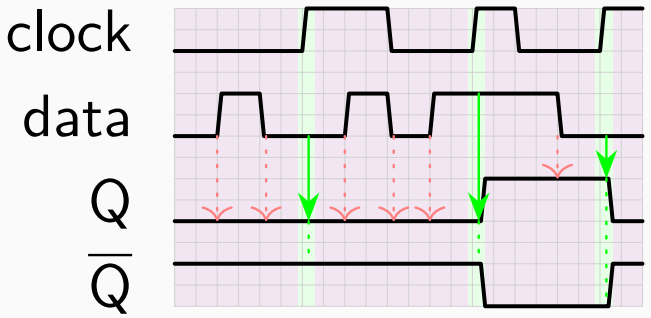


- `demos/memory.circ` `D_Latch`

D-FlipFlop

- Beim Latch wird der Wert übernommen solange `enable=1`
- Beim FlipFlop wird der Wert zum Zeitpunkt des Umschaltens von `clock` von `0` auf `1` übernommen

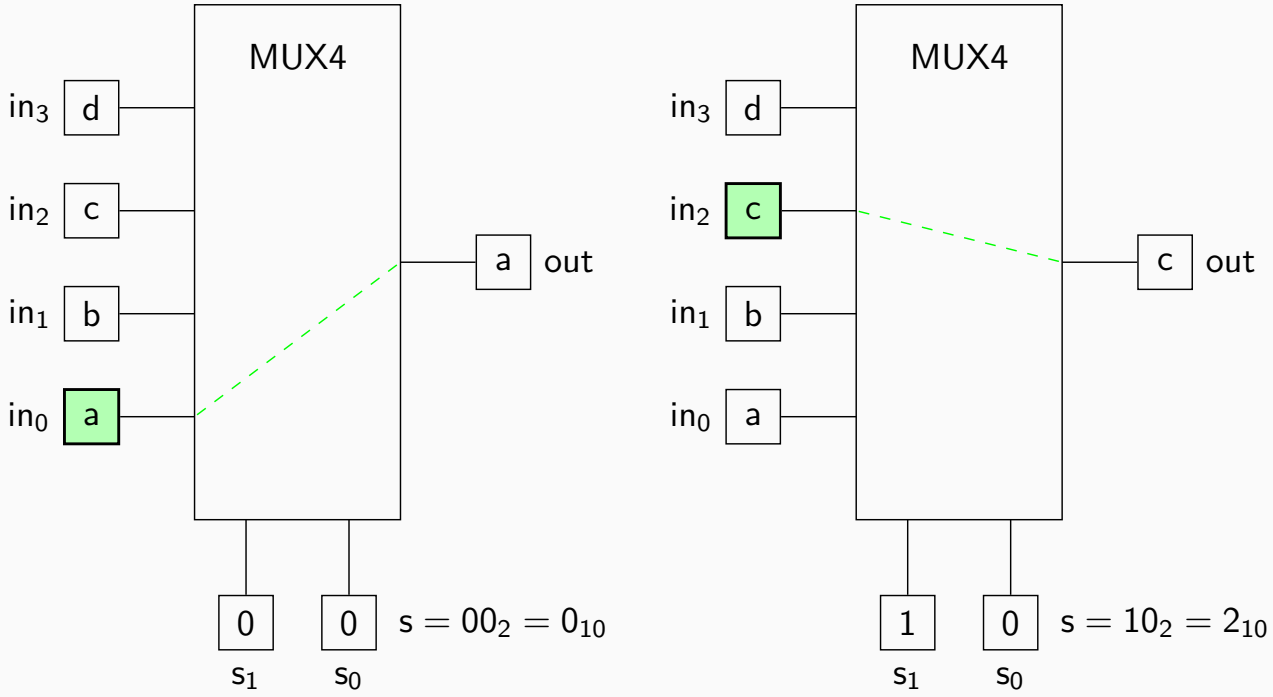




- `demos/memory.circ` `D_FlipFlop`

Multiplexer

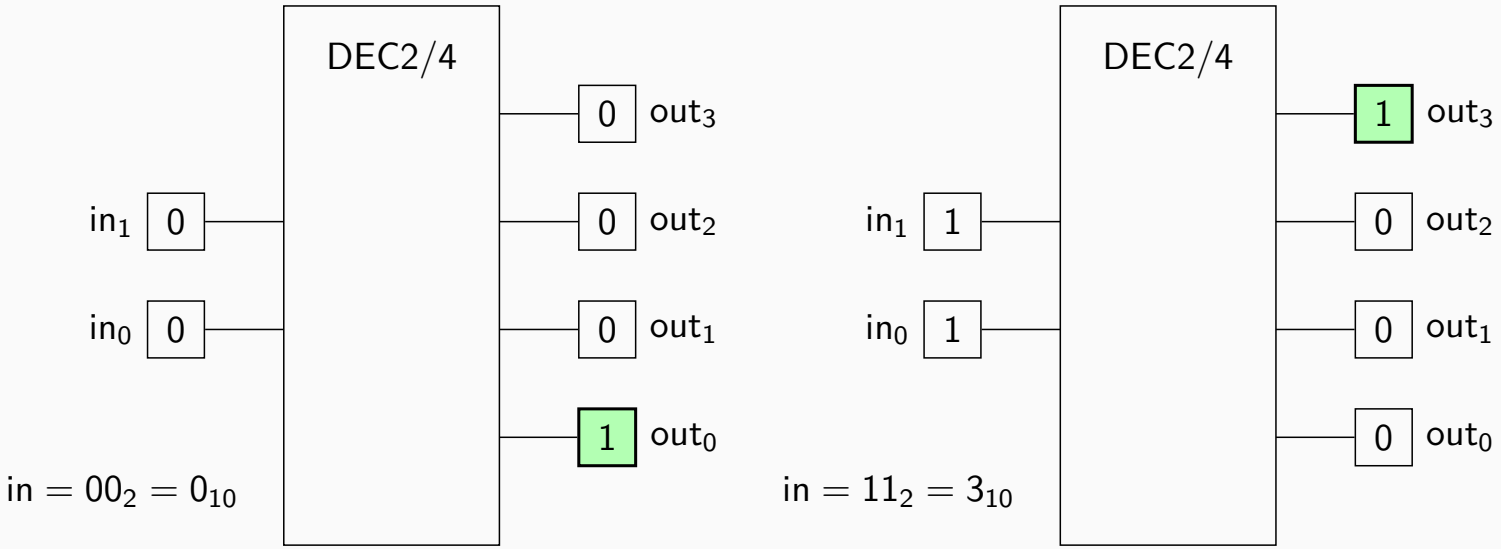
- Multiplexer schaltet einen von mehreren Eingängen durch
- Auswahl erfolgt über die Select-Leitung (z.B. $s_1 s_0$ für **MUX4**)
- Die Daten können dabei auch mehrere Bits sein



- `demos/memory.circ` `mux_4`

Decoder

- Decoder schaltet immer genau einen Ausgang ein, bestimmt durch Binärzahl die an den Eingängen anliegt
- “decodiert” also die Eingabe



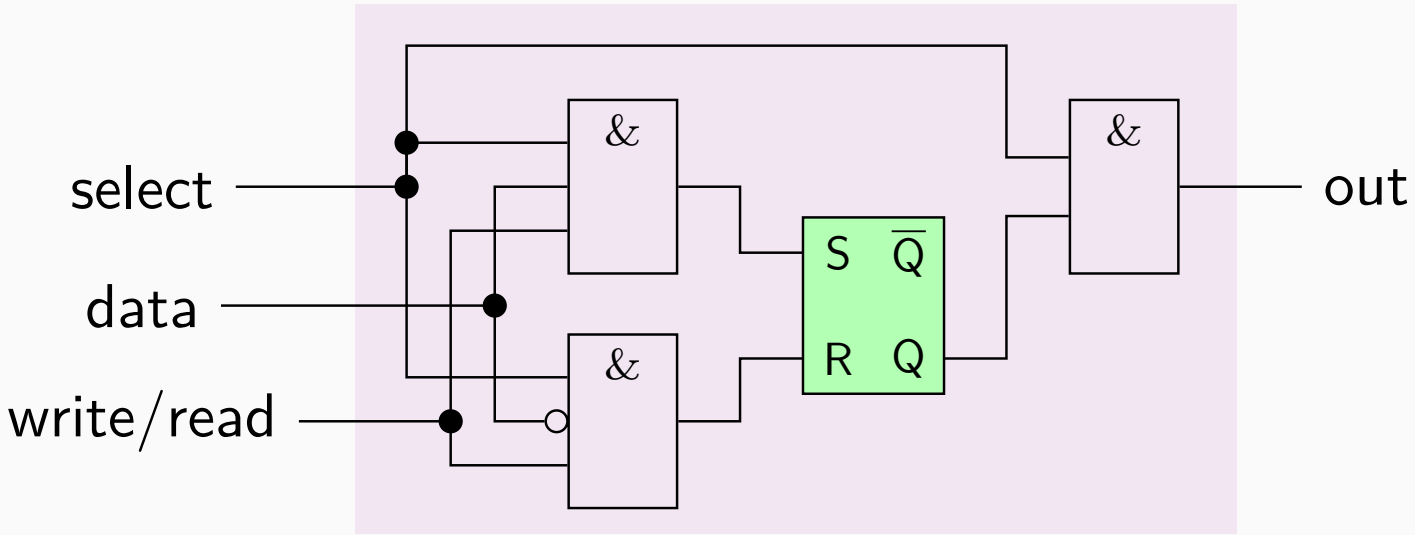
- `demos/memory.circ` `dec_2_4`

Arten von Speicher

- Verschiedene Technologien mit Vor- und Nachteilen
  - Geschwindigkeit
  - physische Größe
  - Speicherkapazität
  - Preis
- schneller bedeutet oft geringere Speicherkapazität und umgekehrt
- allgemeiner Speicher mit beliebigem Zugriff auf verschiedene Adressen, engl. *Random Access Memory* (RAM)
- Static Random Access Memory (SRAM)
  - schneller
- Dynamic Random Access Memory (DRAM)
  - größere Kapazität

SRAM: 1-Bit Speicherzelle

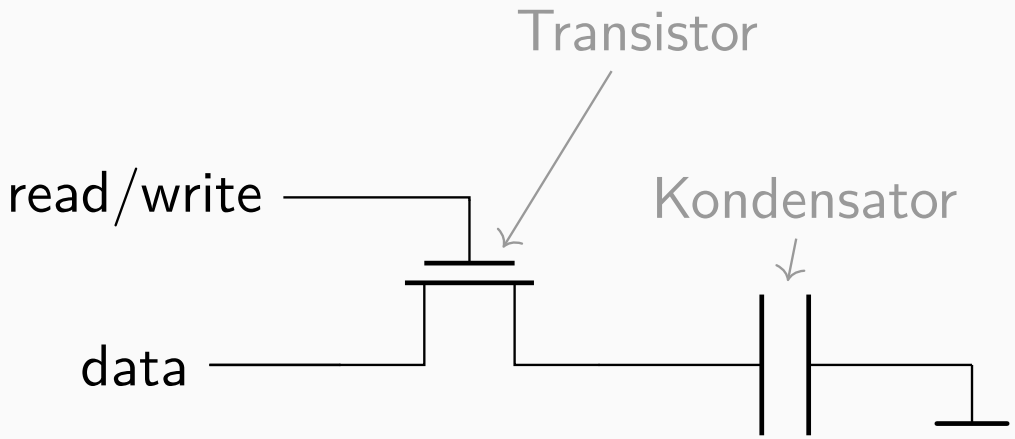
- statische Speicherzelle aus Transistoren, Latches oder Flip-Flops
- typischerweise 4–10 Transistoren
- z.B. mit Hilfe von R-S-Latch
- benötigt konstante Stromversorgung $\rightarrow$ Hitzeentwicklung



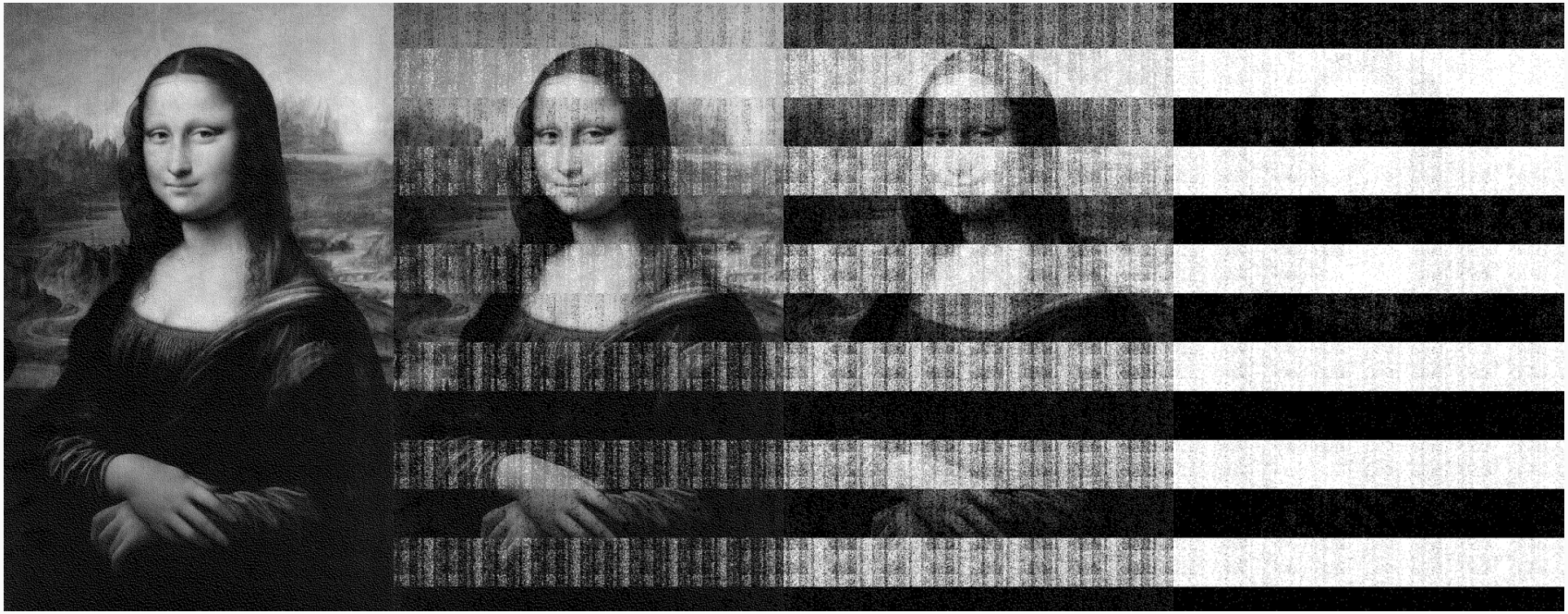
`demos/memory.circ` `mem_cell`

DRAM: 1-Bit Speicherzelle

- Speichern einer kleinen elektrischen Ladung in einem Kondensator, Schalten mit Transistor
- keine permanente Stromversorgung notwendig → kühler
- weniger Bauteile → physisch kleiner, billiger, höhere Speicherkapazität
- komplexere Elektronik zum lesen und schreiben → langsamer
  - benötigt Refresh, da Kondensatoren mit der Zeit entladen



DRAM: Speicherzerfall ohne Refresh



(a) 1s **(b)** 5s **(c)** 60s **(d)** 300s

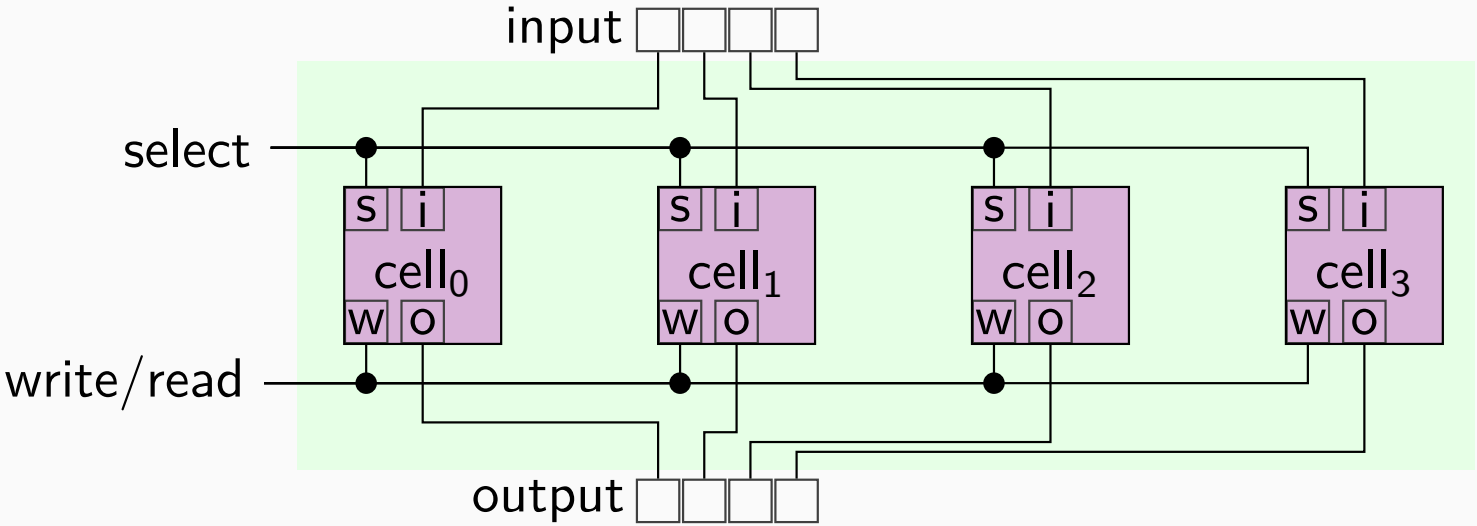
Quelle: Halderman et al. ”Lest we remember: cold-boot attacks on encryption keys.”, Communications of the ACM 52.5 (2009): 91-98.

Vergleich SRAM vs. DRAM

- Vorteile von SRAM
  - geringere Zugriffszeiten
  - höhere Transferraten
- Vorteile von DRAM
  - geringer Stromverbrauch
  - geringe physische Größe $\rightarrow$ hohe Speicherdichte
  - einfacherer Aufbau $\rightarrow$ billiger

Register aus SRAM

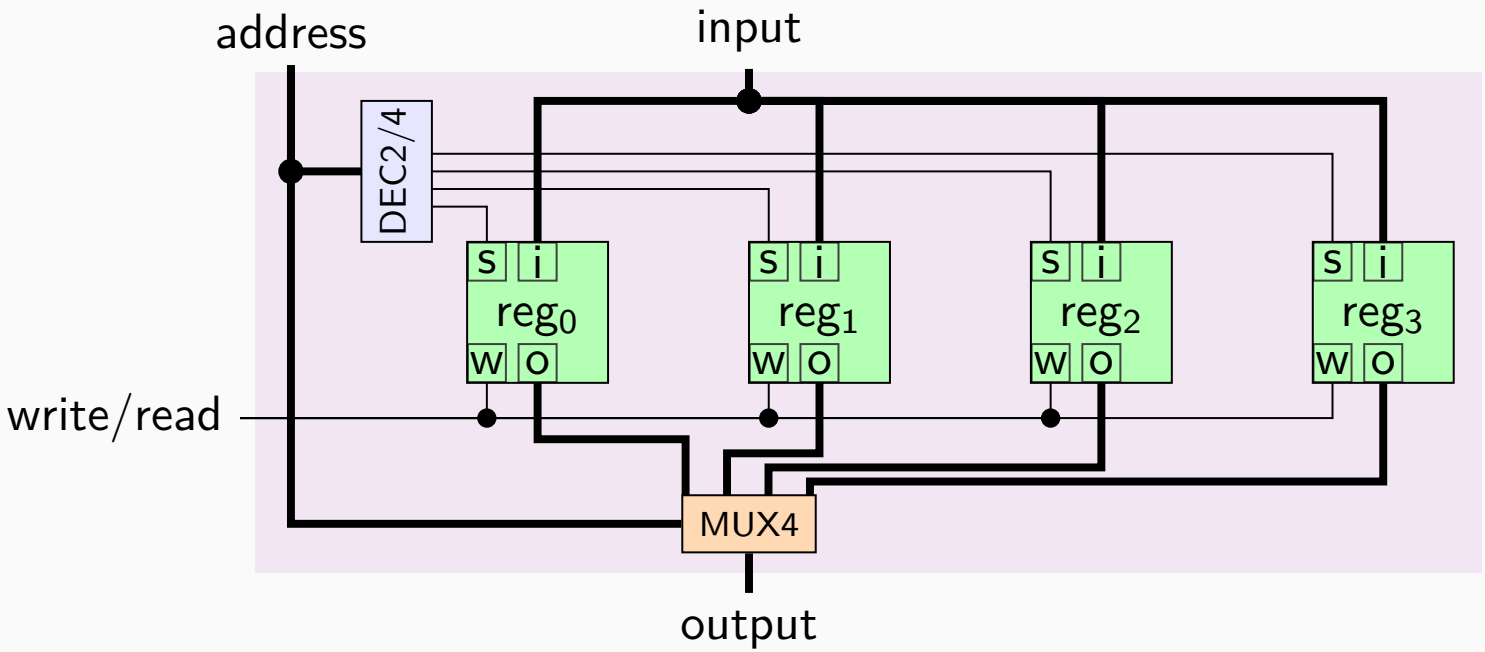
- Verbindung von mehreren 1-Bit Speicherzellen zu Register mit mehreren Bits
- typische Größen: 4, 8, 16, 32, 64, 128 Bits



- `demos/memory.circ` `register_4`

Registerfile mit SRAM

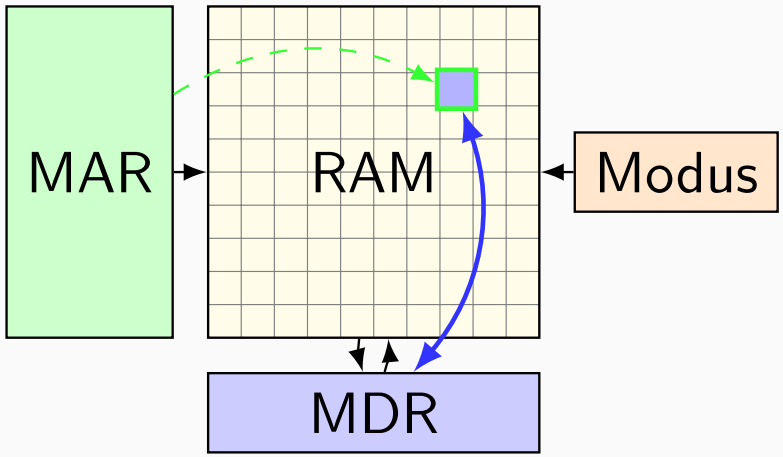
- Ansteuern mehrere Register mit Hilfe einer Adresse
- Auswahl und Auslesen mit Hilfe von Decoder und Multiplexer



- `demos/memory.circ` `register_file`

Speicher Zugriff über Schnittstellen-Register für DRAM

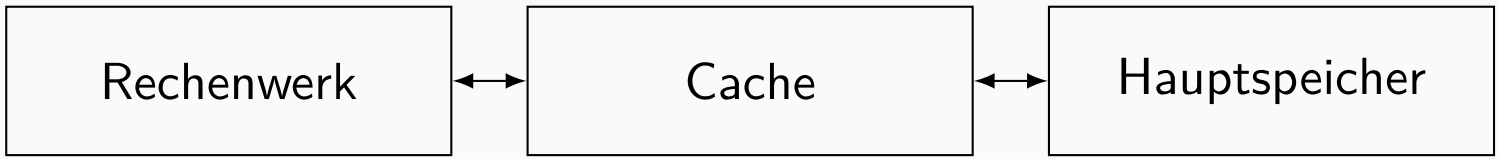
- Adressregister: Memory Address Register (MAR)
  - setzen der Adresse, also der Stelle im RAM auf der operiert werden soll
- Datenregister: Memory Data Register (MDR)
  - falls geschrieben werden soll: enthält den zu schreibenden Wert
  - falls gelesen wurde: enthält den gelesenen Wert
- Modus (Lesen/Schreiben/Warten)



- `demos/memory.circ` `mem` / `RAM+MAR+MDR`

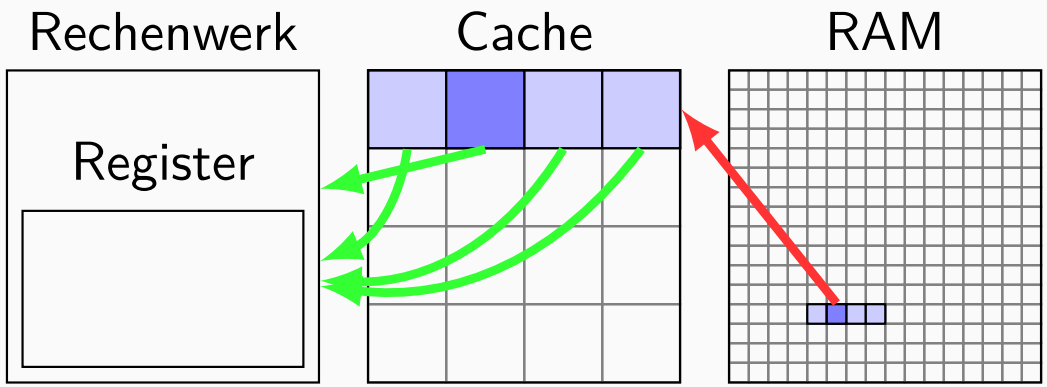
Caching

- für die Praxis sehr relevant
- Geschwindigkeitsunterschied von Registern in der CPU und dem Hauptspeicher wird immer größer
- Idee: häufig genutzte Werte zwischenspeichern (*Caché*, franz. Versteck)



Caching Lokalität

- CPU möchte Wert vom RAM lesen, Cache holt auch benachbarte Werte (sogenannte Cache Line)
- bei folgenden Zugriffen können diese Werte dann aus dem schnelleren Cache gelesen werden



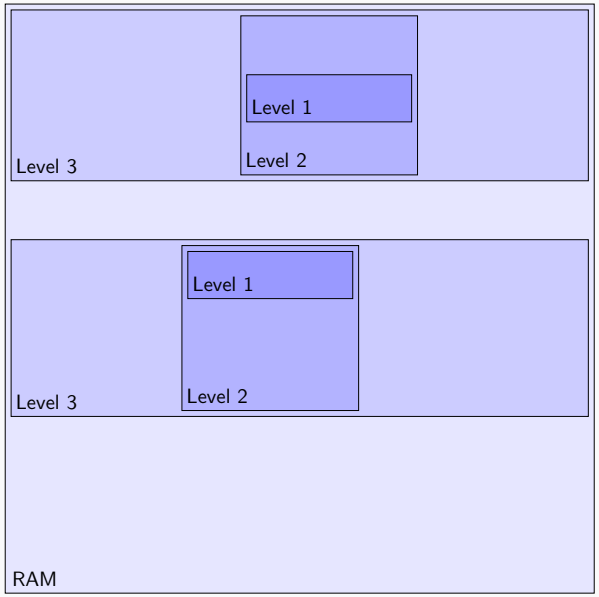
- wenn Cache voll ist wird z.B. ältester Eintrag ersetzt
- Algorithmendesign: benachbarte Adressen ausnutzen

z.B. in einem array kann man auf benachbarte Adressen sehr schnell zugreifen

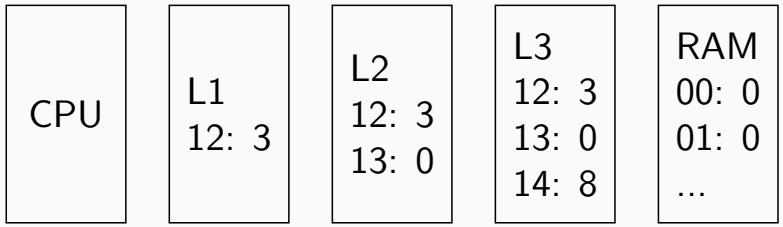
Speicherhierarchie

- in realer Hardware typischerweise mehrstufig (Levels 1 – 3)
  - meist direkt in der CPU integriert
- Jedes höhere Level cached alles aus den unteren Levels
  - RAM enthält alles
- Höhere Level haben höhere Kapazität aber geringere Geschwindigkeit

- Zugriff startet bei höchstem Level
- Adresse nicht verfügbar → Zugriff auf nächstes Level
  - Level 1
  - Level 2
  - Level 3
  - RAM

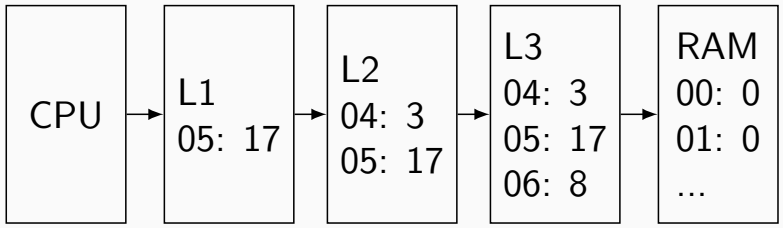


Beispiel



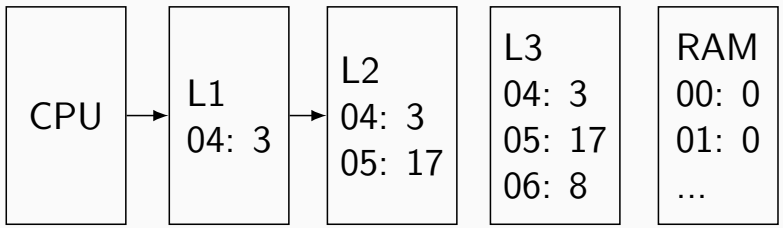
wir können viel schneller rechnen, als auf den RAM zugreifen

- Lesen von Adresse 0x05 (L1→L2→L3→RAM)



CPU Anfrage -> schaut in den Cashes, wenn nicht da holt ers aus dem RAM,
speichert dann den Wert bei 05 mit umliegenden Werten ins L3, reduziert ins L2, reduziert ins L1, von dort wird der Wert von der CPU gelesen. die CPU kann grundsätzlich (evtl gibts Optimierungen) nur
den L1 lesen und schreiben

- danach Lesen von Adresse 0x04 (L1→L2)



Cache Performance Daten

- Geschwindigkeit vs. Kapazität

	CPU Takte	Kapazität [MB]
CPU Register	1	0.001
Level 1 Cache	2.5	0.100
Level 2 Cache	20	4.000
Level 3 Cache	50	16.000
Hauptspeicher	300	32 000.000

- Was wird gecached?
  - jeder Wert auf den im Speicher zugegriffen wird (lesen oder schreiben)
  - Ersetzt/Überschreibt ältere oder selten genutzte Werte

typische/durchschnittliche Werte

Speichertechnologien Performance Daten

Medium	Kapazität[MB]	Zugriffszeit[ns]	Datenrate[GB/s]
Register	0.000004	0.25	1 000.000
Cache	16	0.50	100.000
RAM	32 000	10.00	25.000
SSD	512 000	100.00	0.768
Festplatten	1 000 000	3 000.00	0.128
CD/DVD	2 000	25 000.00	0.072
Disketten	1	100 000.00	0.000128

wichtig:
Geschwindigkeit vs. Kapazität

typische/durchschnittliche Werte