3.3 Speicherstrukturen

Zur Vorlesung Rechenanlagen

SS 2019

Multiportspeicher

Wir haben Schaltungen kennengelernt, die Zustände 0 und 1 halten können (z.B. D-Latches). Um große Speicher bauen zu können, müssen diese Elementarbausteine sehr klein sein und speziell verschaltet werden.

Bei Halbleiterspeichern nutzt man dazu sehr ausgefeilte Strukturen, das Grundprinzip des Aufbaus ist aber sehr ähnlich zu dem, was wir hier vorstellen, so dass uns dies als Orientierung dienen kann.

Unsere Maschine WüRC hatte Befehle, in denen bis zu 2 Speicherworte geändert und bis zu 3 Speicherinhalte gelesen wurden:

Beispiel: `ADD Ri,Rj,Rk`

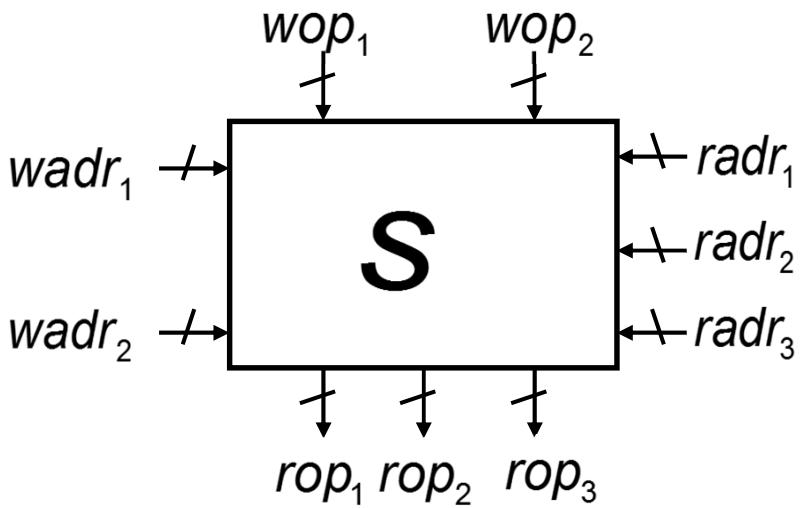
lesen: *pc, Reg[j], Reg[k]*

schreiben: *pc, Reg[i]*

Multiportspeicher ff

Wir bräuchten zur direkten Implementierung der Maschine also einen Multiportspeicher, konkret einen

„3-Lese-2-Schreib-Speicher“:



$rop_i := s[u_n(radr_i)]$

$s[u_n(wadr_i)] := wop_i$

Es seien

$wadr_i$: Adressen der Schreiboperanden wop_i, als unsigned Zahlen.

$radr_i$: Adressen der Lese-operanden rop_i, als unsigned Zahlen.

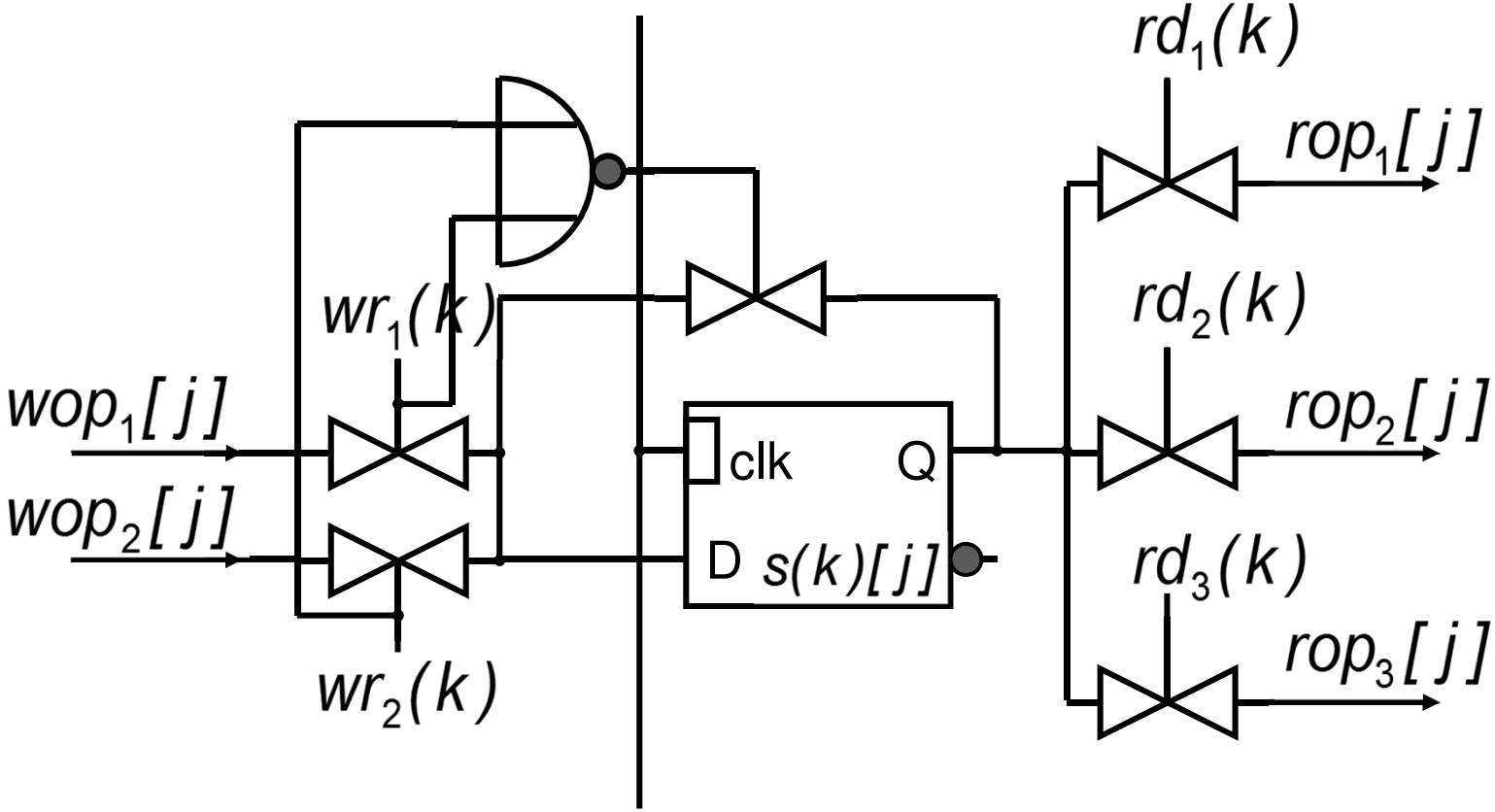
Multiportspeicher ff

Realisierung:

Führe die Schreibleitungen $wop_i[j]$ für $i=1,2$ an alle Latches, die in einer Zelle k das Bit j mit $j=0,\ldots,n-1$ speichern, und lege die Leseleitungen $rop_i[j]$ an deren Ausgänge.

Wir müssen nun nur noch entscheiden, ob der Eingang oder Ausgang mit einer Schreib-, Leseleitung verbunden werden soll. Dazu nehmen wir an, dass wir zu jeder Zelle k eine Auswahlleitung $wr_i[k]$ und $rd_i[k]$ haben, die 1 sind, genau dann, wenn die Zelle k vom i-ten Schreib- oder Leseoperand angesprochen werden soll. Dies führt zu folgender Verschaltung für ein Bit j in einer Zelle k:

Multiportspeicher: Grundzelle



Schreibzugriffe an Port 1 und 2 müssen stets auf verschiedenen Zellen stattfinden, sonst hat man am Eingang der Zelle ggf. einen Kurzschluß!

Der Decoder

Die Schalter unserer Multiportzelle werden angesteuert mit

$$wr_i(k) = 1 \Leftrightarrow u_n(wadr_i) = k \qquad rd_i(k) = 1 \Leftrightarrow u_n(radr_i) = k$$

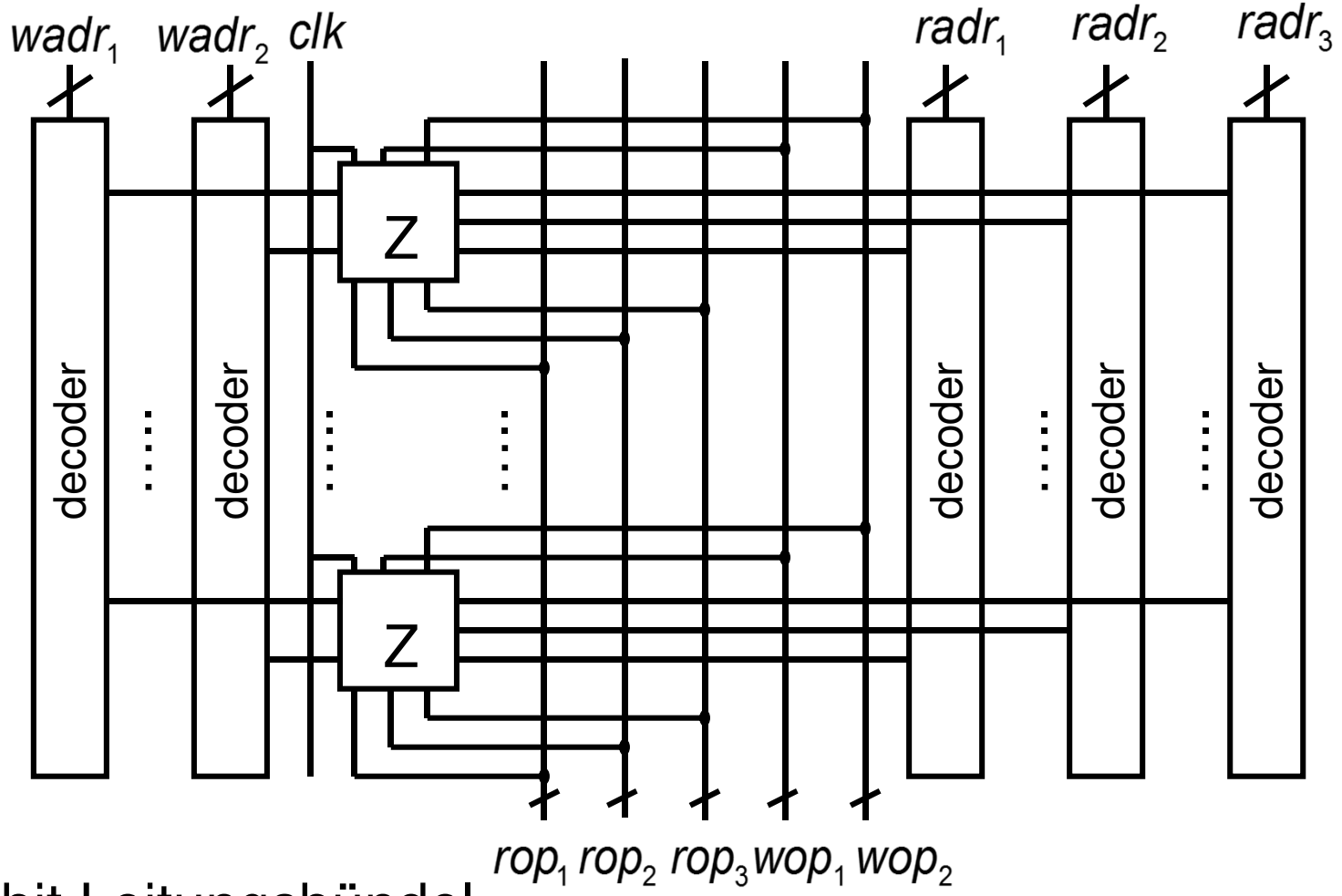
Wir müssen also den Schaltkreis aus Kapitel 2 nehmen, der folgende Dekodierfunktion realisiert:

$$decode_m : B^n \mapsto B^m$$

mit $decode_m(a_0,\ldots,a_{n-1}) = (y_0,\ldots,y_{m-1}) \Leftrightarrow \forall i : y_i = (u_n(a) = i)$

Multiportspeicher -- Übersicht

Der Speicher hätte also folgende Struktur:



$\not\vdash$: *n*-bit Leitungsbündel

Multiportspeicher ff

Diese Anordnung müsste planar auf einem oder mehreren Chips untergebracht werden. Würde man dies wie oben skizziert tun, hätte man bei Speichergröße *k* Worte eine Höhe ~ *k* und eine Breite ~*log k*. Ein unmögliches Streckungsverhältnis!

Die Realisierung einer einzelnen Zelle, die im Aufwand ja am meisten, nämlich mit dem Faktor „Wortbreite“ zu Buche schlägt, ist sehr teuer: Wir benötigen neben dem Latch auch noch ein Gatter und 6 Schalter pro Bit.

Es ist also wenig sinnvoll, solche Speicherstrukturen für wirklich große Speicher zu benutzen:

- hohe Last → lange Schaltzeiten (Fanout, Verdrahtung)
- hohe Kosten pro Bit

Speicherkonzept der Maschine

Deshalb benutzen wir folgendes Speicherkonzept für unsere Maschine:

- einen sehr kleinen Multiport Speicher für 3 Adressbefehle,
- getrennte Speicher für Zellen mit Sonderaufgaben und
- große Single Port Speicher für Transporte (Load/Store).

Die Zellen des kleinen Multiportspeichers nennen wir **GP-Register** (general purpose), die Zellen mit Sonder-aufgaben, wie z.B. der Programmzähler, nennen wir **SP-Register** (special purpose).

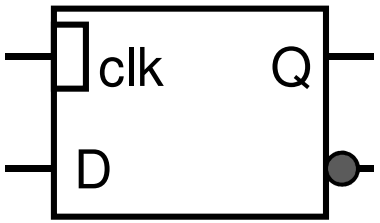
Der Haupteil des Speichers soll nun ein Single Port Speicher sein, der in einem Zyklus entweder ausgelesen oder beschrieben werden kann. Einen solchen Speicher nennen wir auch **RAM** (random access memory).

Struktur eines RAM

Wir wollen zunächst überlegen, wie wir einen RAM Speicher für ein Bit Wortbreite aufbauen können. Andere Wortbreiten erhält man dann durch Parallelschalten von solchen 1-Bit Speicherbausteinen.

- **Die Zelle:**

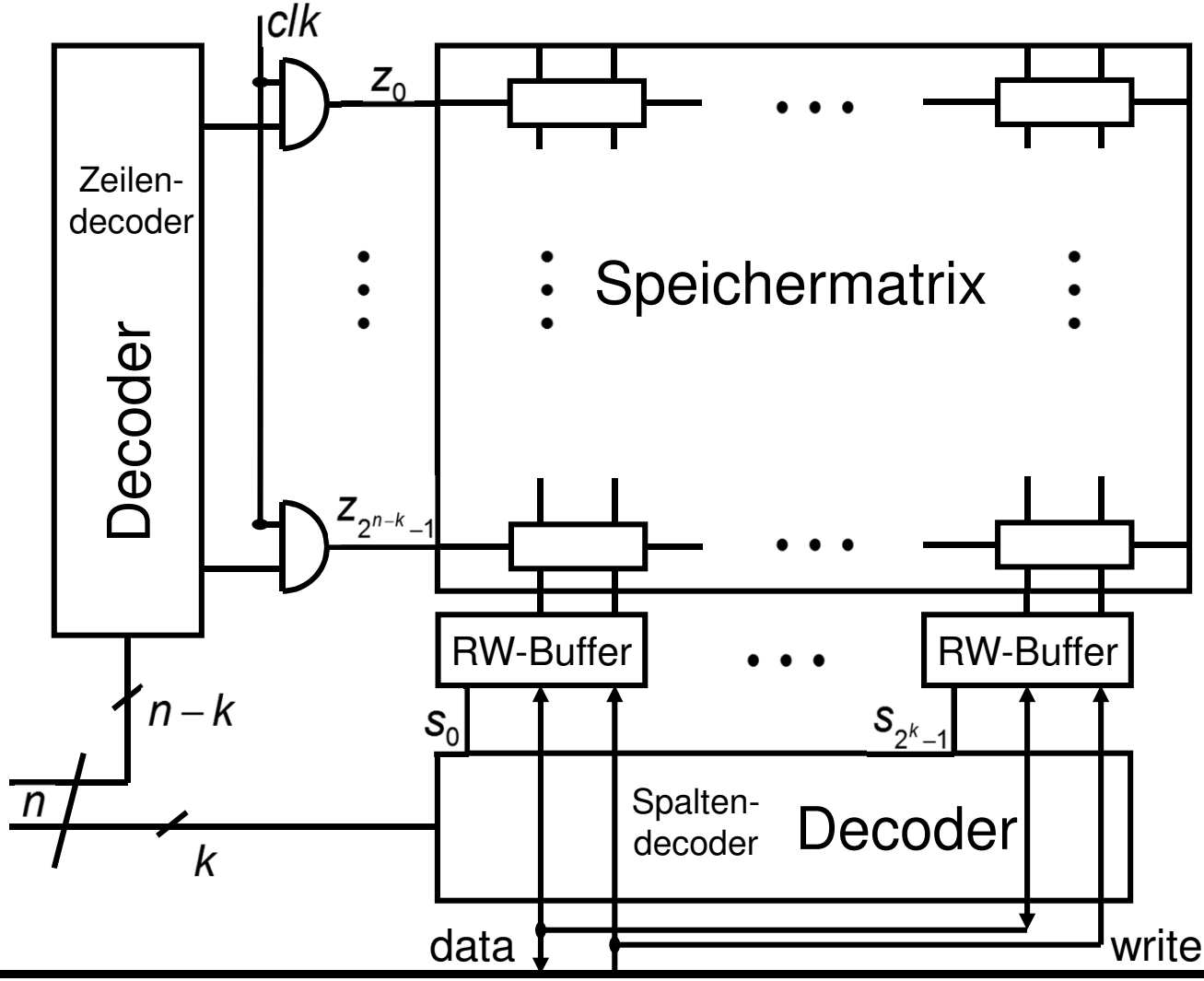
  Als Zelle nehmen wir einfach ein statisches oder dynamisches D-Latch:



Mit dieser Zelle läßt sich der Speicher wie folgt aufbauen:

Struktur eines RAM ff

- **Grobarchitektur eines RAM**

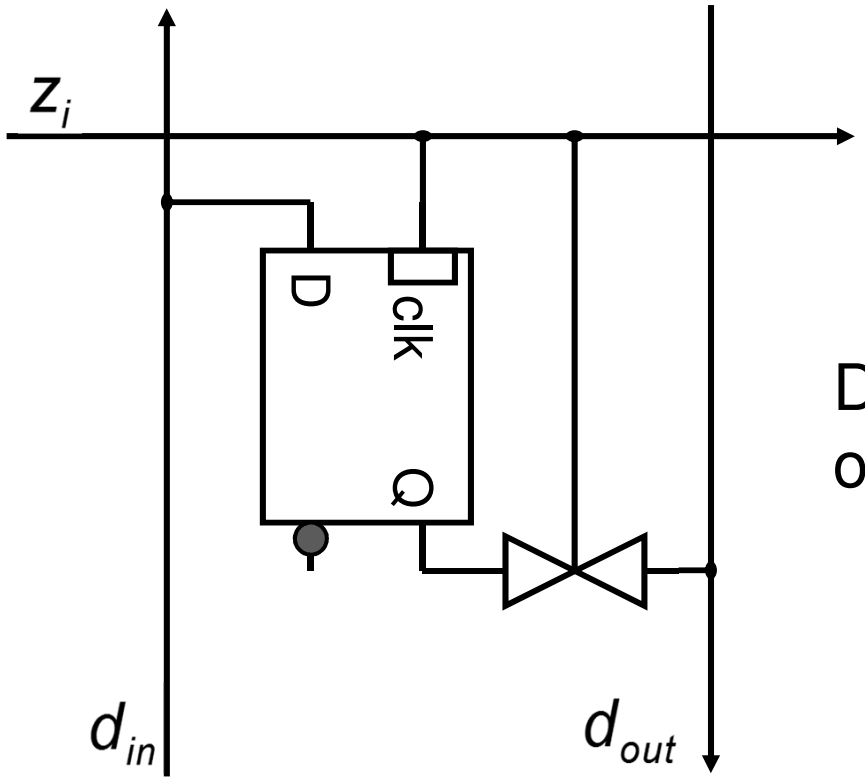


Struktur eines RAM ff

Die Aufspaltung des Decoders in Zeilen- und Spalten-decoder dient dazu, das Streckungsverhältnis des Speichers zu balancieren (Höhe/Breite ~ 1). Ferner werden die Spaltendecodersignale nur zu den RW-Buffern (Read/-Write) geführt, die dann entscheiden, welche Spalte auf die bidirektionale Datenleitung gekoppelt wird. In jedem Schreib- oder Lesezyklus werden also alle Spalten über die RW-Buffer geführt, und die durch die decodierte Zeile angewählten Zellen parallel ausgelesen und ggf. wieder aufgefrischt (wichtig bei DRAMs).

Nun zu den Komponenten:

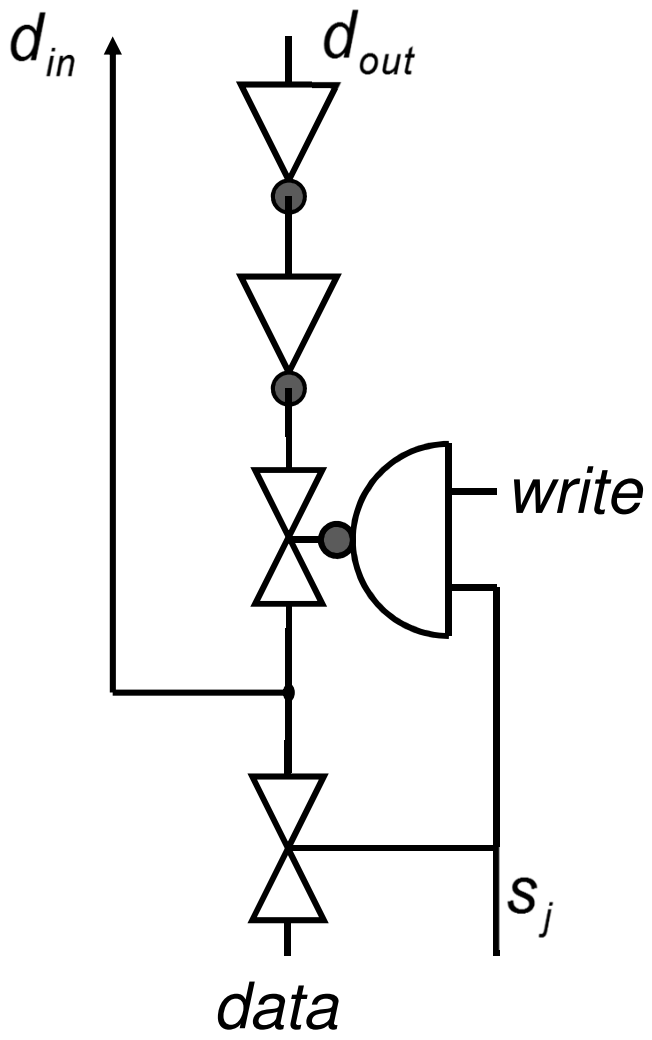
Zellen der Speichermatrix



D-Latch statisch (SRAM)
oder dynamisch (DRAM)

Das Zeilensignal des Decoders steuert die Übernahme des neuen Zustands von d_{in}. Der aktuelle Inhalt wird über den RW-Buffer von d_{out} nach d_{in} durchgeschleift und wieder aufgefrischt (lesen) oder von d_{in} überschrieben (schreiben).

Der Read/Write-Buffer



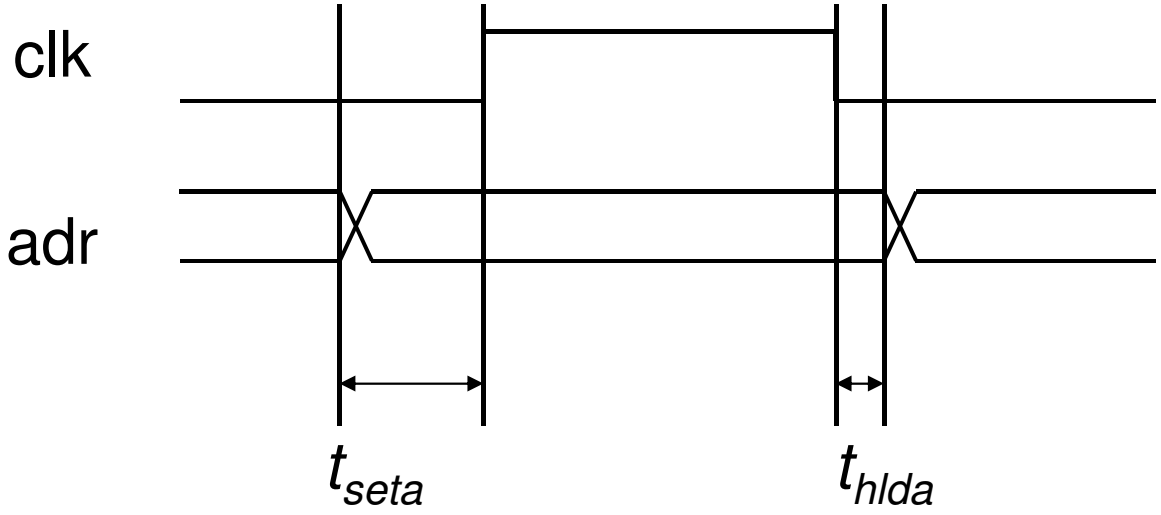
Lesezugriff (write=0):
Inhalt der in der Zeile ausgewählten Zelle wird über die Doppelinverter verstärkt, zurückgeschickt und ausgegeben, falls die Spalte selektiert ist.

Schreibzugriff (write=1):
Inhalt der in der Zeile ausgewählten Zelle wird über die Doppelinverter verstärkt, zurückgeschickt ($s_j = 0$).
Bei ausgewählter Spalte wird aber überschrieben.

Zeitbedingungen

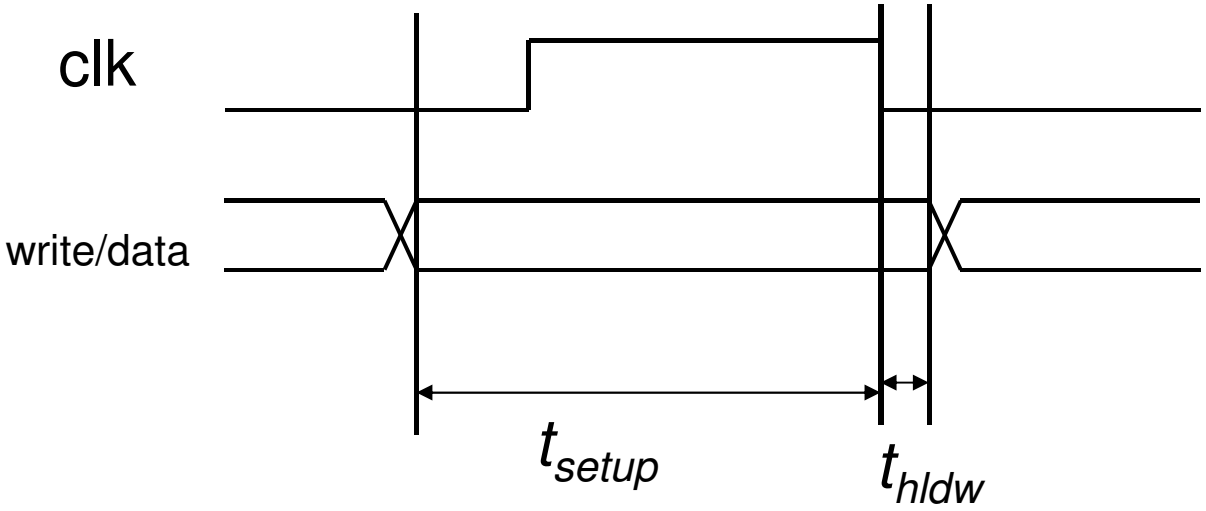
Damit Zugriffe nun korrekt ablaufen muss man Zeitbedingungen einhalten, die den Signallaufzeiten im Speicher Rechnung tragen:

• Dekoderlaufzeiten: Adressen müssen t_{seta} vor dem Clock stabil sein und bleiben, damit die Spalten-/Zeilenauswahl vor Beginn des Zugriffs stabil sind.



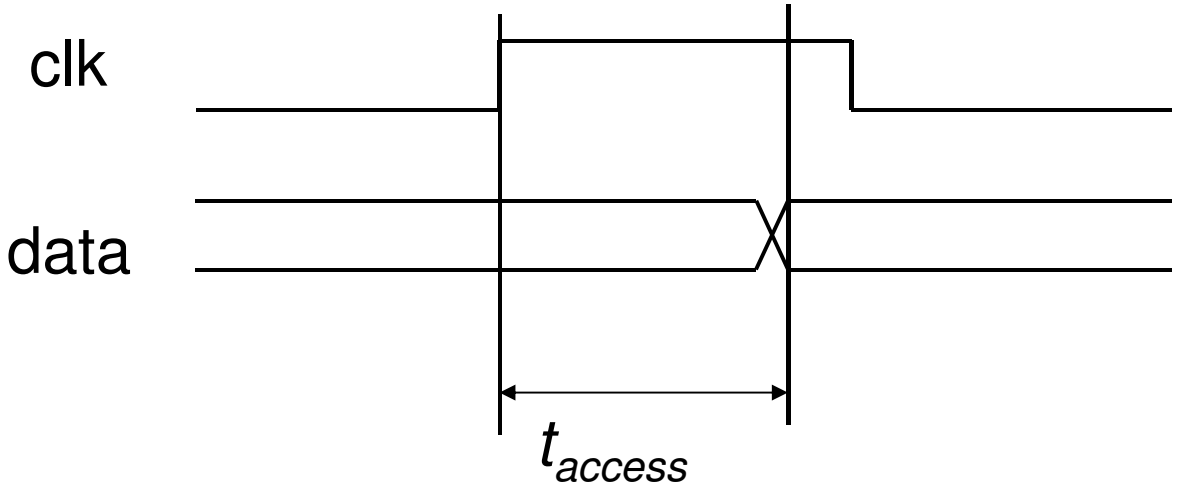
Zeitbedingungen ff

• Matrix- und RW-Bufferlaufzeit: das Kontrollsignal zum Schreibzugriff (write) (sowie das Datensignal (data), falls write=1,) muss t_{setup} vor dem Ende der aktiven Phase des clocks stabil sein und bleiben. Sicher wäre folgende Wahl: $t_{setw} = Pulsbreite + t_{seta}$. Es geht aber je nach Speicher auch kürzer.



Zeitbedingungen ff

• Matrix- und RW-Bufferlaufzeit ff: das Datensignal wird beim Lesezugriff erst t_{access} nach dem Beginn der aktiven Phase des Clocks stabil. Die Clockbreite muss mindestens so breit wie die Laufzeit der Rückführung sein.



Es gibt je nach weiterer interner Unterstützung in der Schaltung Möglichkeiten, die Zugriffszeiten bei bestimmten Zugriffen zu verringern (s. z.B. EDO RAMs).

Speichertypen

Anmerkungen:

• Man kann die Schaltung auch auf größere Wortbreiten auslegen. Aber die Flexibilität bei Konstruktion von Speicherboards sinkt dadurch (verschiedene Wortbreiten, Fehlerkorrekturbits etc.).

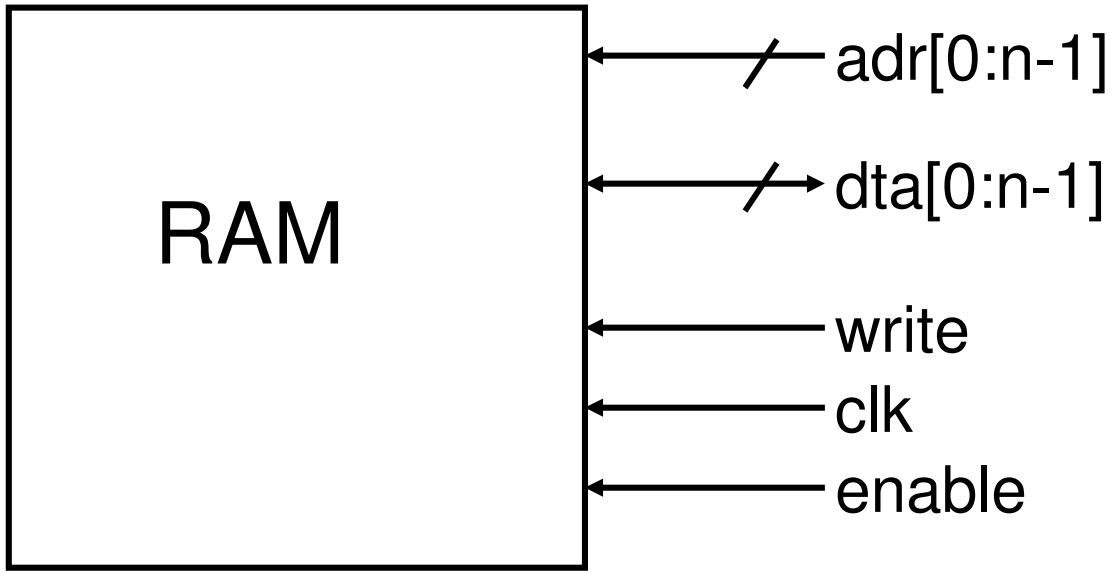
• Ersetzt man die Latches durch Konstanten und verzichtet auf die Schreiblogik, erhält man **ROM**s (Read Only Memories).

• Benutzt man Zellen, die durch eine höhere Spannung permanent auf einen Wert programmiert werden können erhält man **EPROM**s (Electrically Programmable Read Only Memories).

Der Speicher

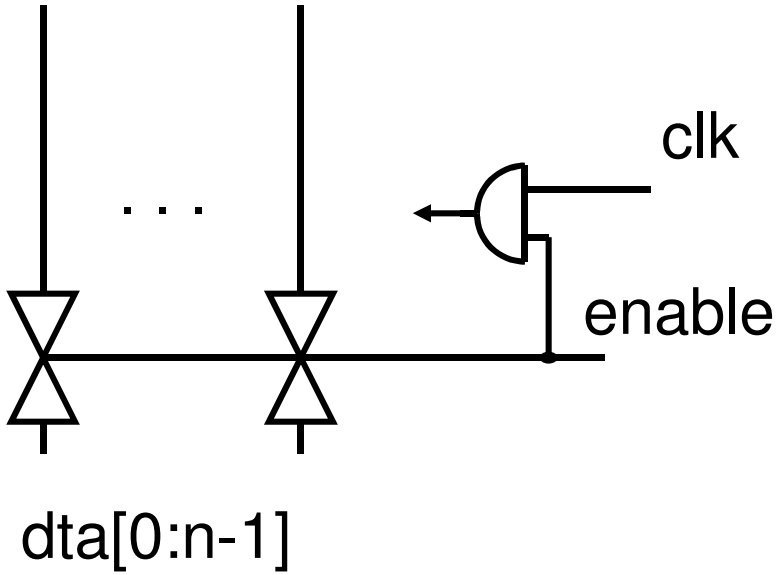
Wir wollen nun annehmen, dass wir Speicherbausteine mit 1-Bit Wortbreite haben, und skizzieren, wie wir damit den Hauptspeicher einer Maschine konstruieren:

Der "Haupt"-speicher kommuniziere mit der Maschine über folgenden einfachen Bus:



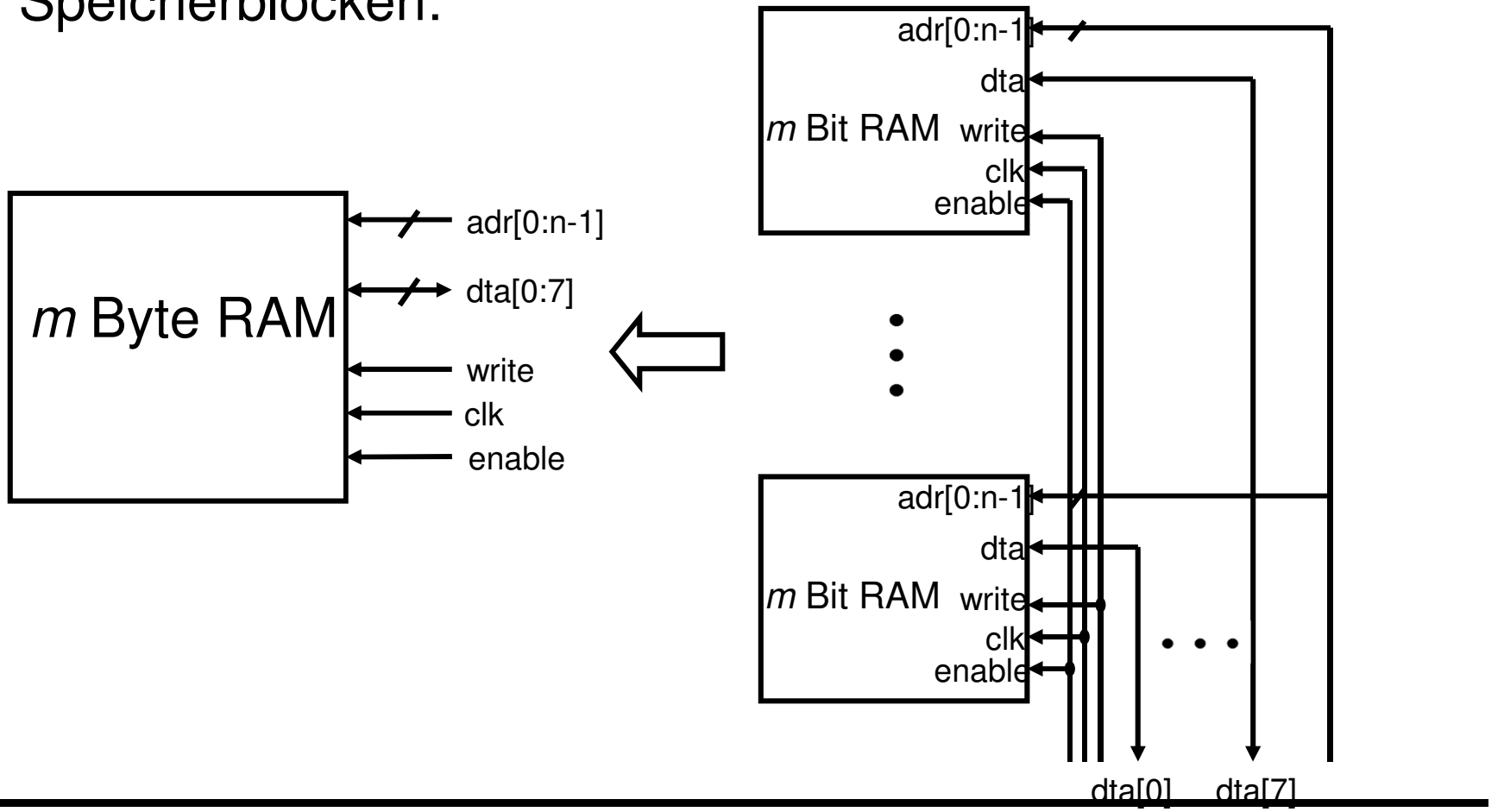
Der Speicher ff

Hinzugenommen haben wir eine enable Leitung, die dazu benutzt werden kann, den Speicher überhaupt erst zum Arbeiten zu ermächtigen. Sie trennt im Zustand *0* die Datenleitungen des Speichers vom Bus, und stellt den Arbeitstakt ggf. ab:



Der Speicher ff

Einen byteorientierten Speicherblock erhalten wir nun durch Parallelschalten von 8 bitorientierten Speicherblöcken:



Der Speicher ff

Der Speicher selbst soll nun Wortzugriffe, aber auch Teilwortzugriffe unterstützen, wobei Byte die kleinste adressierbare Einheit sei. Je nach Vereinbarung ergeben sich dadurch mehr oder weniger einfache Schaltungen. Wir unterscheiden

• aligned access, d.h. Teilworte mit *k* Bytes *(k=1,2,4,…,n/8)* Länge beginnen stets auf einer Adresse, die ein Vielfaches von *k* ist.

• misaligned access, d.h. Teilworte beginnen an beliebigen Adressen *a* und setzen sich dann aus den Bytes *a+0, a+1, …, a+k-1* zusammen.

Teilwortzugriffe

Bei einem Teilwortzugriff möchte man ein Register, das ja Wortlänge viele Bits hat, nur mit einem Teilwort laden. Dazu muss vereinbart werden:

- Wie wird das Teilwort im Register abgelegt?
- Welcher Teil des Registers wird beim Schreibzugriff weggeschrieben?

Man möchte Teilworte natürlich auch als (kleine) Zahlen interpretieren können. Deswegen soll ein Teilwort stets auf den am wenigsten signifikanten Stellen platziert werden (d.h. rechtsbündig). Allerdings hat man dann immer noch die folgenden zwei Ablegungsvarianten:

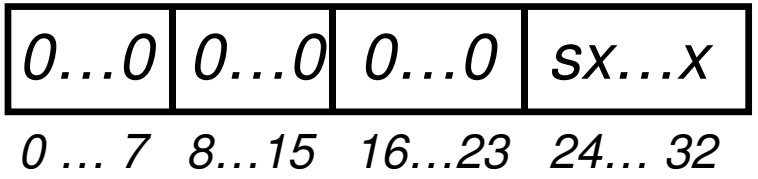
- **Unsigned**: signifikante Stellen mit *0* auffüllen.
- **Sign extended**: signifikante Stellen mit Vorzeichen füllen.

Teilwortzugriffe -- Beispiele

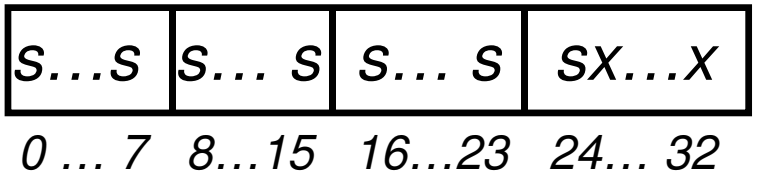
Big Endian

Byte *sx...x*

unsigned

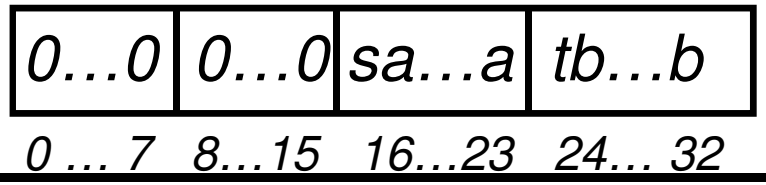


signexd



Short i:*sa…a i+1:tb…b*

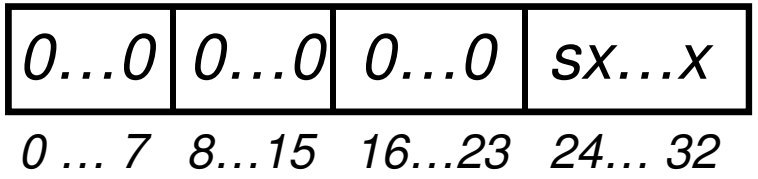
unsigned



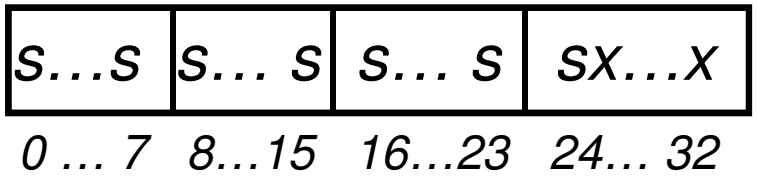
Little Endian

Byte *sx...x*

unsigned

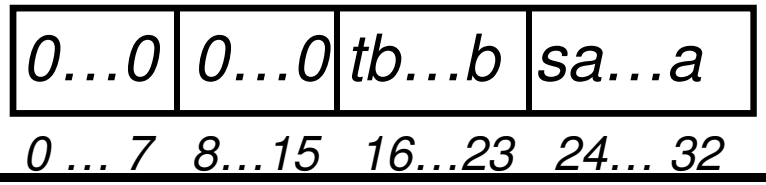


signexd



Short i:*sa…a i+1:tb…b*

unsigned



Speicher ff

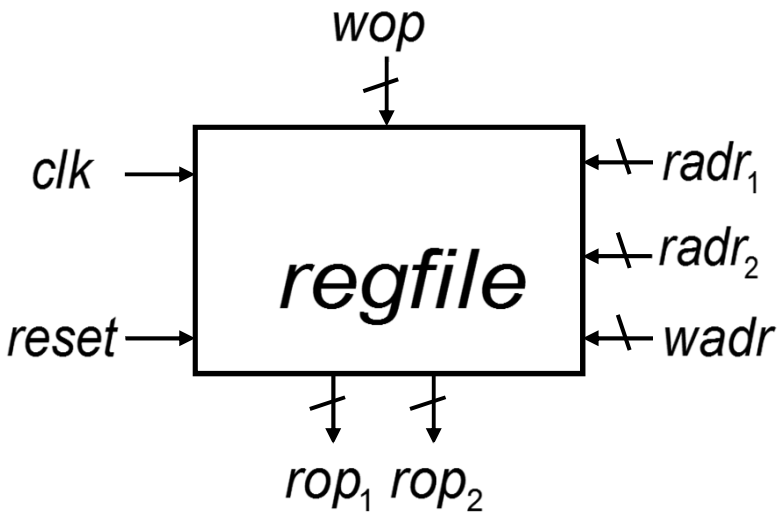
Wir vereinbaren weiter:

- Zwischen Hauptspeicher und Maschine werden stets nur ganze Worte, die das adressierte Teilwort enthalten, übertragen. Der Prozessor sorgt für die richtige Einbettung je nach Befehl. Der Speicher sorgt bei Schreibzugriffen dafür, dass nur das adressierte Teilwort überschrieben wird.
- Aligned access: Das übertragene Wort legen wir stets einfach ab *a = u(adr[0:n-k],0..0)* (*k* - Wortlänge in Bytes), oder
- Misaligned access: Das übertragene Wort beginne stets mit dem Teilwort.

Problem: Bei misaligned access muss man ggf. auch das Inkrement von *adr[0:n-k]* berechnen (kostet Zeit!)

Die Registerbank

Da unsere Maschine relativ wenige Register hat, werden wir diesen Speicher, wie eingangs beschrieben, durch einen (sehr schnellen, da sehr kleinen) Multiportspeicher ersetzen. Den PC realisieren wir getrennt, d.h. wir benötigen lediglich einen 1-Schreib-2-Lese-Multiportspeicher, den wir aber mit 0 initialisieren können sollten:



Die Adressleitungen *xadr* haben die Breite *log*(#Register)