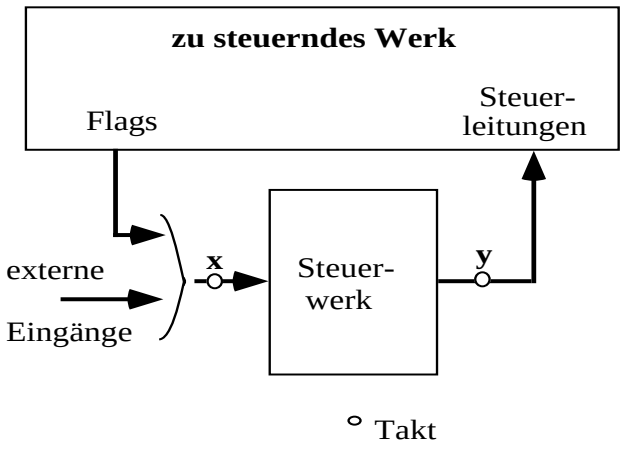
Kapitel 3 Steuerungstypen

Es wird die Erzeugung der Steuersignale für das Rechenwerk betrachtet.



Das zu steuernde Werk erwartet Steuersignale und stellt Flags für Abfragen bereit. Das Steuerwerk hat als Eingänge diese Flags und zusätzliche Signale von außen, die zusammen den Eingangsvektor **x** bilden. Es stellt dem zu steuernden Werk den Vektor **y** der Steuerleitungen zur Verfügung.

Ein externes Taktsignal schaltet die Steuervektoren nacheinander auf das zu steuernde Werk auf. Das Steuerwerk durchläuft dabei intern verschiedene Zustände z; zur Zeit t_n befinde es sich im Zustand $z^{(n)}$, zum Zeitpunkt t_{n+1} werde der Zustand $z^{(n+1)}$ angenommen. Der externe Takt markiert diese Zeitpunkte. Es muß sichergestellt sein, daß der Eingang **x** sich in einem Zeitintervall Δt um den Takt herum nicht ändert, um Hazards auszuschließen.

3.1. Codierte Steuerungen

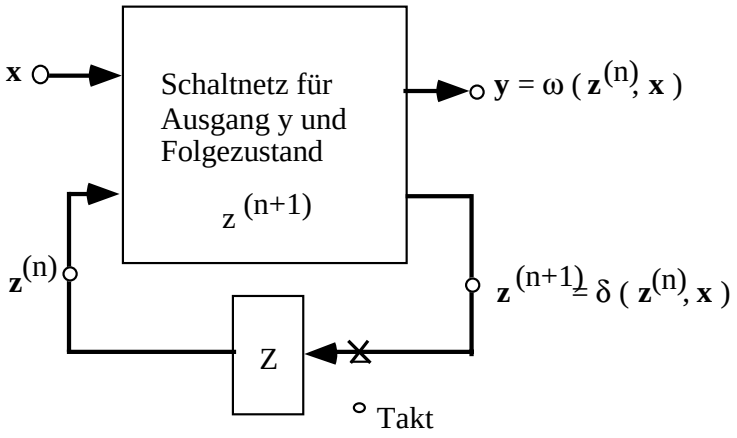
Das zu steuernde System habe m Zustände mit

$2^{p-1} < m \leq 2^p$.

Es werden die m Zustände in p Bit codiert.

3.1.1. Mealy-Automat

Der allgemeinste Fall einer codierten Steuerung ist ein Mealy-Automat: Folgezustand und Steuerleitungen sind Funktionen der Eingänge und des momentanen Status.



Beispiel DLX:

Man hat zunächst 45 Zustände, die in 6 Bit codiert werden:

- 5 auf der obersten Ebene
- 12 für Datentransportbefehle
- 11 für ALU-Operationen
- 5 für SET-Befehle
- 3 für Verzweigungen
- 9 für Sprünge

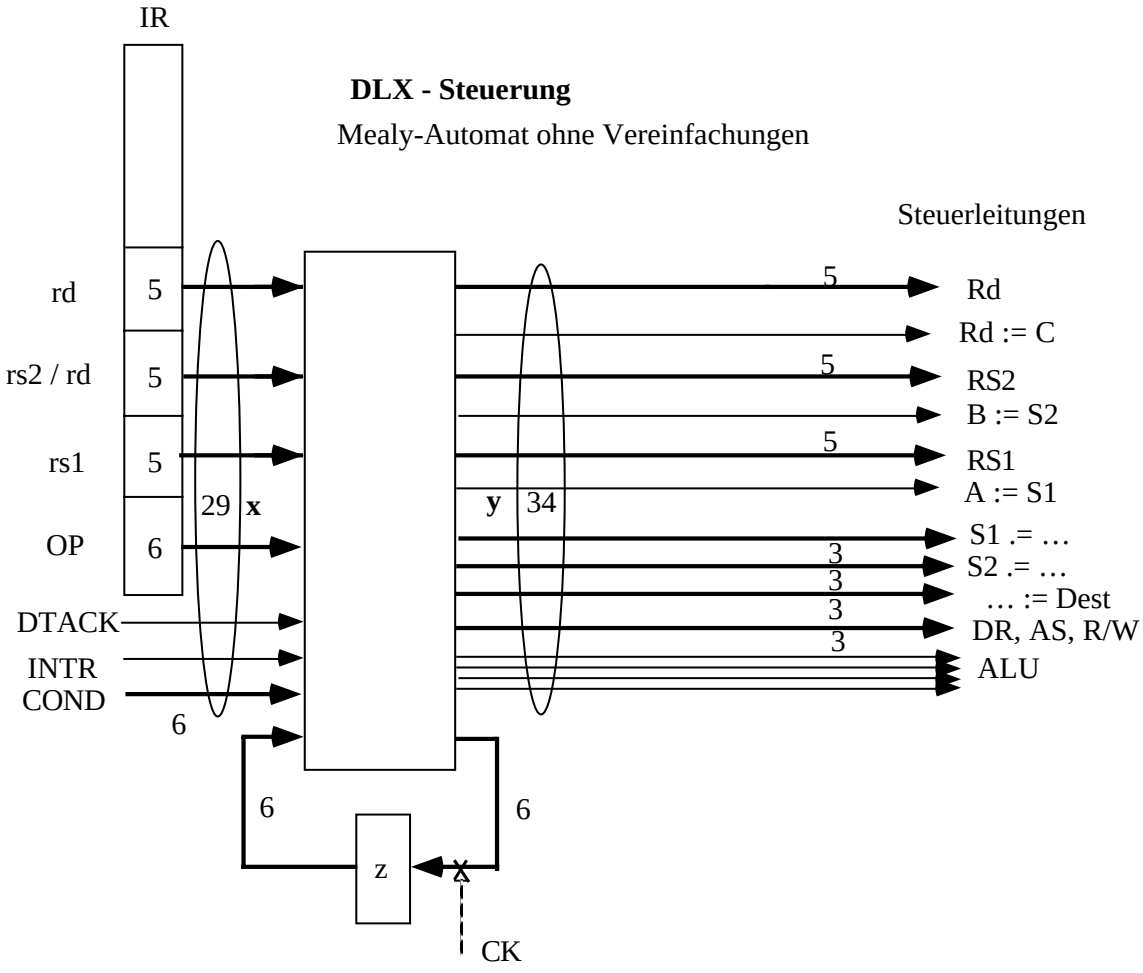
und 29 Eingänge **x** :

- 15 Bit für die Kennzeichnung der Register,
- 6 Bit aus dem OP-Code,
- 2 Bit für Signale INTR und DTACK von außen
- 6 Bit COND-Signal aus dem Rechenwerk).

Die 34 Steuerleitungen y teilen sich auf in

- 15 Steuerleitungen für die Adressierung der Register,
- 3 Steuerleitungen zur Quellenauswahl für die Sammelleitung S1,
- 3 für Sammelleitung S2
- 3 Steuerleitungen zum Festlegen der Senken von Dest bzw. Lesen/Schreiben und Übernahme in MDR oder IR.
- 3 Steuerleitungen für die Übernahme in die Zwischenregister A, B oder Rd
- 4 Funktionsleitungen um die Verknüpfungen in der ALU festzulegen und die Quellen der Sammelleitng Dest zu bestimmen.
- 3 Steuerleitungen für den Speicher : DR, AS und R/W

Das zugehörige Schaltnetz ist nicht handhabbar: 35 Eingänge, 40 Ausgänge!



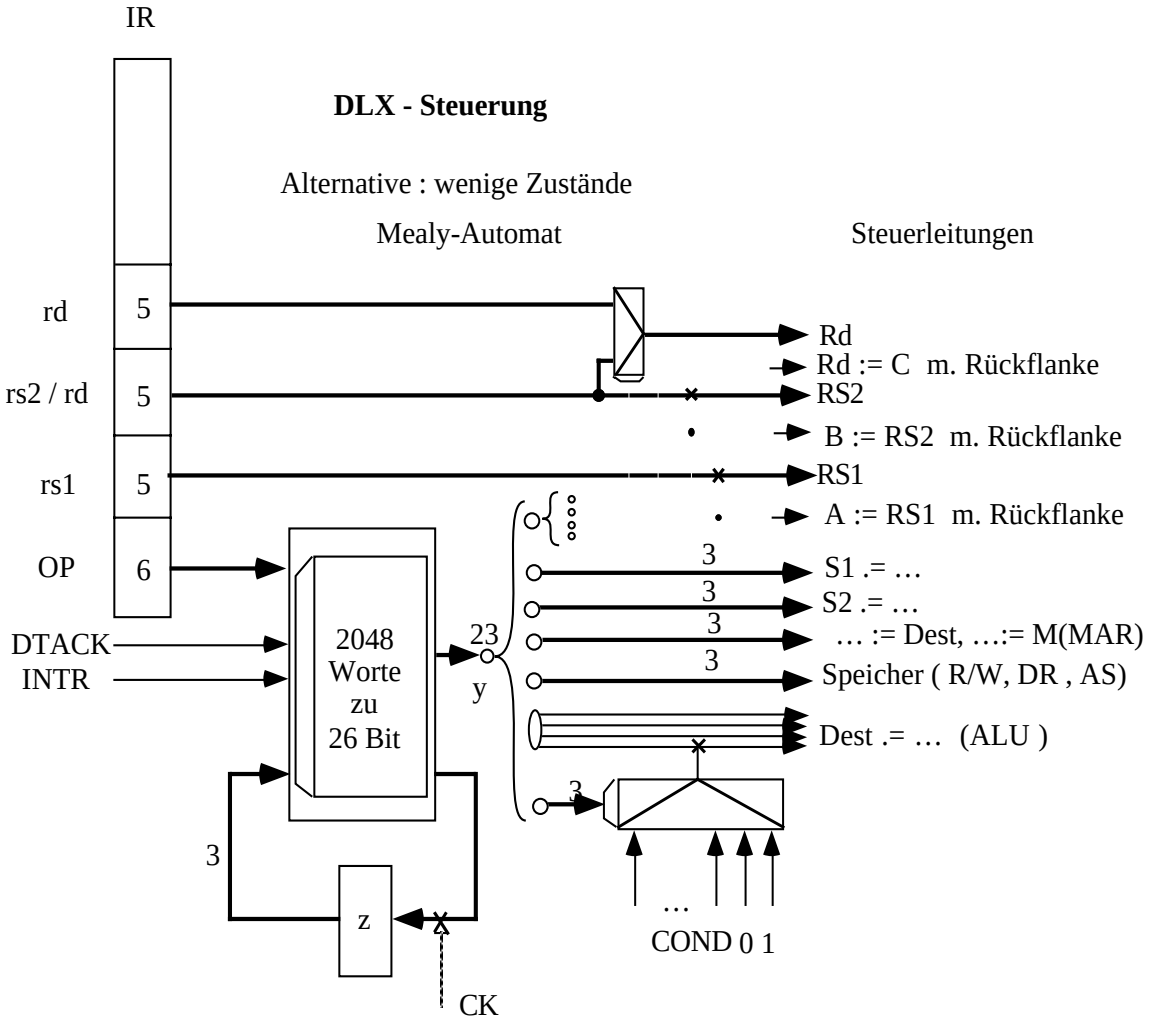
Man wird Vereinfachungen einführen:

- Die Adressleitungen für das Registerfile werden direkt durchgeschaltet und mit Durchschaltsignalen freigegeben. Dazu werden die Übernahmesignale für A,B und Rd benutzt und mit einem zusätzlichen Signal die passende Registeradresse auf Rd durchgeschaltet.
- Die Zahl der Zustände wird auf 5 begrenzt:
  - holen, dekodieren, ausführen, komplettieren und rückschreiben.

Es reichen 3 Bit zur Codierung.

- Eine der 6 Bedingungen aus COND oder konstant 0 oder 1 werden mit drei Bit in einem Multiplexer ausgewählt und mit einer der Steuerleitungen der ALU verrechnet.

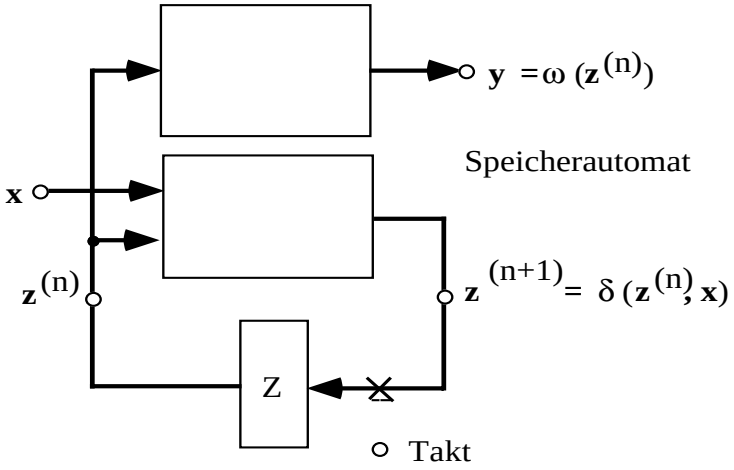
Man erhält ein Schaltnetz mit 11 Eingängen und 23 + 3 = 26 Ausgängen: ein Speicher mit 2048 Worten zu 4 Byte reicht aus!

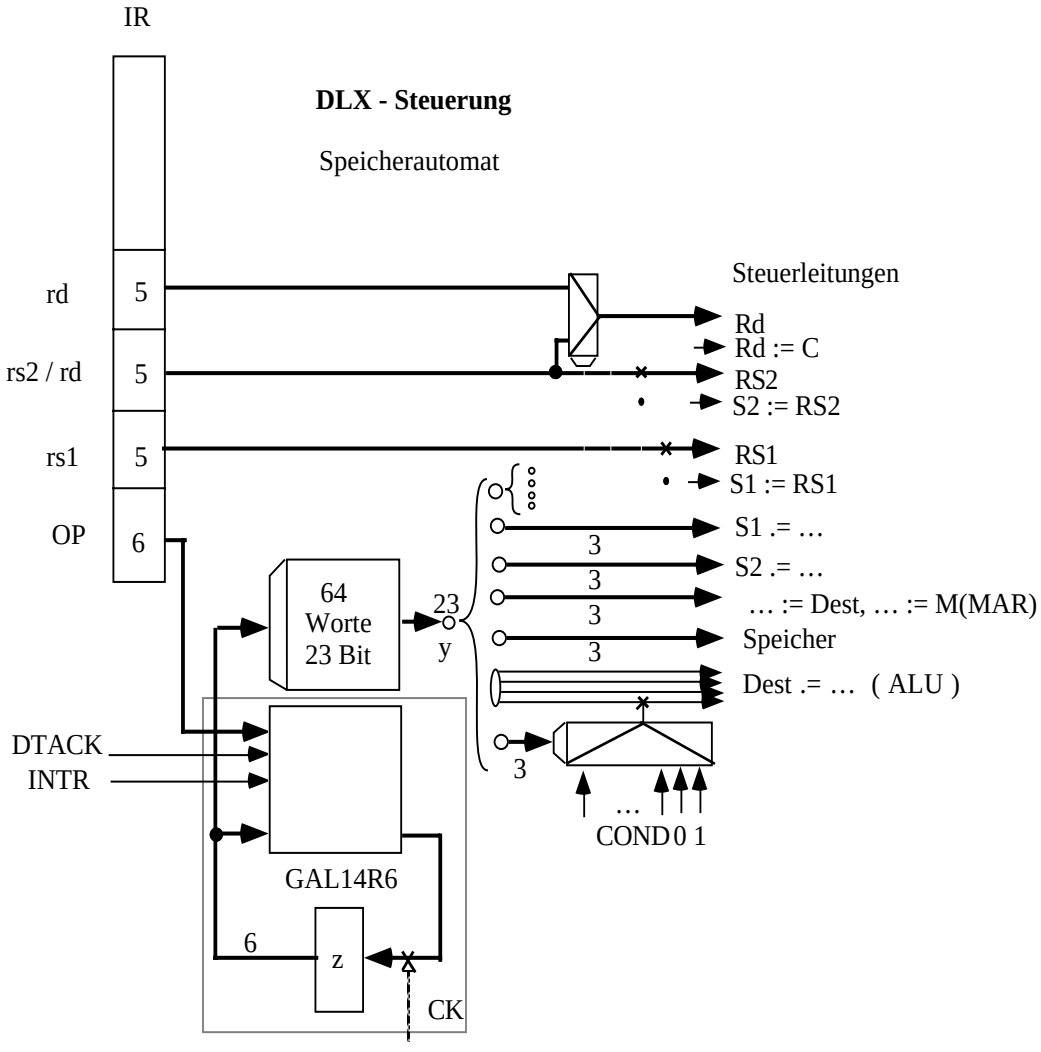


3.1.2. Speicherautomat

Hier ist der Ausgang nur eine Funktion des Zustandes $z^{(n)}$.

Im Beispiel der DLX-Maschine wird dann allerdings die Zahl der Zustände in 6 Bit zu codieren sein und damit einerseits der Speicher $y = \omega (z^{(n)})$ einfach, dafür aber das Schaltnetz für die Berechnung des Folgezustandes komplexer mit 14 Eingängen.





Im Prinzip können beide, der Speicher und das Schaltnetz, durch GAL's (gate array logic) realisiert werden. Wie oben gezeigt, kann das Schaltnetz und das Zustandsregister durch ein einziges GAL mit 14 Eingängen und 6 Ausgängen realisiert werden. Die Methoden zur Vereinfachung der Schaltnetze sind hier nicht Gegenstand der Untersuchung.

3.2. Mikroprogrammierte Steuerungen

3.2.1. Horizontale Mikroprogrammierung

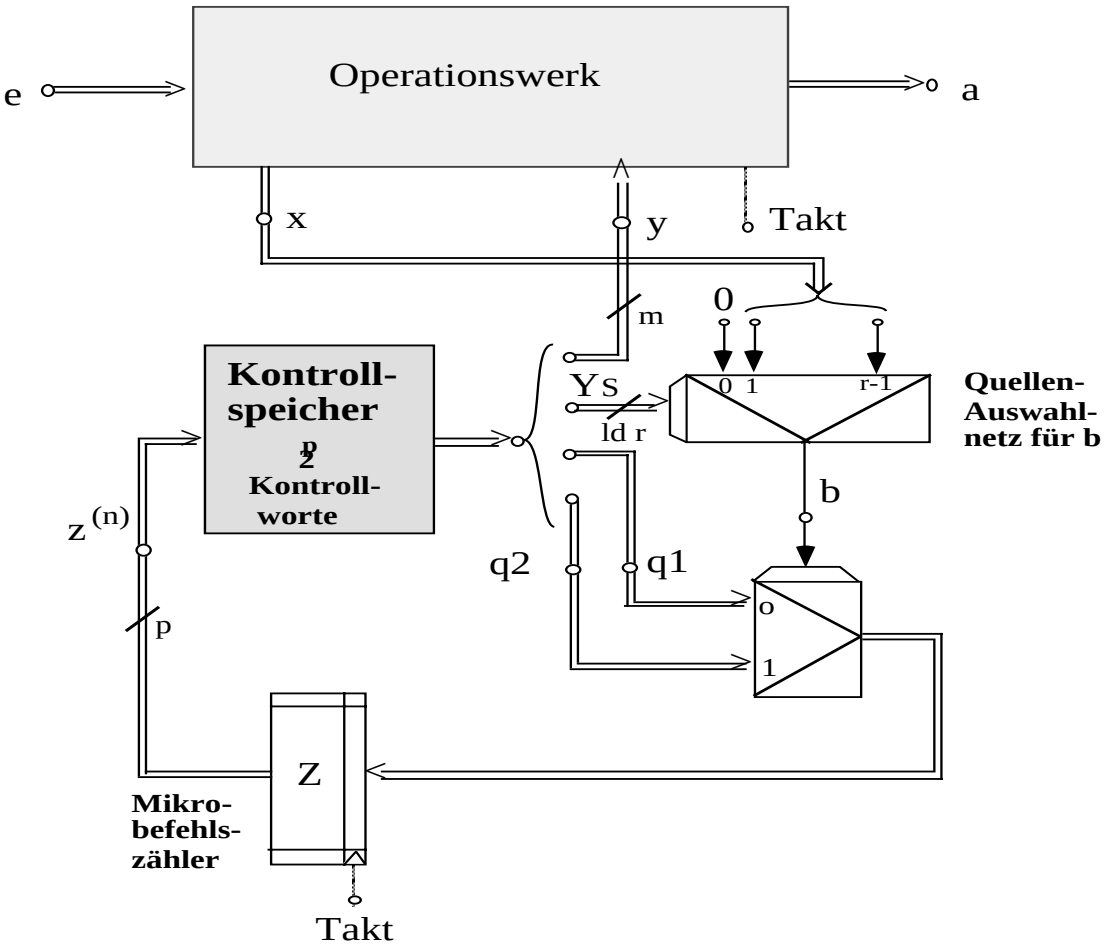
In jedem Schritt werden **alle** in einem Takt parallel ausführbaren Aktionen codiert:

- welche Steuerleitungen für das Operationswerk aktiv sein sollen;
- welche Bedingungsleitung ausgewählt werden soll;
- wohin gesprungen werden soll, wenn die Bedingung erfüllt / nicht erfüllt ist.

Diese Informationen werden einem Kontrollspeicher entnommen, der mit dem Zustand adressiert ist.

Man hat damit folgende Spezialisierung gemacht:

- ein Speicherautomat : $\mathbf{y} = \omega(\mathbf{z}^{(n)})$
- der Folgezustand $\mathbf{z}^{(n+1)}$ ergibt sich als $\mathbf{z}^{(n+1)}$.= if b then q_1 else q_2
  mit b .= case ys of **x** ## 0 (Multiplexer)



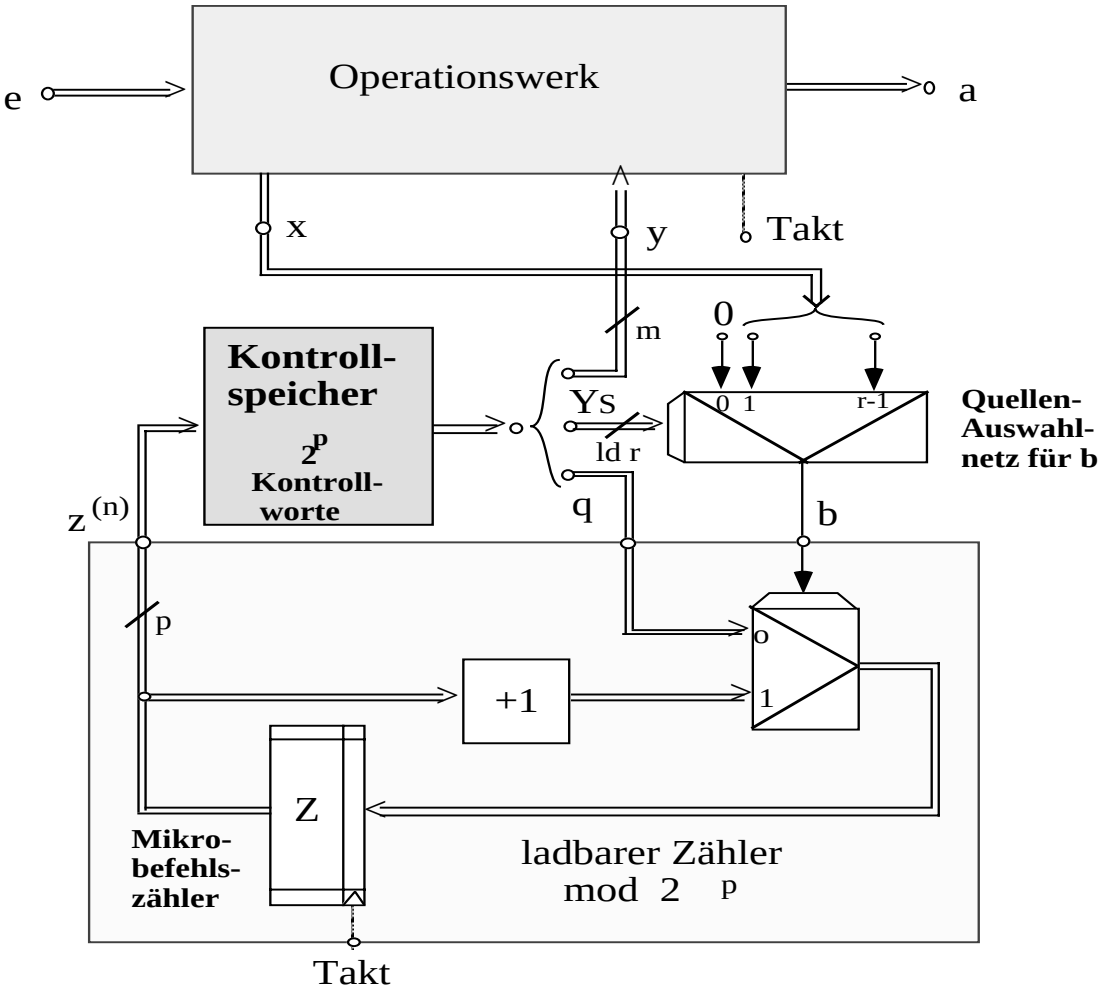
Der Fall, daß keine Bedingungsleitung ausgewählt wird, d. h. der Folgezustand unabhängig von **x** ist, wird durch die " 0" berücksichtigt.

Der Kontrollspeicher hat dann 2^p Worte von (m + 2p + r) Bit mit m = Anzahl der Kom-

ponenten von **y** = (y_1, ..., y_n); p = Anzahl der Komponenten von **z** = (z_{p-1}, ..., z_0) und r = ld(n+1) für **x** = (x_{n-1}, ..., x_0) konkateniert mit "0".
b hat die Funktion eines Flag, das aus **x** erzeugt wird, indem man mit ys **eine** Leitung von **x** auswählt.

In vielen Fällen kann man die Schaltung vereinfachen, indem man implizit einen Folgezustand $z^{(n+1)} = z^{(n)} + 1$ annimmt:

$z^{(n+1)}$.= if b then ($z^{(n)}$ +1) else q .



Technisch hat man einen ladbaren Zähler für 2^p vor sich.

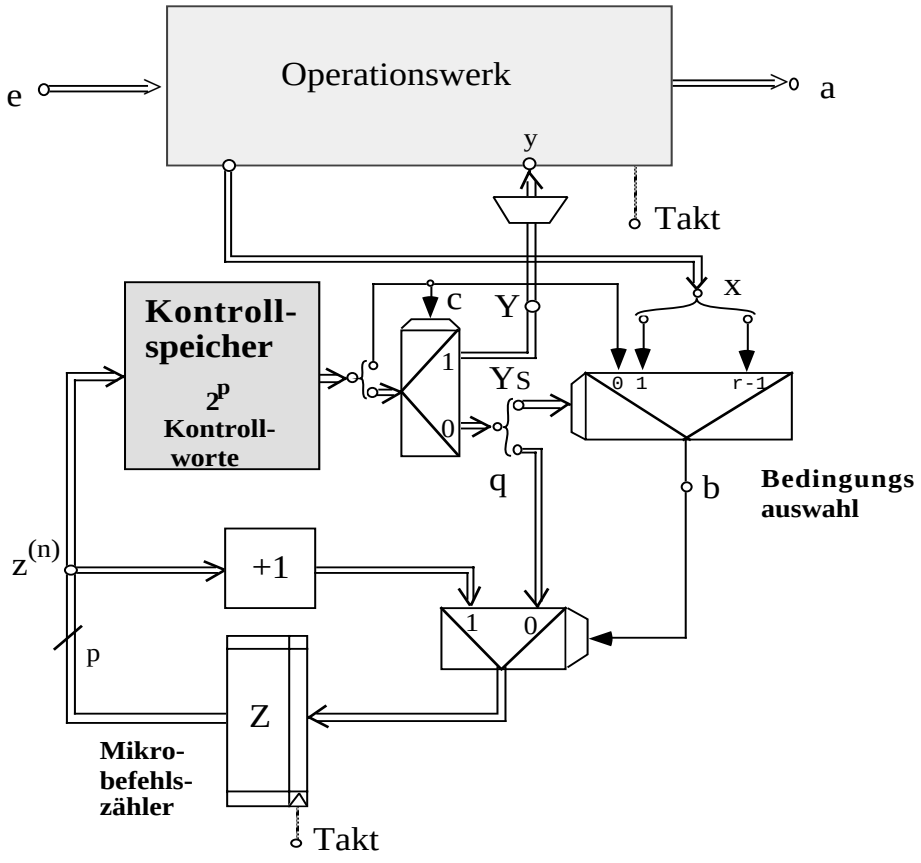
Die Wortbreite des Kontrollspeichers ist im allgemeinen relativ groß, dafür kann alles, was parallel gemacht werden kann, auch parallel ausgeführt werden.

3.2.2. Vertikale Mikroprogrammierung

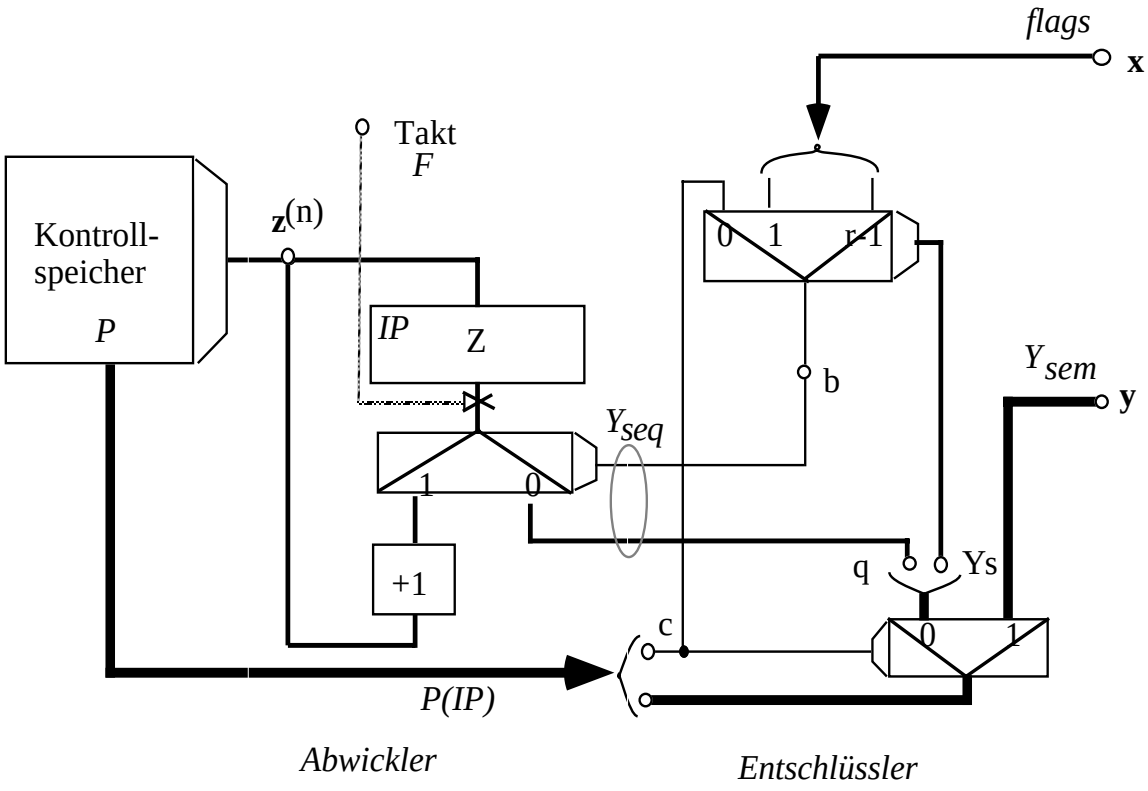
In vielen Fällen kann man auf die Parallelausführung von Operationen im OPW und Sprüngen im Kontrollspeicher verzichten: man hat deutlich getrennt datenbearbeitende Befehle und Sprungbefehle.

Man hat zwei Befehlsformate der Worte im Kontrollspeicher

und kann die Steuerung entsprechend organisieren:
ein Bit c schaltet entweder Daten auf **y** durch (dann darf **y** = (0, ..., 0) keine Aktion im OPW auslösen) und inkrementiert den Zustand: z := z + 1 oder bei c = 0 schaltet das Kontrollswort einen Multiplexer, der eine Komponente von **x** auswählt und dieses Auswahlbit b bestimmt über einen Multiplexer den Folgezustand: z := if b then z+1 else q.



Umzeichnung dieses Bildes ergibt den Abwickler und Entschlüssler einer Wegener-Maschine.



Vertikale Mikroprogrammierung und Programmsteuerung einer Wegener-Maschine sind äquivalent.

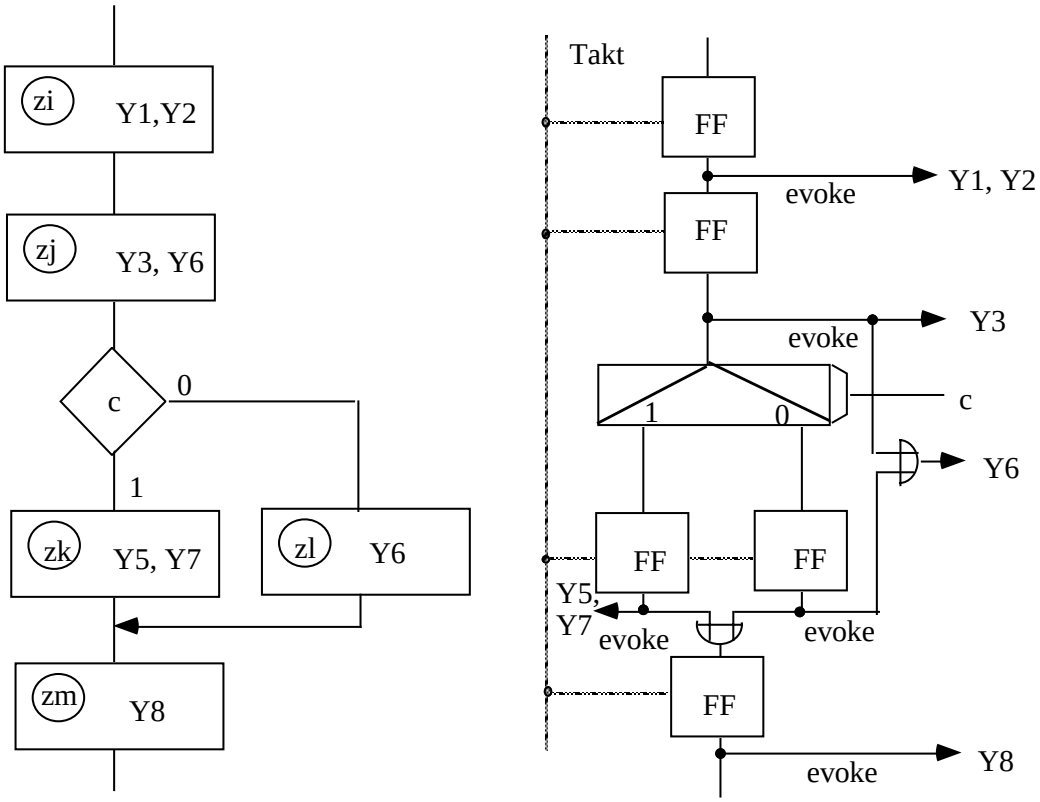
3.3. Verteilte Steuerungen

Hat man nur wenige Zustände insgesamt, dann macht es Sinn, jedem Zustand ein eigenes Flip-Flop zuzuordnen. Die Fortschaltung von einem Zustand in den nächsten kann dabei synchron oder asynchron erfolgen.

3.3.1. Synchrone verteilte Steuerung

Für alle Flip-Flops gibt es einen gemeinsamen Takt.

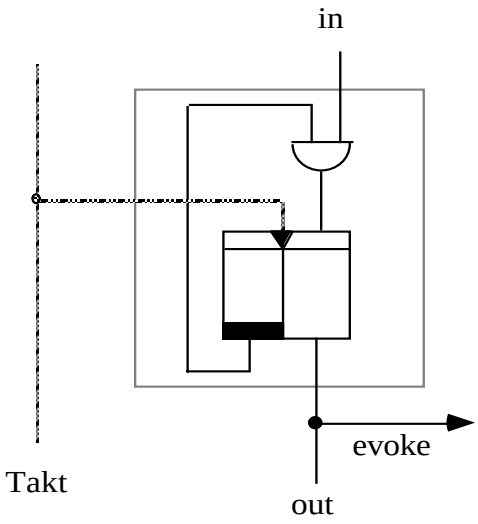
Die Gegenüberstellung einer ASM-Karte und der Realisierung mit Flip-Flops zeigt die Abbildung.



Werden die gleichen Steuerleitungen in mehr als einem Zustand aktiviert, sind sie mit ODER-Gattern zu verschalten.

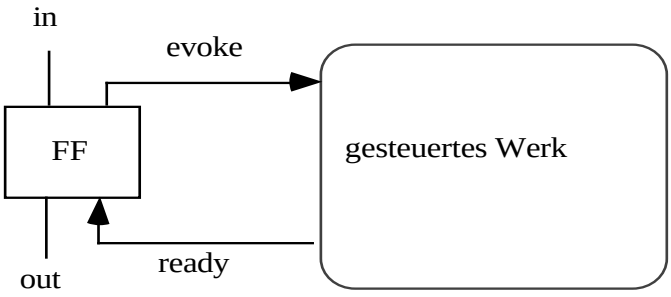
Den Aufbau eines einzelnen Flip-Flops zeigt die nächste Abbildung:

Wenn der Ausgang Eins war, dann wird er im nächsten Takt auf Null fallen. Nur wenn der Ausgang Null ist, kann ein Eingangssignal Eins in den Flip-Flop übernommen werden.

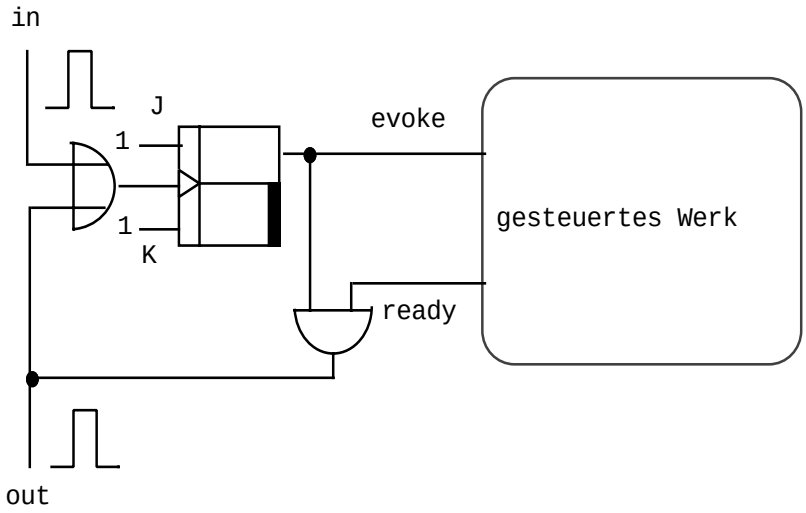


3.3.2. Asynchrone verteilte Steuerung

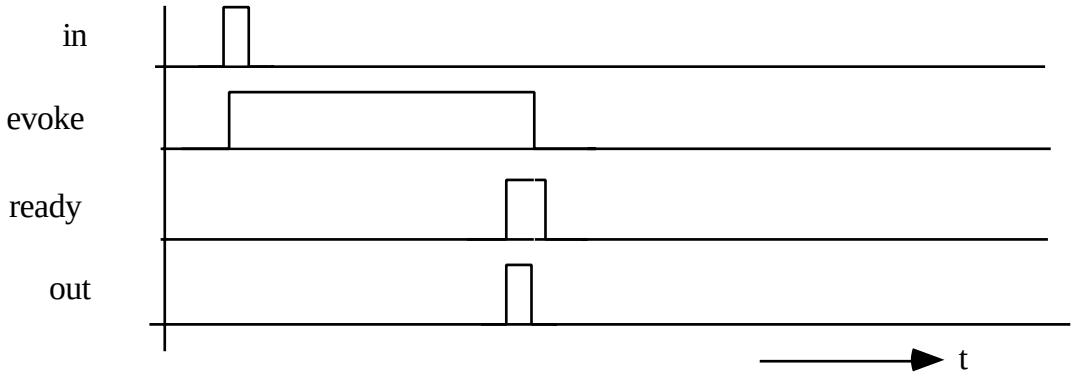
Die Idee hier ist der Aufbau einer Steuerung so, daß Signale (= Fertigmeldungen) aus dem zu steuernden Gerät die Weiterschaltung in den Folgezustand auslösen. Diese Signale sind ihrer Natur nach asynchron.



Die Hardware-Realisierung könnte dann so aussehen:

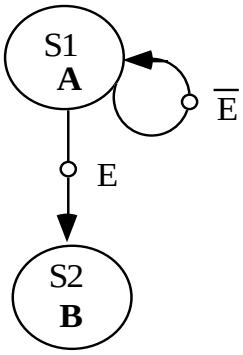


und der zeitliche Verlauf so:



3.3.3. Speicherprogrammierte Steuerung (SPS)

Die technische Realisierung dieser Art von Steuerungen geschieht heute in der Form Speicherprogrammierter Steuerungen (SPS): Ein Rechner simuliert die asynchrone Steuerung und tastet die Fertigmeldungen hinreichend häufig ab, um innerhalb garantierter Zeitunsicherheiten die Weiterschaltung machen zu können.



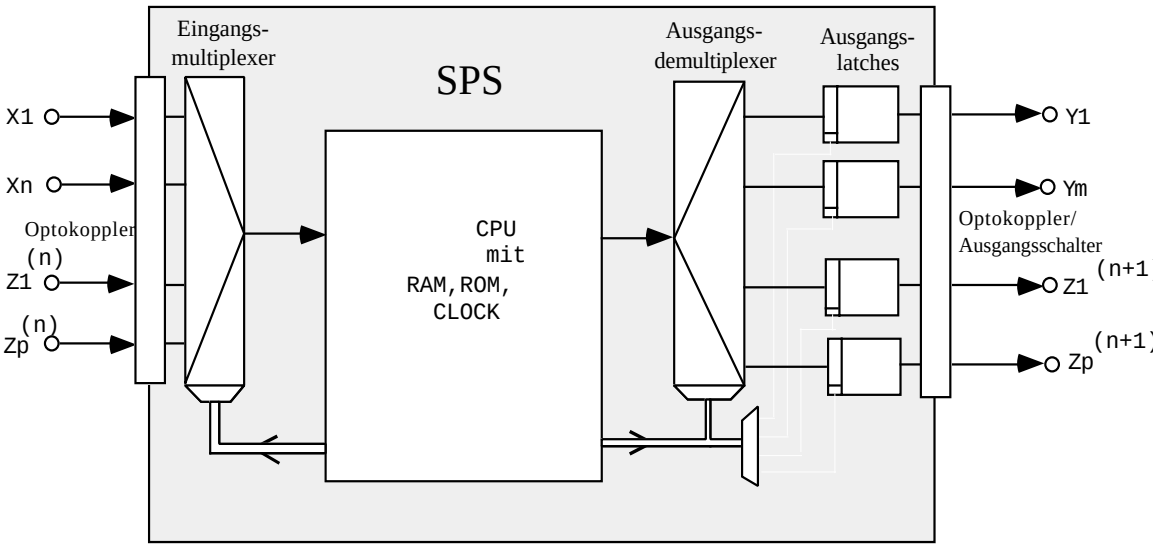
Bei Vorliegen der Weiterschaltbedingungen E im Zustand S1 wird nach S2 geschaltet.

In S1 werden Ausgangssignale A erzeugt.

Die Hardware-Realisierung besteht aus einem Rechner, der die Eingänge des Schaltnetzes abtastet, miteinander verrechnet, d. h. die Funktionen

$$y_i = \omega_i (\mathbf{x}^t, \mathbf{z}^{(n)}) \text{ und } z_j^{(n+1)} = \delta_j (\mathbf{x}, \mathbf{z}^{(n)}) ; \; i = 1, ..., m; \; j = 1, ..., p$$

bildet, und über Ausgangslatches nach außen hin bereitstellt. Ein- und Ausgänge sind über Optokoppler von der CPU im Inneren entkoppelt.



Für viele technische Anwendungen braucht das Schaltnetz, das simuliert wird, nur Gatterlaufzeiten von einigen Millisekunden zu haben.

Rechnet man mit c = 10 ms, dann muß alle 10 ms ein neuer Satz von Ausgängen ($\mathbf{y}$, $\mathbf{z}^{(n+1)}$) erzeugt werden.

Seien (r + p) = 100 Eingänge zu bedienen und sei (r + m) = 100, d. h. 100 Ausgänge, dann sind in 10 ms 100 x 100 = 10^4 Schritte nötig.

Rechnet man pro Schritt 1µs (d. h. rd. 10 - 50 Befehle), dann ist die Forderung von 10 ms einzuhalten:
Nach außen hin verhält sich das Schaltnetz als habe es eine Gatterlaufzeit von 10 ms entsprechend einem mit Relais aufgebauten Werk.