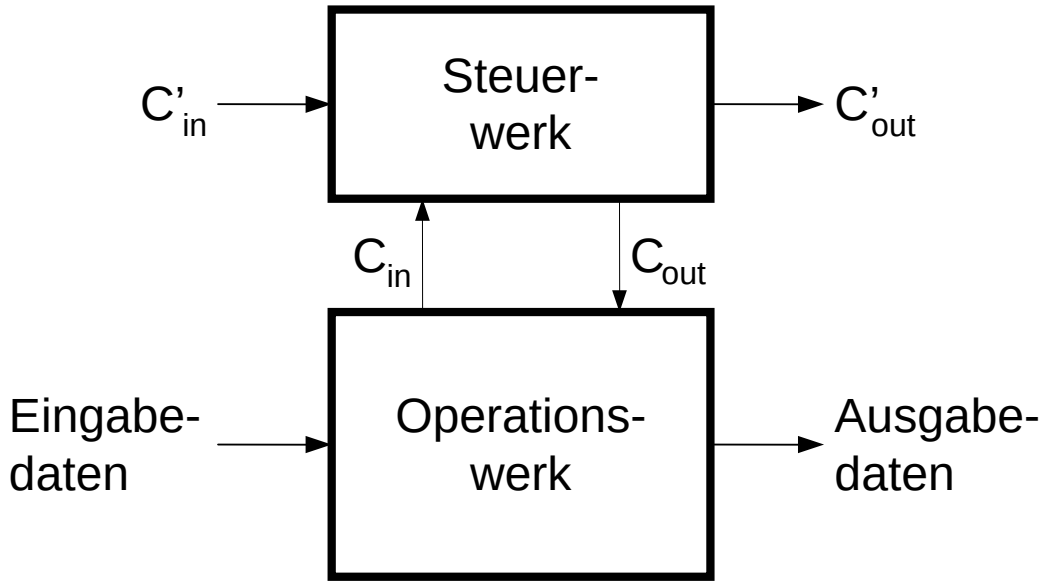
7. REALISIERUNG VON STEUERWERKEN

7.1 Grundprinzip

Eine bewährte Entwurfsmethode bei komplexen Digitalsystemen wie Mikroprozessoren ist die strikte Aufteilung in einen datenverarbeitenden und einen steuernden Teil (vgl. Kap. 6).

Ein **Steuerwerk** (Kontrolleinheit, Control Unit) steuert den Ablauf in einem **Operationswerk** (Verarbeitungseinheit, Data Processing Unit, Datenpfad, Data Path). Zusätzlich werden i. Allg. Kontrollsignale (Start, Stopp, Fehler etc.) von außen verarbeitet bzw. nach außen generiert.



C_{in}: interne Eingabesignale (Kriterien), die die Ablaufsteuerung im Steuerwerk beeinflussen

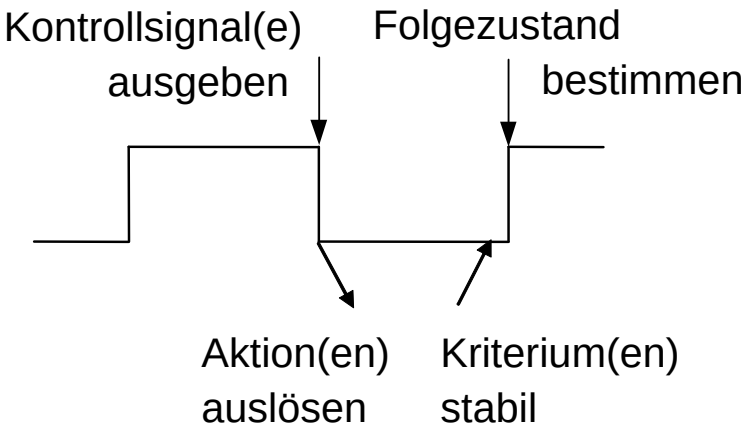
C_{out}: interne Kontrollsignale (Mikrooperationen) vom Steuerwerk an die Verarbeitungseinheit

C'_{in}: externe Eingabesignale von anderen Kontrolleinheiten (z. B. BEGIN, RESET)

C'_{out}: externe Ausgabesignale an andere Kontrolleinheiten (z. B. BUSY, END)

Das Timing von Steuerwerk und Operationswerk muss sorgfältig abgestimmt werden, z. B. bei Moore-Timing:

- *positiv* flankengetriggerte FFs für das Steuerwerk
- *negativ* flankengetriggerte oder zwei-zustands-gesteuerte FFs für das Operationswerk



Gilt für andere Steuerwerk-Implementierungen mit Moore-Timing ganz analog.

Dieses zweiphasige Design hat den Vorteil, dass laufzeit-bedingte Effekte (Hazards, Glitches) bis zum Beginn der nächsten Phase abgeklungen sind.

Steuerwerk und Operationswerk arbeiten also immer auf stabilen Signalen (wenn die Taktphasen lang genug sind).

Anmerkung: Die im Folgenden für Steuerwerke vorgestellten Methoden sind natürlich auch direkt für Schaltwerke im Allgemeinen anwendbar.

Realisierungsprinzipien für Steuerwerke (mehr unten):

- festverdrahtet (hardwired control)
  - Zustandstabellen-Verfahren (Moore- oder Mealy-Automat)
  - Verzögerungselemente (delay-element method)
  - Zählersteuerung (sequence-counter method)

  und andere Verfahren

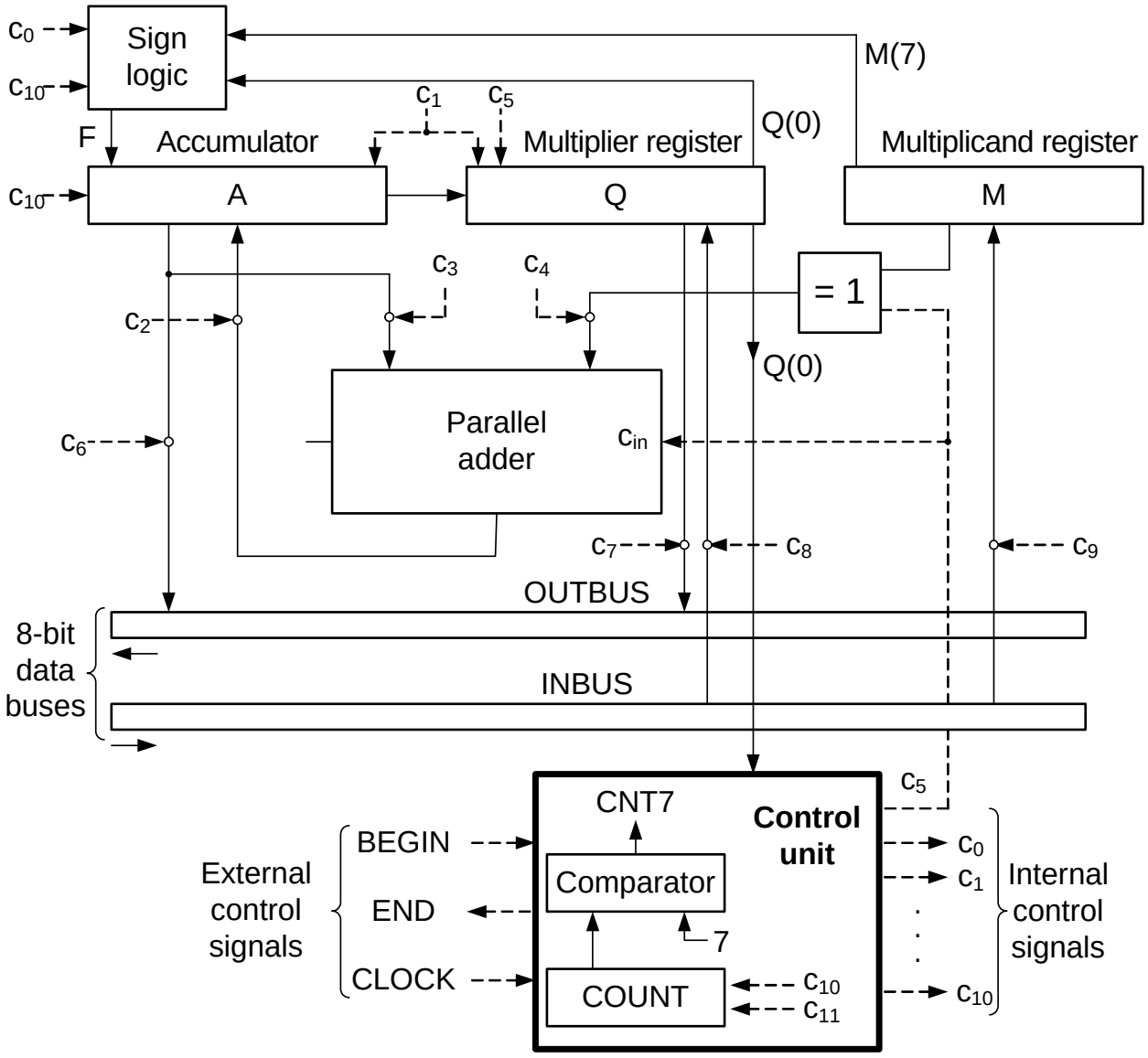
- mikroprogrammiert (microprogramming)

  Die Generierung eines Steuersignals wird als Mikrobefehl aufgefasst. Ein Mikroprogramm ist dann eine Folge von Mikrobefehlen zur Realisierung der Ablaufsteuerung.

  Es wird in einem Mikroprogrammspeicher abgelegt und vom Mikroprogrammwerk ausgeführt.

7.2 Festverdrahtete Steuerwerke

7.2.1 Beispiel: Steuerwerk für einen Zweierkomplement-Multiplizierer



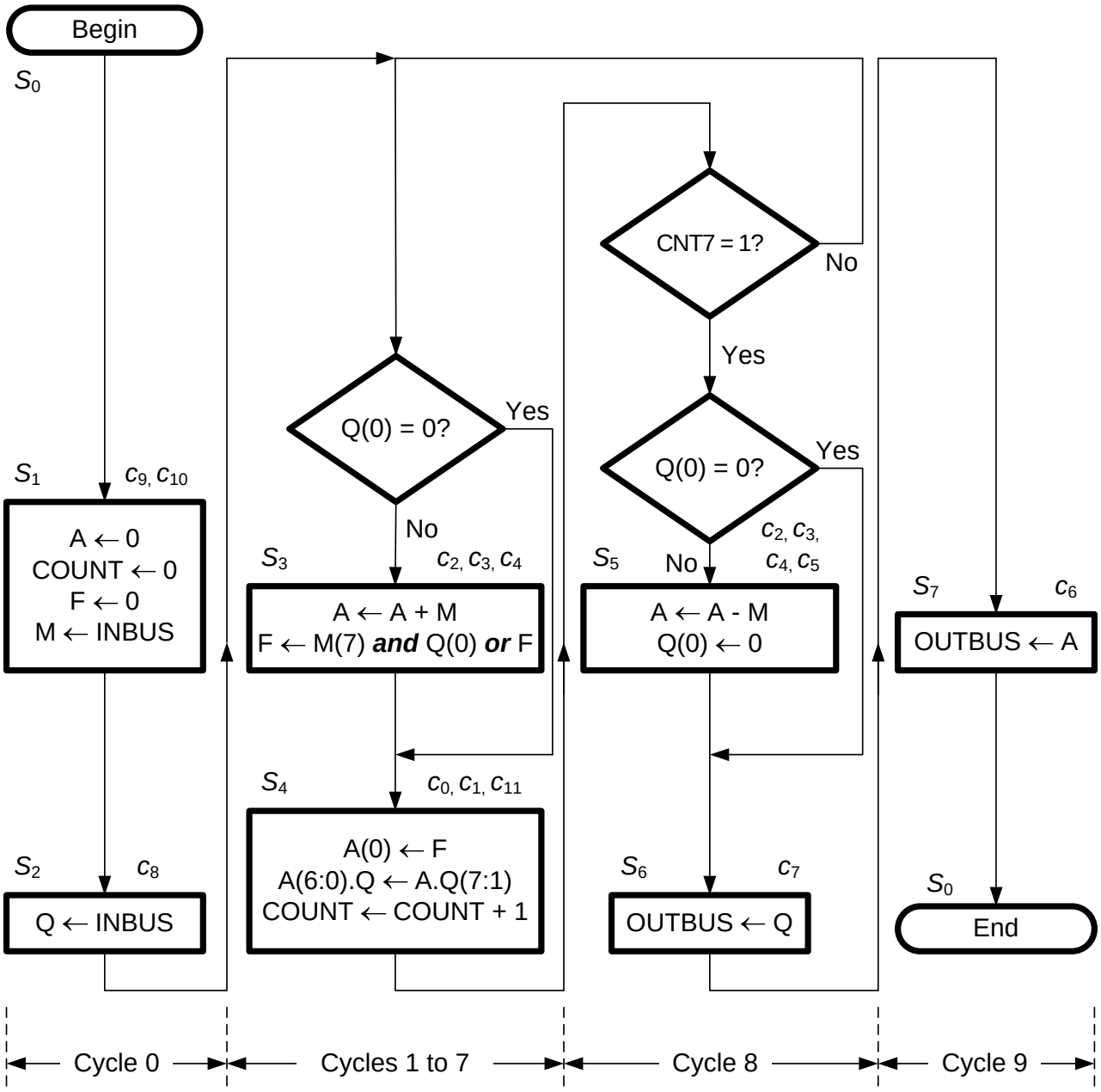
Festlegung der Kontroll-signale und Kriterien

(ggf. zusammengefasst)

Input	conditions
BEGIN	Start signal
Q(0)	LSB of Q
CNT7	COUNT = 7

Control signal	Operation controlled
c_0	Set sign bit of A to F.
c_1	Right-shift register-pair A.Q.
c_2	Transfer adder output to A.
c_3	Transfer A to left input of adder.
c_4	Transfer M to right input of adder.
c_5	Perform subtraction (correction). Clear Q(0).
c_6	Transfer A to OUTBUS.
c_7	Transfer Q to OUTBUS.
c_8	Transfer word on INBUS to Q.
c_9	Transfer word on INBUS to M.
c_{10}	Clear A, COUNT, and F registers.
c_{11}	Increment COUNT.
END	Completion signal

Flussdiagramm für Zweierkomplement-Multiplikation
(siehe auch Kap. 6.7)



Diese Darstellung verdeutlicht, dass die Multiplikation in verschiedenen Phasen abläuft, die sich in den auszuführenden Operationen und zu prüfenden Bedingungen unterscheiden.

Zustandstabellen-Verfahren

Das Steuerwerk wird als ***Moore- oder Mealy-Automat*** nach bekannten Verfahren der Schaltwerksynthese entworfen.

Eingabe:	CNT7	(Count = 111_2)
	Q(0)	(rechtes Bit des Q-Registers)
	BEGIN	(externes Startsignal)

$\Rightarrow$ 8 mögliche Eingabekombinationen

Ausgabe:	c_0, ..., c_{11}	(interne Kontrollsignale)
	END	(externes Kontrollsignal)

Zustände:	S_0, ..., S_7	laut Flussdiagramm

Anmerkung: Moore-Automat, wenn Ausgaben nur fest an den Zustand gebunden sind;

Mealy-Automat, wenn Ausgaben auch direkt von der Eingabe abhängen

Vgl. Entwurfsschema aus Kap. 6.3

Zustandstabelle (Moore-Automat)

Vereinfachung durch Zusammenfassen von Ausgaben, die immer gemeinsam auftreten:

c_0 für $\{c_0, c_1, c_{11}\}$
c_2 für $\{c_2, c_3, c_4\}$
c_9 für $\{c_9, c_{10}\}$

ASSIGN	PS	NS									OUTPUTs*	INPUT-VARs
		1	0	-	-	-	-	-	-	-		BEGIN
		-	-	-	1	0	1	0	0	1		Q(0)
		-	-	-	-	-	1	1	0	0		CNT7
000	S_0	S_1	S_0	-	-	-	-	-	-	-	00000001	
001	S_1	-	-	S_2	-	-	-	-	-	-	00000010	
010	S_2	-	-	-	S_3	S_4	-	-	-	-	00000100	
011	S_3	-	-	S_4	-	-	-	-	-	-	01000000	
100	S_4	-	-	-	-	-	S_5	S_6	S_4	S_3	10000000	
101	S_5	-	-	S_6	-	-	-	-	-	-	01100000	
110	S_6	-	-	S_7	-	-	-	-	-	-	00001000	
111	S_7	-	-	S_0	-	-	-	-	-	-	00010000	

(Hier Tabellenformat vom Designtool LogicAid mit:

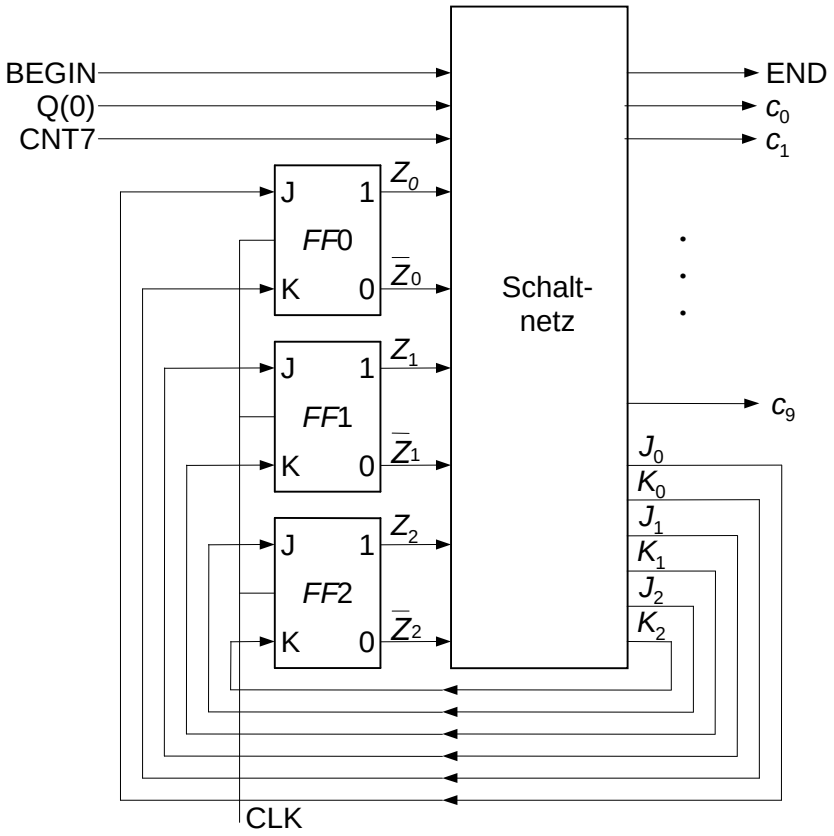
ASSIGN: Zustandscodierung,
PS: Present State,
NS: Next State)

* Ausgangsvektor: $c_0, c_2, c_5, c_6, c_7, c_8, c_9$, END

„-" entspricht Don't Care für Eingänge bzw. nicht spezifiziert für Zustände

Hier liegt ein partiell definierter Automat vor, weil nicht alle Folgezustände definiert sind.

Steuerwerk als Schaltwerk mit JK-Flipflop



Übergangs- und Ausgabegleichungen im Schaltnetz:

$$J_0 = \mathrm{BEGIN}\,\overline{Z_2}\,\overline{Z_1} + Q(0)\,Z_1 + Q(0)\,Z_2\,Z_1, \qquad K_0 = 1$$

$$J_1 = Z_0 + \overline{Q(0)}\,\mathrm{CNT7}\,Z_2 + Q(0)\,\overline{\mathrm{CNT7}}\,Z_2, \qquad K_1 = \overline{Q(0)}\,\overline{Z_2} + Z_0$$

$$J_2 = \overline{Q(0)}\,Z_1 + Z_1\,Z_0, \qquad K_2 = Q(0)\overline{\mathrm{CNT7}}\,\overline{Z_1}\,\overline{Z_0} + Z_1\,Z_0$$

$$c_0 = Z_2\,\overline{Z_1}\,\overline{Z_0}, \qquad c_2 = \overline{Z_2}Z_1\,Z_0 + Z_2\overline{Z_1}Z_0$$

$$c_5 = Z_2\overline{Z_1}Z_0, \qquad c_6 = Z_2Z_1Z_0$$

$$c_7 = Z_2Z_1\overline{Z_0}, \qquad c_8 = \overline{Z_2}Z_1\overline{Z_0}$$

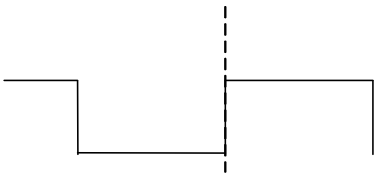
$$c_9 = \overline{Z_2}\,\overline{Z_1}Z_0, \qquad \mathrm{END} = \overline{Z_2}\,\overline{Z_1}\,\overline{Z_0}$$

Entwurf mit Standardmethoden bzw. CAD-Tools

7.2.2 Zweiphasensteuerung

Neues Sprachkonstrukt für RTeasy:

Der „Pipe-Operator“ **|** trennt einen Takt in zwei Phasen



Zwei-Phasen-Timing bzw. Zweiflankensteuerung ist erforderlich, um beim Moore-Automat den Folgezustand von einer RT-Operation abhängig zu machen, die im **gleichen** Takt aktiviert wird.

D. h., auf der zweiten Flanke wird das ausgewertet, was als Reaktion auf die erste Flanke passiert ist.

Die links vom |-Operator stehende Anweisungsliste wird also in der ersten Taktphase, die rechts stehende Anweisungsliste in der zweiten Taktphase ausgeführt.

Mit dem |-Operator kann ein RT-Programm nicht nur syntaktisch einfacher (schöner) formuliert werden. Er bietet auch die Möglichkeit, einen Moore-Automat so zu beschreiben, dass Takte eingespart werden können.

Beispiel:

Q ← INBUS **| if** Q(0) = 0 **then goto** RSHIFT **fi**

spart einen Takt im Vergleich zu:

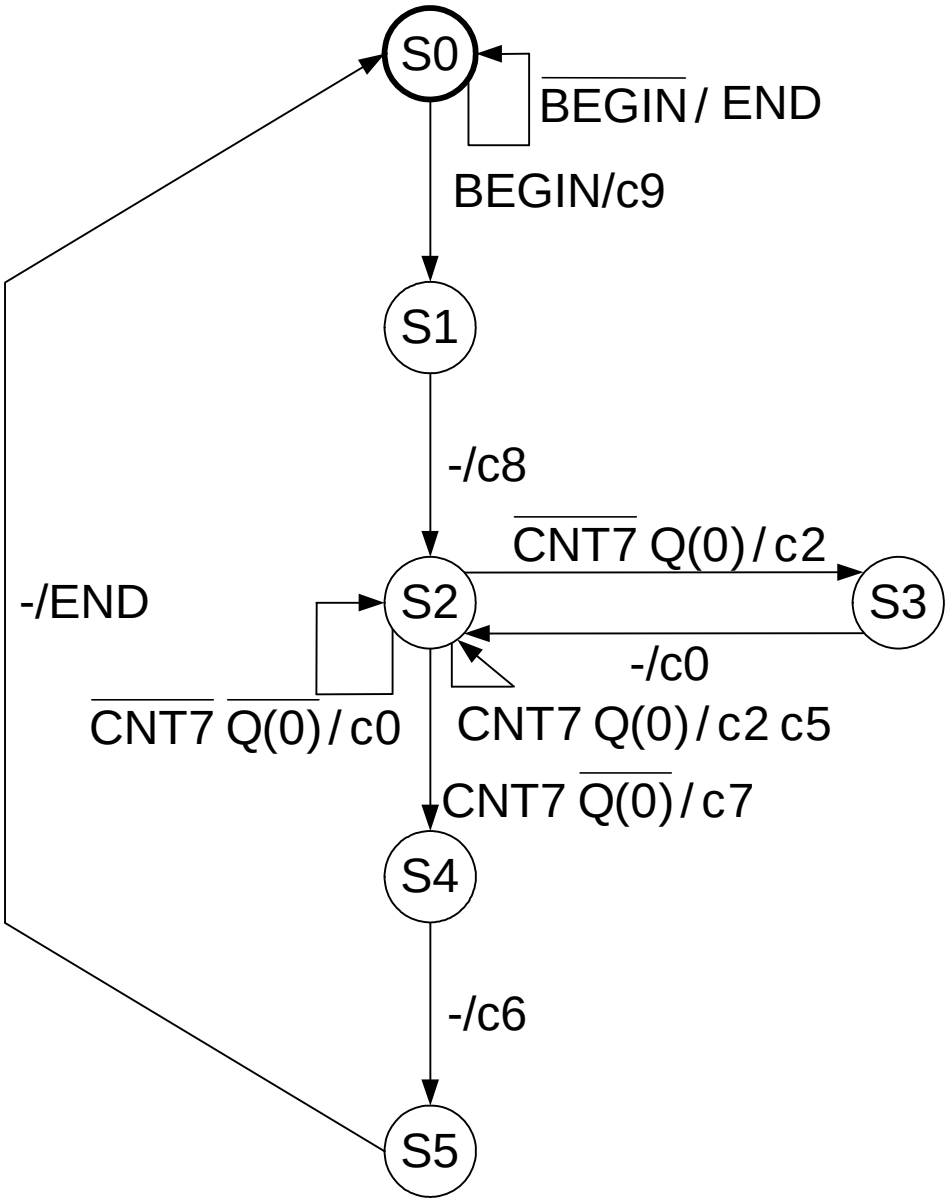
Q ← INBUS;
if Q(0) = 0 **then goto** RSHIFT **fi**

Beschreibung der Multiplikation mit erweitertem RTeasy
(Kontrollsignale als Kommentar zugeordnet)

```
# Zweierkomplement-Multiplizierer (Moore, mit Pipe-Operator)
declare register F, A(7:0), Q(7:0), M(7:0), COUNT(2:0);

declare bus     BEGIN, END,                          # Ext. Kontrollsignale
                INBUS(7:0), OUTBUS(7:0)

START:          END <- 1 | if BEGIN = 0 then goto START fi;
                                                     # END
INPUT:          A<-0, COUNT <- 0, F <- 0,

                M <- INBUS;                          # c9
                Q <- INBUS |                         # c8

                if Q(0) = 0 then goto RSHIFT fi;

ADD:            A <- A + M, F <- (M(7) and Q(0)) or F;   # c2

RSHIFT:         A(7) <- F, A(6:0).Q <- A.Q(7:1),
                COUNT <- COUNT + 1 |                 # c0
                if COUNT <> 7 then
                   if Q(0) then goto ADD
                           else goto RSHIFT fi
                   else if Q(0) = 0 then goto OUTPUT fi fi;

CORRECT:        A <- A - M, Q(0) <- 0;               # c2,c5

OUTPUT:         OUTBUS <- Q;                         # c7

                OUTBUS <- A | goto START;            # c6
```

Die Verwendung des |-Operators vermeidet im Gegensatz zu dem RTeasy-Programm aus Abschnitt 6.7 unnütze Leertakte bei ADD und TEST für Q(0) = 0 bzw. COUNT <>7.

7.2.3 Steuerwerk als Mealy-Automat



Weil bei einem Mealy-Automat abhängig von den Eingangssignalen verschiedene Aktionen ausgeführt werden können, während der Automat in einem Zustand bleibt, sind hier nur 6 Zustände erforderlich, da die Ausgaben den Kanten zugeordnet werden.

Allerdings ist die Abarbeitungsstruktur schlechter erkennbar und wartbar als beim Moore-Automat.

RTeasy-Programm zum Mealy-Automat

```
# Zweierkomplement-Multiplizierer (Mealy, ohne Pipe-Operator)

declare register     F, A(7:0), Q(7:0), M(7:0), COUNT(2:0);

declare bus  BEGIN, END,                                    #Zustaende
             INBUS(7:0), OUTBUS(7:0);

START:  if BEGIN then A <- 0, COUNT <- 0, F <- 0,
                  M <- INBUS else goto START fi;            #S0

INPUT:  Q <- INBUS;                                         #S1

TEST:   if COUNT <> 7 then                                  #S2
          if Q(0) then A <- A + M, F <- (M(7) and Q(0)) or F,
                       goto RSHIFT
             else A(7) <- F, A(6:0).Q <- A.Q(7:1),
              COUNT <- COUNT + 1, goto TEST fi
          else if Q(0) then A <- A - M, Q(0) <- 0, goto TEST
                  else OUTBUS <- Q fi fi;

OUTPUT: OUTBUS <- A;                                        #S4

        END <- 1, goto START;                               #S5

RSHIFT: A(7) <- F, A(6:0).Q <- A.Q(7:1),                    #S3
        COUNT <- COUNT + 1, goto TEST;
```

Entsprechend den Datenabhängigkeiten sind hier keine Leerzyklen nötig.

Durch verschachtelte IF-Schleife und „alleinstehende“ Befehle (z. B. RSHIFT) ziemlich unübersichtlich.

Das *Mealy-Timing* ist mit RTeasy ohne das **|**-Konstrukt darstellbar, d. h. Kriterien werden immer erst im *nächsten* Takt ausgewertet.
Dadurch ist ein gemeinsamer Takt für das Steuer- und das Operationswert möglich (kein 2-Phasen-Timing erforderlich).

7.3 Andere Arten von Steuerwerken

7.3.1 Steuerwerke mit Verzögerungsketten (One-Hot Design)

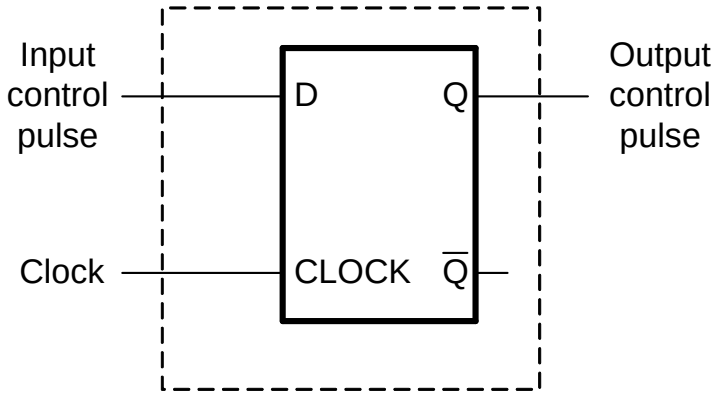
Ein Ablauf lässt sich auch darstellen als eine zeitliche Sequenz von Kontrollsignalen:

Zeit: ↓

t_1 : activate $\{c_{1,j}\}$
t_2 : activate $\{c_{2,j}\}$
...
t_n : activate $\{c_{n,j}\}$

Die sequentielle Aktivierung der Steuersignale erfolgt durch eine Kette von n „Verzögerungselementen“. Dabei wird ein Zustand immer durch **genau ein** Flipflop repräsentiert, das den Wert 1 hat. D. h., es ist immer nur genau ein Verzögerungs-element zu einem Zeitpunkt aktiv. → ***One-Hot***

Beispiel: D-Flipflop als Verzögerungselement (Delay element)



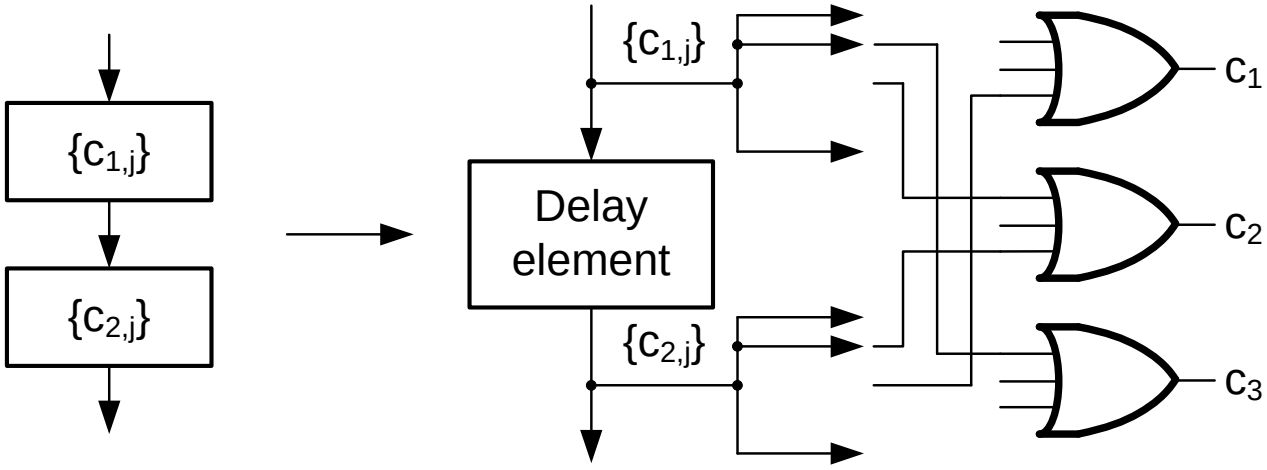
Die Ausgaben sind den Zuständen direkt zugeordnet. Die einzelnen Steuersignale werden einfach durch ODER-Verknüpfung der Ausgänge von den Verzögerungselementen generiert, in deren Zustand sie ausgegeben werden sollen.

Technische Umsetzung

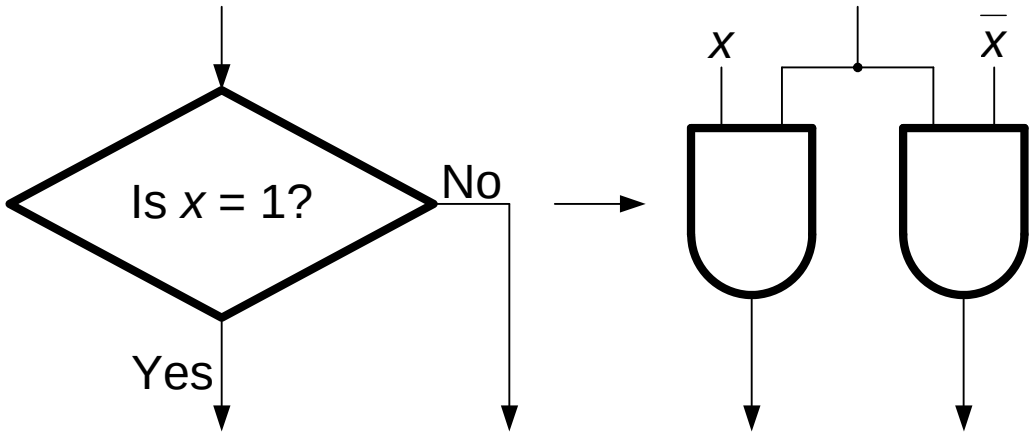
Die Aktivierung des Folgezustands/Flipflops erfolgt ggf. abhängig von den Kriterien.

Ein Flussdiagramm kann dann unmittelbar in eine Kette von Verzögerungselementen transformiert werden.

Sequenz:

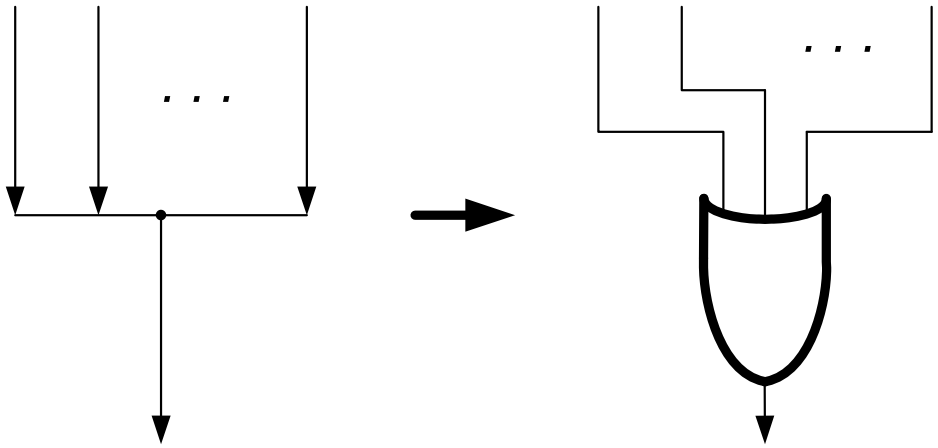


Verzweigung:



Bei einer Verzweigung wird durch (komplementäre) Verriegelungen (UND-Verknüpfung) mit dem Verzweigungskriterium immer **genau ein** nachfolgendes Verzögerungselement aktiviert.

Vereinigung:



Die Vereinigung fasst im Verarbeitungsablauf mehrere alternative Zweige für die Aktivierung **eines** nachfolgenden Verzögerungselements zusammen (ODER-Verknüpfung).

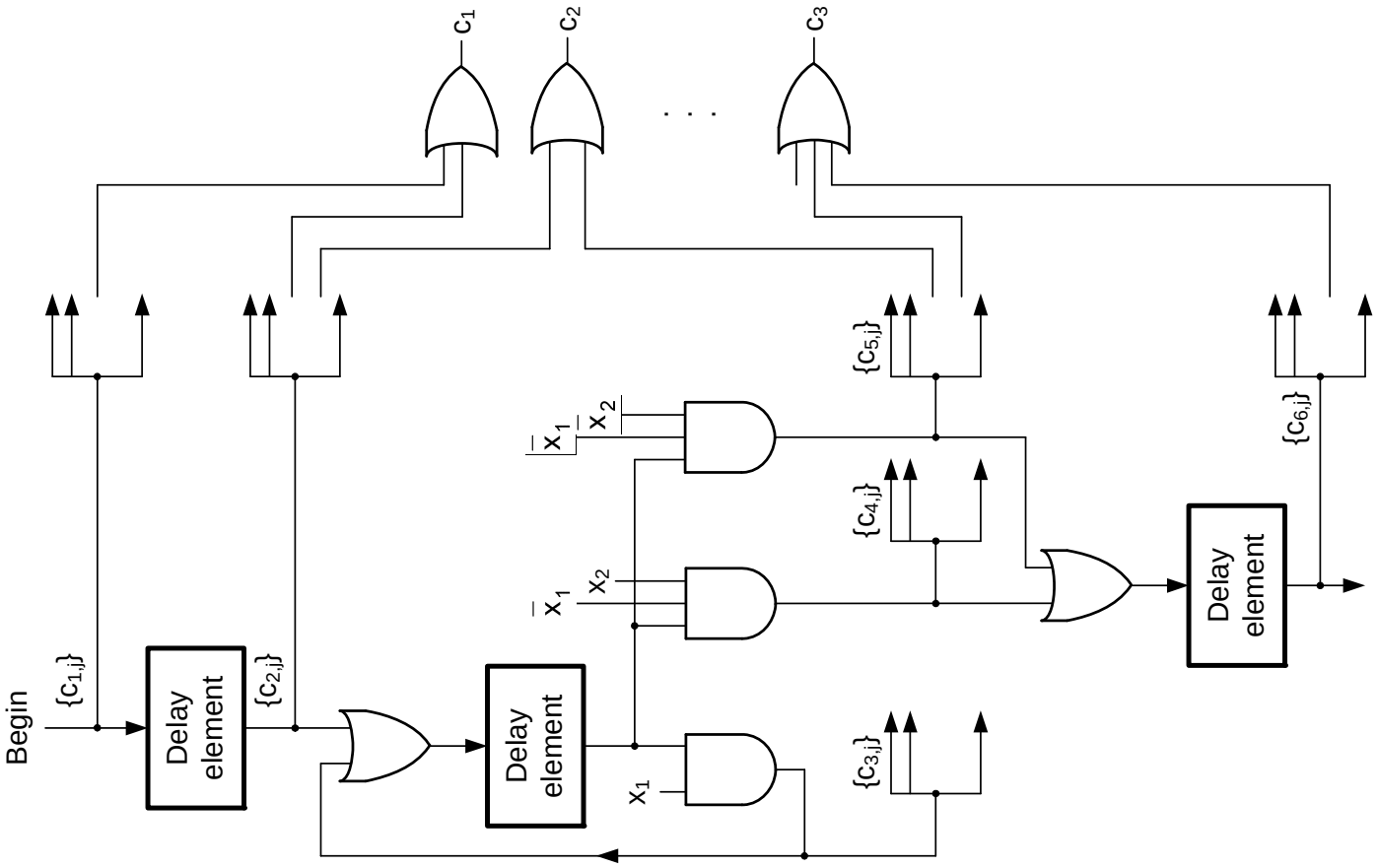
Mit Verzögerungsketten kann sowohl *Mealy*- als auch *Moore*-Timing implementiert werden, je nachdem ob die Eingaben auch Ausgaben direkt beeinflussen können oder die Ausgaben nur vom Zustand abhängen.

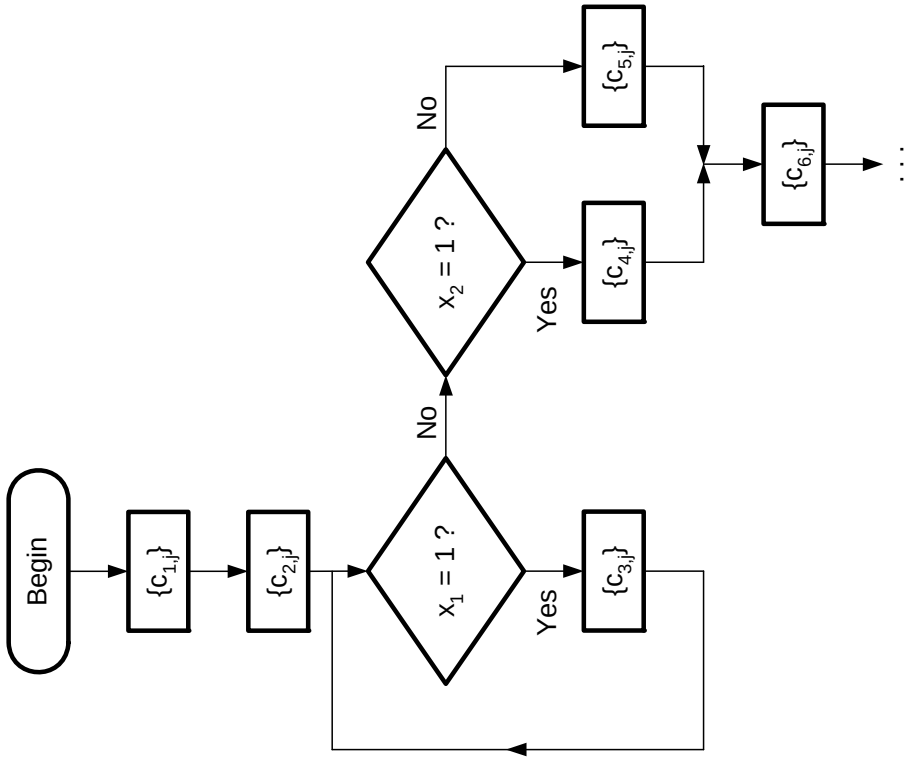
Beim Moore-Timing wirken die Eingaben nur auf die Weiterschaltbedingung, und die Ausgaben hängen nur von den Zuständen der Verzögerungselemente ab.

Beim Mealy-Timing wirken die Eingaben sowohl auf die Weiterschaltbedingungen als auch über das Ausgabeschaltnetz auf die Ausgänge.

Im Vergleich zu einer konventionellen Schaltwerksrealisierung sind bei Verzögerungsketten zwar mehr Flipflops, aber weniger/einfachere Gatter erforderlich.

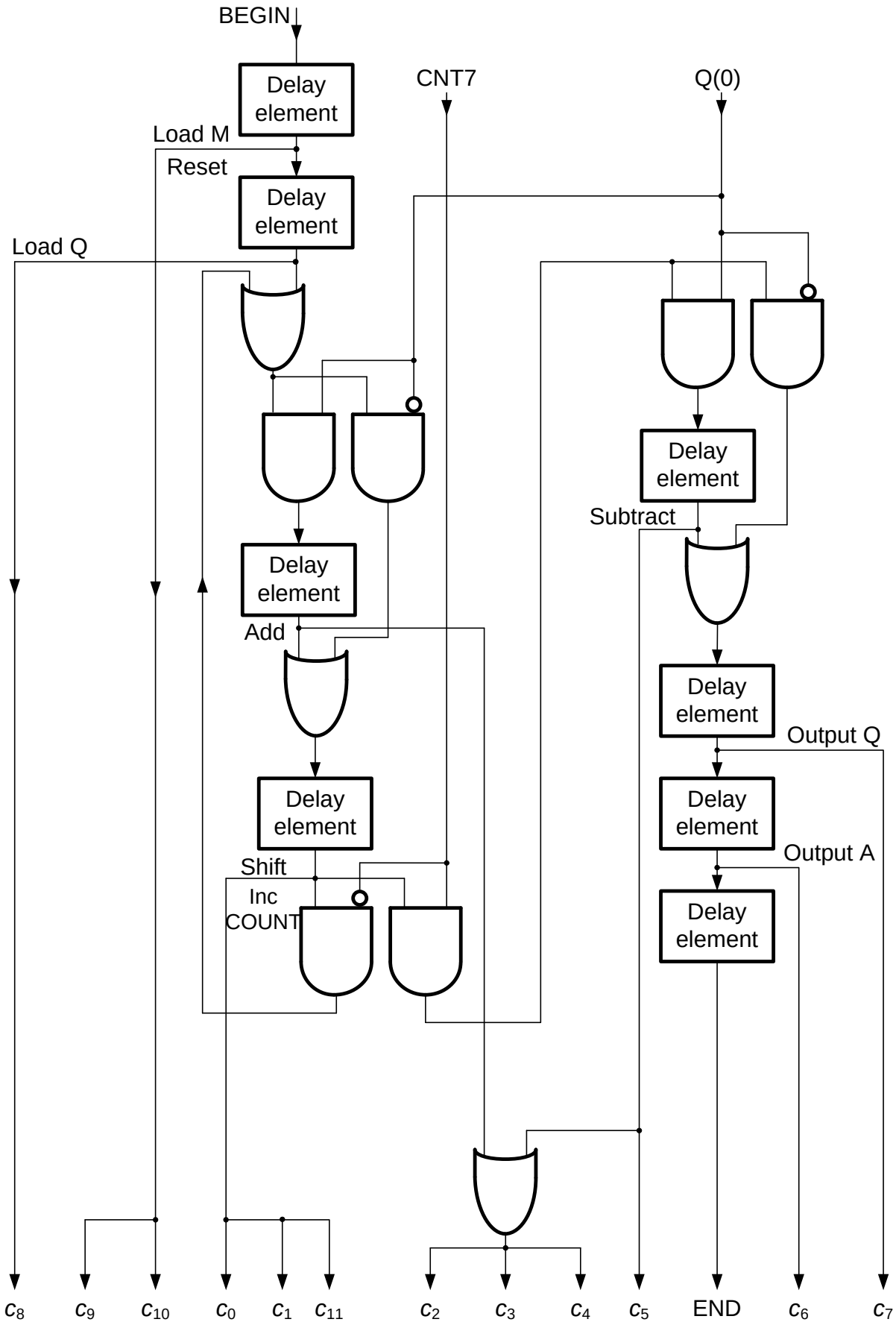
Beispiel für Mealy-Timing:





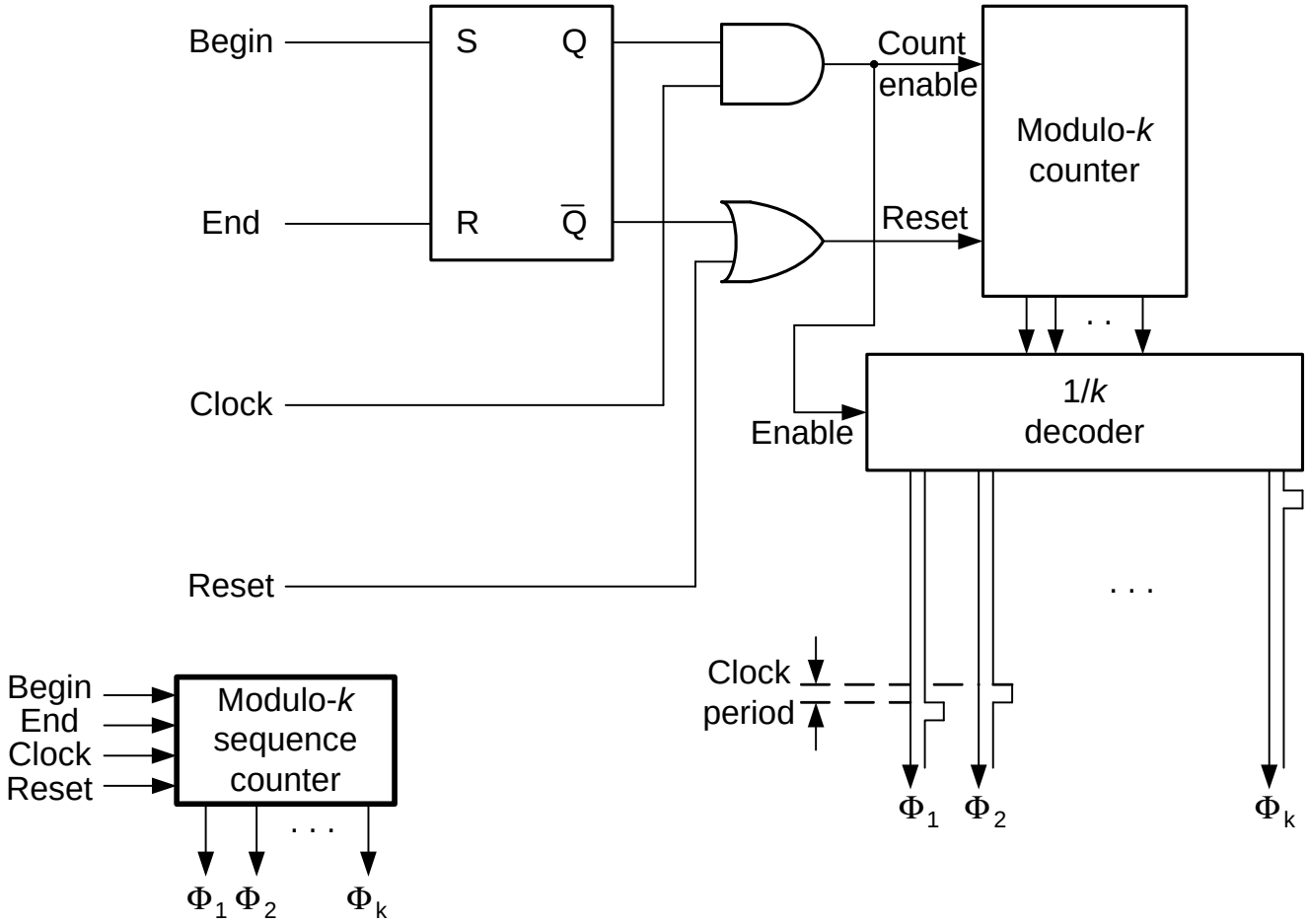
Beachten: Ausgaben $\{c_{3,j}\}$, $\{c_{4,j}\}$, $\{c_{5,j}\}$ von Eingaben x_1 und x_2 abhängig.

Beispiel: Steuerwerk des Zweierkomplement-Multiplizierers aus Verzögerungselementen mit Moore-Timing



7.3.2. Steuerwerke mit Zählersteuerung

Bei einer Zählersteuerung wird die Verarbeitung in zwei Ebenen aufgeteilt, nämlich in Schritte, die mehrere Phasen haben können. Ein Modulo-k-Zähler mit Decoder dient zur Erzeugung von k Steuersignalen Φ_i (Phasensignale), von denen immer **genau eins** aktiv ist.
So werden in einzelnen Verarbeitungsschritten mehrere eindeutige Phasensignale erzeugt.

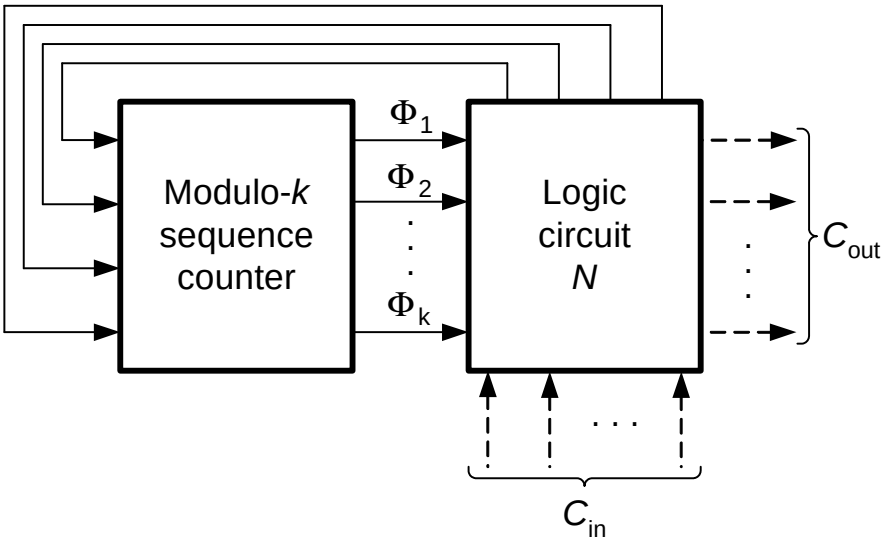


Das Weiterschalten zwischen den Schritten erfolgt durch Setzen des Folge-(RS-)Flipflops und gleichzeitiges Rücksetzen des (RS-)Flipflops für den aktuellen Schritt.

Falls möglich wird dazu ein eindeutiges Steuersignal des aktuellen Schrittes verwendet, um die Logik für die Generierung eines speziellen Weiterschaltsignals zu sparen.

Die bedingte Ausführung von Operationen (z. B. ADD) und das bedingte Weiterschalten erfolgt durch entsprechende logische Verknüpfung mit Kriterien und externen Signalen.

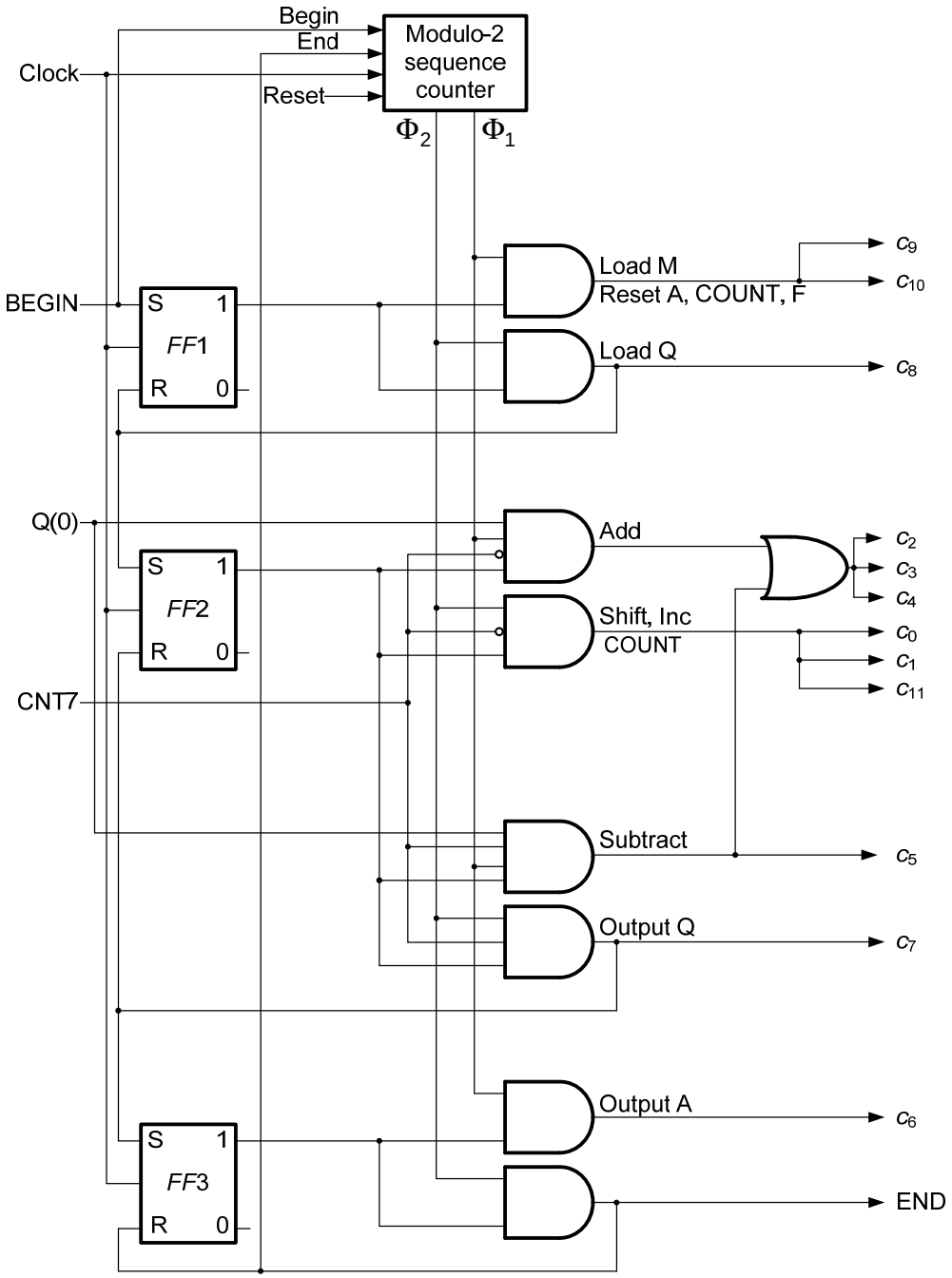
Um aus den Phasensignalen und den Kriterien die Kontrollsignale zu generieren und den Ablauf durch Kriterien zu steuern, ist zusätzliche Logik erforderlich. Sie kann ggf. auch den Zähler bei einem Weiterschalten in den nächsten Schritt zurücksetzen.



Bei dem folgenden Beispiel wirken Kriterien direkt auf die Steuersignale und die Weiterschaltbedingung. D. h., dieses Steuerwerk verwendet *Mealy-Timing*, aber Leertakte, falls nicht addiert bzw. korrigiert wird (vermeidbar durch vorzeitiges Rücksetzen des Zählers über zusätzliche Logik).

Innerhalb der Schritte reicht eine Zählersteuerung mit einem Zähler für die maximale Phasenzahl, hier ein mod-2-Zähler (Phasen Φ_1, Φ_2).

Beispiel: Steuerwerk des Zweierkomplement-Multiplizierers mit Zählersteuerung



Der Ablauf wird mittels RS-Flipflops in 3 Schritte eingeteilt, die jeweils mehrere Phasen haben können:

(1) Initialisierung, Laden von Multiplikator und Multiplikand (Cycle 0 in Kap. 7.2; S1 und S2 beim Automaten)

(2) Produktbildung, Multiplikation (ein Mult-bit pro Zyklus (Cycle 1 - 7; S3, S4), ggf. Korrektur, Ausgabe A (Cycle 8, S5, S6)

(3) Ausgabe Q, END-Signal (Cycle 9; S7)

7.4 Vergleich festverdrahteter Steuerwerke

Zustandstabellen-Verfahren

- nur für kleine Zahl von Zuständen praktikabel
- Struktur der Ablaufsteuerung (z. B. Schleifen) wird verdeckt (bei Mealy- noch mehr als bei Moore-Automaten)
- unterschiedliches Timing bei Moore- und Mealy-Automaten
- schwierige Fehlersuche, schlecht wartbar
- nicht sehr änderungsfreundlich

Verzögerungskette

- einfacher Entwurf, der die Struktur des Flussdiagramms unmittelbar wiedergibt
- Timing wie Mealy- oder Moore-Automat möglich
- geringer Aufwand an kombinatorischer Logik (Gatter)
- hoher Aufwand an Verzögerungselementen (Flipflops):
  n statt $\lceil \log_2 n \rceil$ bei n Zuständen
- gut wartbar

Zählersteuerung

- besonders für Algorithmen mit Schleifen geeignet
- zusätzliche Logik zur Berücksichtigung von Kriterien
- Timing wie Mealy-Automat
- ohne vorzeitiges Rücksetzen des Zählers evtl. Leerzyklen

7.5 Mikroprogrammierte Kontrolleinheiten

7.5.1 Grundprinzip

Der klassische Entwurf eines Steuerwerks für komplexe Digitalsysteme (mit vielen Zuständen sowie Eingangsgrößen und Kriterien des Operationswerks) wird für einen Handentwurf schnell recht aufwändig, fehleranfällig und ist wenig flexibel und schlecht wartbar (denn jede Modifikationen und Erweiterung bedingt einen komplett neuen Hardwarentwurf).

Anstelle einer festverdrahteten Kontrolleinheit wird deshalb oft ein Mikroprogramm-Steuerwerk verwendet.

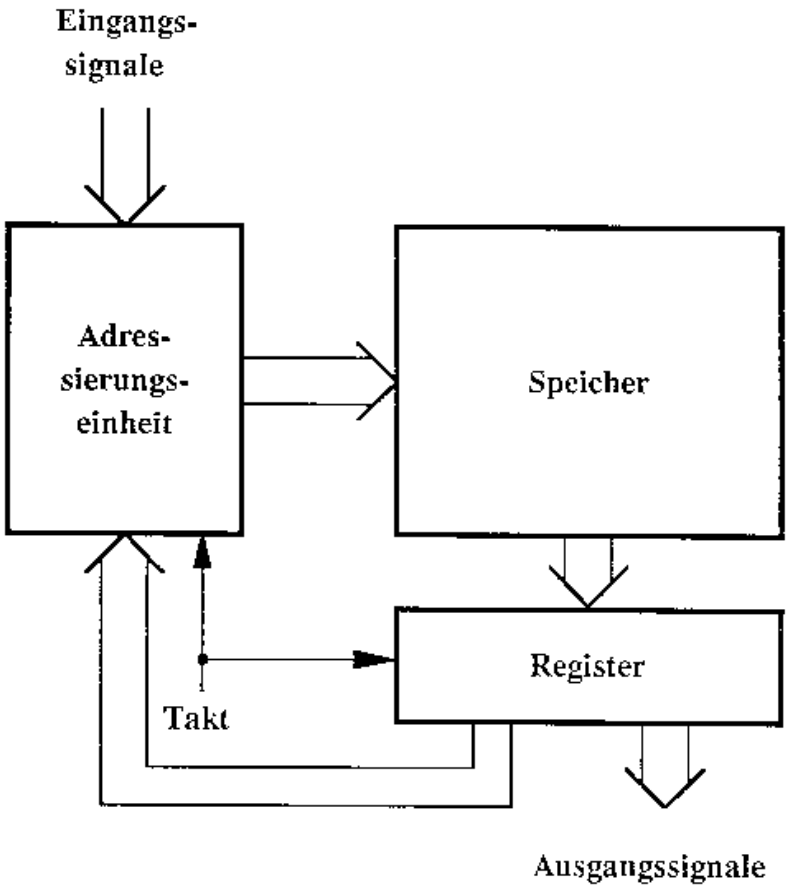
Der Kontrollalgorithmus wird dabei als *Mikroprogramm* in einem (Mikroprogramm-)Speicher im Steuerwerk abgelegt und (Mikro-)Befehl für (Mikro-)Befehl abgearbeitet.

Die Mikrobefehle sind wortweise organisiert (µ-Kontrollworte) und enthalten parallel alle Steuersignale und Angaben für den nächsten auszuführenden Mikrobefehl, d.h.:

- Steuerimpulse (Kontrollsignale, *Mikrooperationen*) für die Datenverarbeitungseinheit (Operationswerk)
- zu berücksichtigende Kriterien (z. B. AC = 0, AC < 0) für bedingte Verzweigungen (im Mikroprogramm)
- Adressinformation für die Generierung der Folgeadresse (implizit mittels µ-Befehlzähler oder explizit)

D. h., ein Teil der Ausgangssignale (des Mikrobefehls) wird zur Steuerung der Adressiereinheit und damit des Ablaufs des Mikroprogramms selbst verwendet.

Prinzipieller Aufbau



Weil der eigentliche funktionsbestimmende Teil in den Mikrobefehlen spezifiziert und im Mikroprogrammspeicher abgelegt ist, ist das Schaltwerk für die Abarbeitung des Mikroprogramms recht einfach und vom Inhalt des Mikroprogrammspeichers weitestgehend unabhängig.

Das Schaltwerk besteht prinzipiell aus:

- einem Register für den aktuellen Zustand (Mikrobefehl)
- einer Adressierungseinheit zur Bestimmung des nächsten auszuführenden Mikrobefehls

Der Mikroprogrammspeicher ist i. d. R. ein von Hersteller programmierter Nur-Lese-Speicher ("mikroprogrammiert").

Die Abarbeitung des Mikroprogramms erfordert eine Adressierungseinheit, die die Mikrobefehle der Reihe nach aktiviert (lineare Adressierung mittels "Mikro-Programmzähler" µPC) oder ggf. auch Verzweigungen ausführt (z. B. durch direktes Setzen der Adresse des gewünschten Mikrobefehls).

Über Kriterien als die Eingangssignale kann die Abarbeitungsreihenfolge der Mikrobefehle, d. h. der Programmablauf, beeinflusst werden.

Vorteil der Mikroprogrammierung:

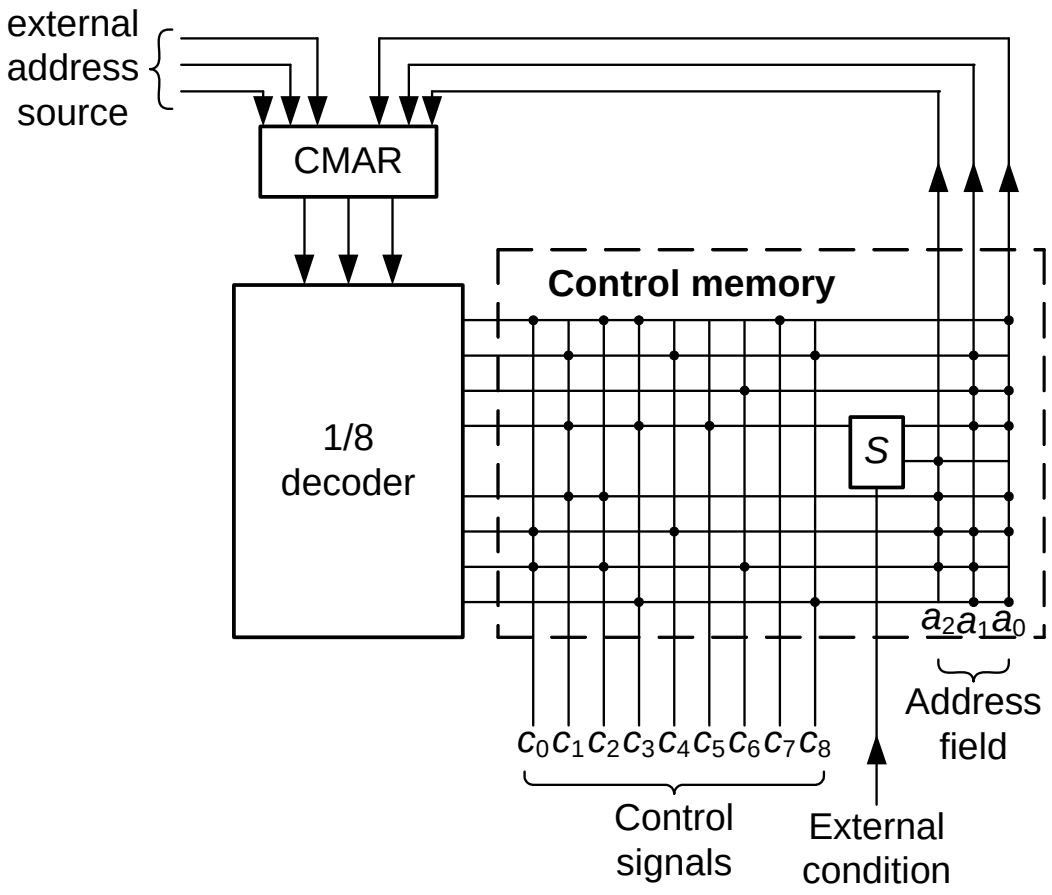
- mit relativ wenigen Mikrobefehlen lässt sich ein recht komplexer Funktionsumfang implementieren
- durch den Austausch des Mikroprogramms ist es möglich, die von außen sichtbare Funktionalität zu ändern, ohne den Rest der Hardware (Steuerwerk zur Abarbeitung des Mikroprogramms und ggf. auch Operationswerk) verändern zu müssen
- einfacher zu entwerfen und zu warten als beim konventionellen festverdrahteten (RT-)Entwurf

Nachteil der Mikroprogrammierung:

- Ausführungszeit oft länger als bei festverdrahteter Logik, weil (Mikroprogramm)Speicherzugriff relativ lang

Die befehlsweise Abarbeitung ähnelt einem von Neumann-Rechner. In der Hierarchie virtueller Rechner liegt die Mikroprogrammebene aber unterhalb der Maschinensprach-Ebene. D.h., ein Maschinenbefehl wird durch eine Sequenz von elementaren Mikrobefehlen (Mikroprogramm) interpretiert.

7.5.2 Mikroprogrammwerke nach Wilkes (1951)



Der Mikroprogrammspeicher (Control Memory) ist als PLD-artige Diodenmatrix realisiert (Zeile = Mikrobefehl, Spalte = Kontrollsignal bzw. Adressbit).

Ein Mikrobefehl (hier: Zeile) besteht aus Mikrooperationen (Control Signals) und dem Adressfeld (*horizontale* Mikroprogrammierung).

Die Adresse des aktuellen Mikrobefehls steht im Register CMAR (Control Memory Address-Register).

Der entsprechende Mikrobefehl (Zeile im Mikroprogramm-speicher) wird über den Decoder aktiviert.

Ein Mikrobefehl enthält im Adressfeld *explizit* die Folgeadresse, die bei seiner Ausführung ins CMAR geladen wird.

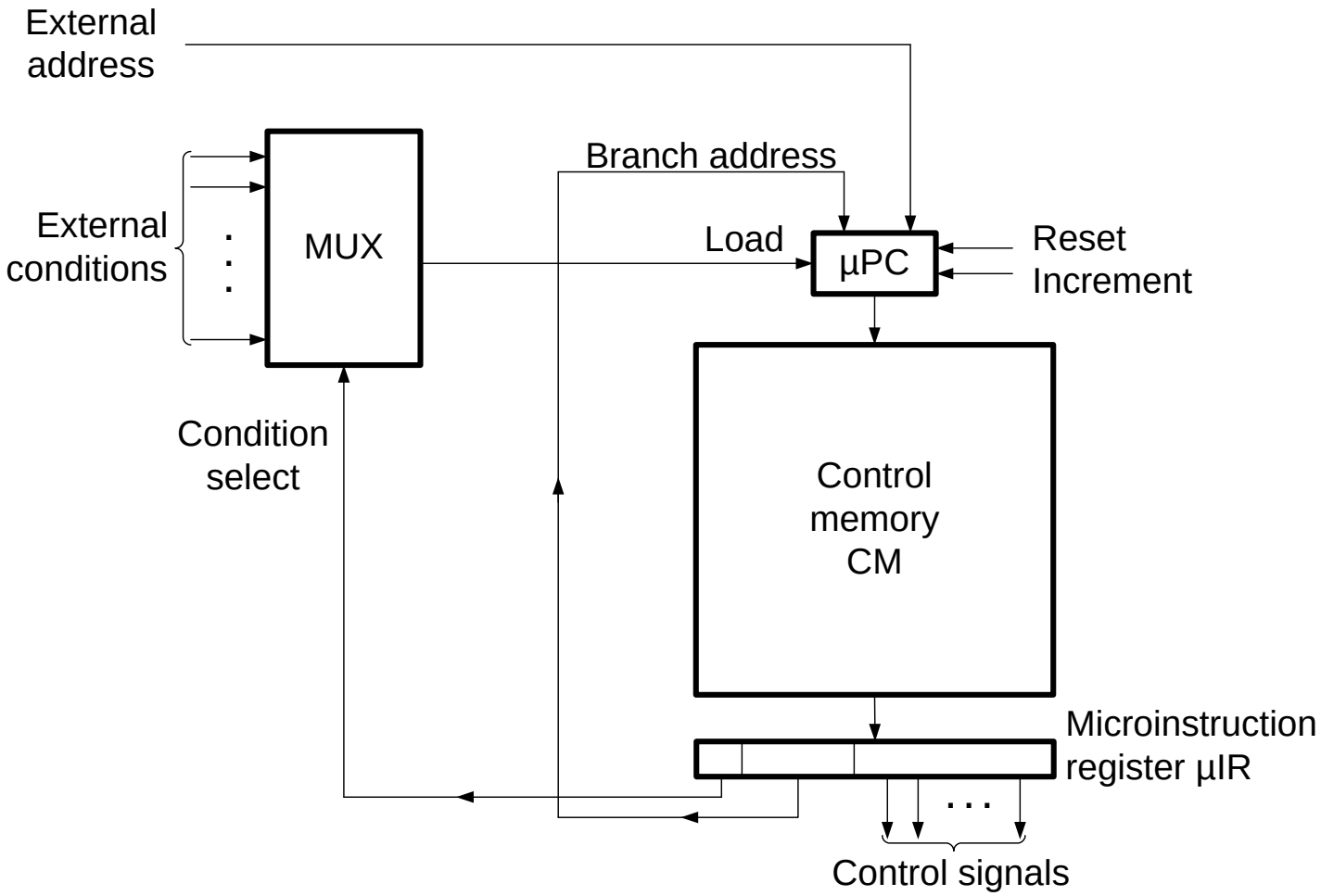
Die Folgeadresse kann von externen Bedingungen (Kriterien) abhängig gemacht werden (bedingte Verzweigungen).

Bei Wilkes ist die dazu gehörige Logik direkt im Mikro-programmspeicher abgelegt und steuert die Folgeadresse direkt (Selektor S).

Das CMAR kann auch von außen geladen werden (Startadresse der Mikroprogramme, z. B. gleich Op-Code aus dem Instruktionsregister eines Mikroprozessors).

7.5.3 Moderne Variante des Mikroprogramm-werks

Ansatz: Bestimmung der Folgeadresse außerhalb des Mikroprogrammspeichers



Typischer Ablauf:

1. Lesen des auszuführenden Mikrobefehls aus dem Mikroprogrammspeicher Control memory CM initiieren
2. Schreiben des auszuführenden Mikrobefehls in das Mikroinstruktionsregister µIR
3. Ausgabe der Kontrollsignale und Auswerten des Mikro-instruktionsregisters µIR im Operationswerk sowie der internen und externen Bedingungssignale zur Bestimmung des nächsten auszuführenden Mikrobefehls im Steuerwerk
4. Adresse des nächsten auszuführenden Mikrobefehls in das Mikrobefehlsregister µPC laden bzw. inkrementieren

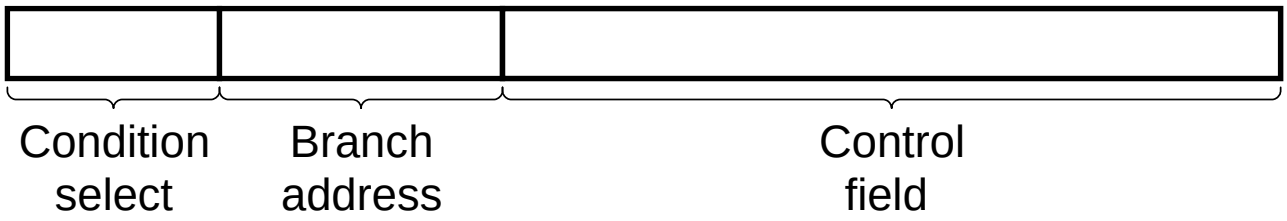
Die Adressfortschaltung wird durch einen Mikrobefehlszähler µPC wie folgt gesteuert:

- Normaler *sequentieller* Befehlsablauf: Erhöhen um 1 (Increment)
- Bei *Verzweigungen* wird die explizit im Mikrobefehl angegebene Branch-Adresse in den µPC geladen (laod).
- *Bedingte und unbedingte Verzeigungen* werden durch Selektion der gewünschten, im Mikrobefehl spezifizierten Bedingungen mittels eines Multiplexers (MUX) realisiert, z. B.

s_0	s_1	**Bedeutung**
0	0	No Branching
0	1	Branch, if ν_1 =1
1	0	Branch, if ν_2 =1
1	1	Uncond. Branch

s_0, s_1: Bedingungs-Bits im µ-Befehl
ν_1, ν_2: externe Bedingungen
MUX-Eingang x_i wird auf Ausgang geschaltet, wenn $i = (s_0, s_1)_2$ mit $x_0 = 0$, $x_1 = \nu_1$, $x_2 = \nu_2$, $x_3 = 1$
für das Laden des Mikroprogrammzählers bei LOAD = 1.

Typisches Mikrobefehlsformat („horizontal“)

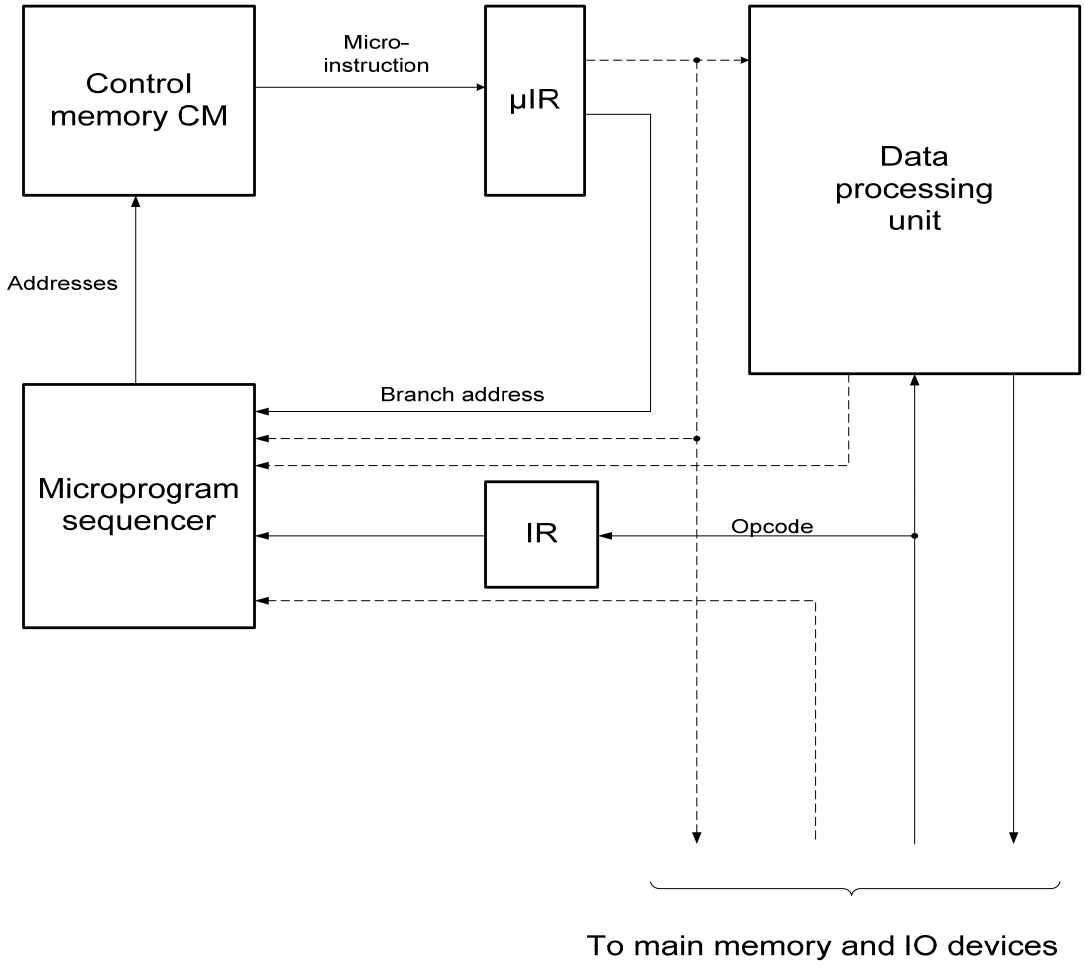


Felder:

- Bedingungsselektion (Condition select) spezifiziert die Bedingungen (Kriterien) für bedingte Verzweigungen
- Adressfeld (Branch address, Address field) enthält die Folgeadresse bei Sprungbefehlen
- Kontrollfeld (Control field) enthält die Kontroll- und Ausgangssignale

Die Ablaufsteuerung bei der Mikroprogrammierung ist recht universell, weil sie vom Inhalt des Mikroprogramms unabhängig ist. *Mikroprogramm-Sequenzer* sind daher auch als integrierte Bausteine erhältlich, die die komplette Logik (z.B. Register und Steuerlogik) zur Mikroadressfortschaltung enthalten (z. B. AMD 2909, TI 8835).

Sie enthalten i. Allg. noch zusätzliche Mechanismen wie z. B. Stack für Rücksprungadressen von µ-Unterprogrammen.



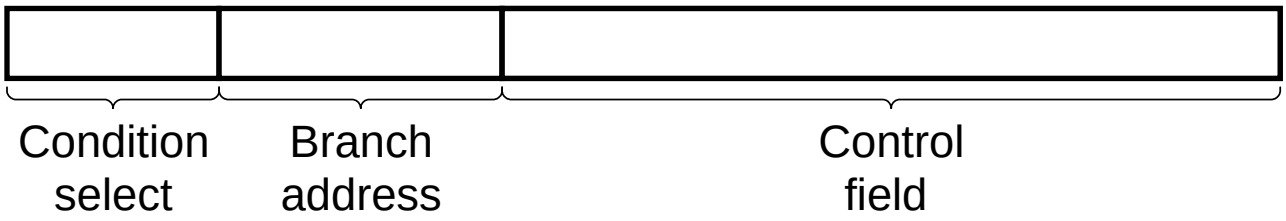
Verfahren zur Optimierung von Mikroprogrammen bzgl. des Speicherbedarfs bekannt (Kompaktifizierung).

Erweiterung: Eine externe Startadresse (z. B. OP-Code vom auszuführenden Maschinenbefehl) kann direkt in den µPC geladen werden, um z. B. von außen ein anderes Mikroprogramm aufzurufen.

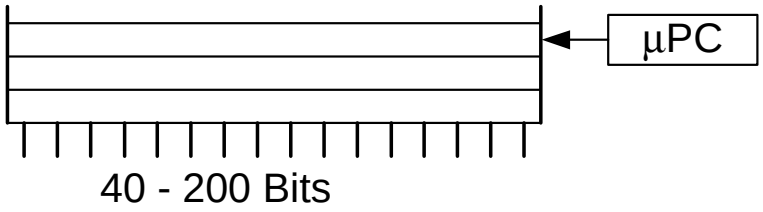
7.5.4 Mikrobefehlsformate

Es sind eine Reihe verschiedener Befehlsformate denkbar, die in Hardware-, Speicher- und Laufzeitbedarf variieren.

Horizontale Mikroprogrammierung



Die einzelnen Bits eines Mikrobefehls entsprechen direkt den Kontrollsignalen und wirken direkt auf die Kontrollpunkte c_i. Außerdem sind alle Angaben für die Bestimmung der Adresse des nächsten Mikrobefehls mit den Sprungbedingungen unmittelbar enthalten.



Das Mikroprogramm besteht aus wenigen, aber breiten Mikrobefehlen. Damit können, falls ohne Konflikte möglich, mehrere Mikrooperationen gleichzeitig gesteuert werden.

Allerdings schließen sich zu einem Zeitpunkt viele Kontrollsignale gegenseitig aus und werden daher nicht gleichzeitig aktiviert.

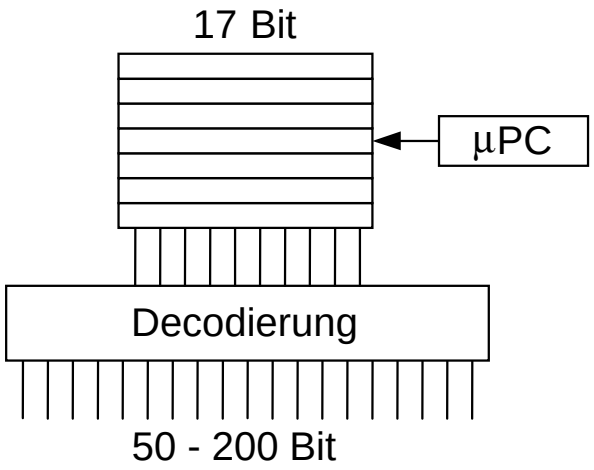
Vertikale Mikroprogrammierung

Die breiten Kontrollwörter eines horizontalen Mikroprogramms sind meist redundant, da nicht alle möglichen Bitkombinationen in einer sinnvollen Steuerung vorkommen.

Jede im Mikroprogramm vorkommende Kombination von Steuersignalen erhält daher eine eigene Kennung.

→ Codierung → Kürzere Kontrollwörter/Mikrobefehle

→ anschließende Decodierungsstufe nötig



Der Mikroprogrammspeicher wird kleiner (und ist etwas schneller auszulesen). Aber es wird zusätzliche (Lauf-)Zeit für die Decodierung (bei jedem Mikrobefehl) benötigt.

Varianten der vertikalen Mikroprogrammierung:

- Codierung nur des Kontrollteils
- Codierung der ganzen Mikrobefehle

Mischform: Quasi-horizontale Mikroprogrammierung

Bei der rein vertikalen Mikroprogrammierung werden die Decoder sehr aufwändig. Mehrere Felder der Mikrobefehle können daher zusammengefasst und für sich vertikal codiert werden.

Sie erhalten dann jeweils eine eigene, aber dafür kleinere (und schnellere) Decodierstufe.

In der Praxis meist diese Art der quasi-horizontalen Mikro-programmierung

→ durch zusätzliche Decodierung zwar etwas langsamer als horizontale, dafür aber kleinere (und schnellere) µ-Programmspeicher

Variables Befehlsformat

Darüber hinaus können Signalgruppen, die nicht gleichzeitig aktiviert werden, ein gemeinsames Feld im Mikroprogramm nutzen. Für ein solches variables Befehlsformat muss eine zusätzliche Kennung ins Mikroprogramm (Indikatorbits) eingefügt werden.

Z. B. kann, anstatt Folgeadresse und Kontrollbits parallel in einem Befehl zu speichern, durch ein Indikatorbit angezeigt wird, ob es sich um einen Steuer- oder Sprungbefehl handelt.

7.6 Mikroprogrammierte Multiplizierer-Kontrolleinheit

7.6.1 RT-Programme und Mikroprogramme

Ein RT-Programm ist leicht in ein Mikroprogramm transformierbar, weil in den einzelnen Zuständen die Kontrollsignale, die Verzweigungsbedingungen und der Folgezustand direkt abgelesen und in einen Mikrobefehl umgesetzt werden können.

Timing und Verzweigungen beachten!

Anmerkungen:

In einem Mikroprogramm sind nur bedingte 2-fach-Verweigungen (Sprung / kein Sprung) möglich, weil der Adressteil nur eine Sprungadresse aufnehmen bzw. spezifizieren kann. D.h. geschachtelte IF-Abfragen müssen durch mehrere Mikrobefehle umgesetzt werden.

Deshalb kann hier keine Mehrfachverzweigung (wie beim Moore-Automat (RSHIFT) aus Kap. 7.2) implementiert werden. Das RT-Programm muss daher partiell umgeschrieben werden.

Beim folgenden Beispiel brauchen die Abfragen in TEST1 bzw. TEST2 deshalb je einen eigenen Takt. Die Multiplikation dauert auch dadurch länger.

Auch aus diesem Grund sind mikrogrogrammierte Steuerwerke i. Allg. langsamer als festverdrahtete.

Beispiel: Umsetzung des Zweierkomplement-Multiplizierers aus Kap. 7.2 mit dem modernen Mikroprogrammwerk aus Kap. 7.5.3

Der Start erfolge mit dem ersten Befehl bei BEGIN.

```
declare register  F, A(7:0), Q(7:0), M(7:0), COUNT(2:0)

declare bus       END,                          # Ext. Kontrollsignal
                  INBUS(7:0), OUTBUS(7:0)

BEGIN:    A <- 0, COUNT <- 0, F <- 0, END <- 0,
          M <- INBUS;                             # c9,c10

          Q <- INBUS;                             # c8

TEST1:    if Q(0) = 0 then goto RSHIFT fi;        # no control action
ADD:      A <- A + M, F <- (M(7) and Q(0)) or F;  # c2,c3,c4
RSHIFT:   A(7) <- F, A(6:0).Q <- A.Q(7:1),
          COUNT <- COUNT + 1 |                    # c0,c1,c11
          if COUNT <> 7 then goto TEST1 fi;

TEST2:    if Q(0) = 0 then goto OUTPUT fi;        # zero

CORRECT:  A <- A - M, Q(0) <- 0;                  # c2,c3,c4,c5

OUTPUT:   OUTBUS <- Q;                            # c6

          OUTBUS <- A;                            # c7

HALT:     END <- 1 | goto HALT;                   # END
```

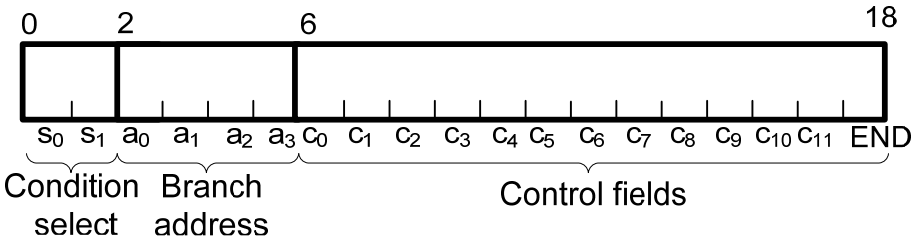
Beachten: Verschachtelte IF-Abfrage nicht umsetzbar, weil nur 1 Sprungziel angegeben werden kann.

Hier Moore-Timing mit **|**-Konstrukt für 2-Phasen-Timing

7.6.2 Multiplikations-Mikroprogramm (horizontales Format)

Für die Kontrolleinheit des Multiplizierers sind verschiedene Mikrobefehlsformate möglich. Hier zunächst horizontales Format.

Horizontales Mikrobefehlsformat



Bedingungen (Condition Select):

1) 00 Keine Verzweigung
2) 01 Verzweigung, falls Q(0) = 0
3) 10 Verzweigung, falls CNT7 = 1
4) 11 Unbedingte Verzweigung

Die Branch-Adresse gibt das Sprungziel bei Verzweigung an, sonst erfolgt die Ausführung des nächsten Mikrobefehls (→ Inkrementieren des µPC).

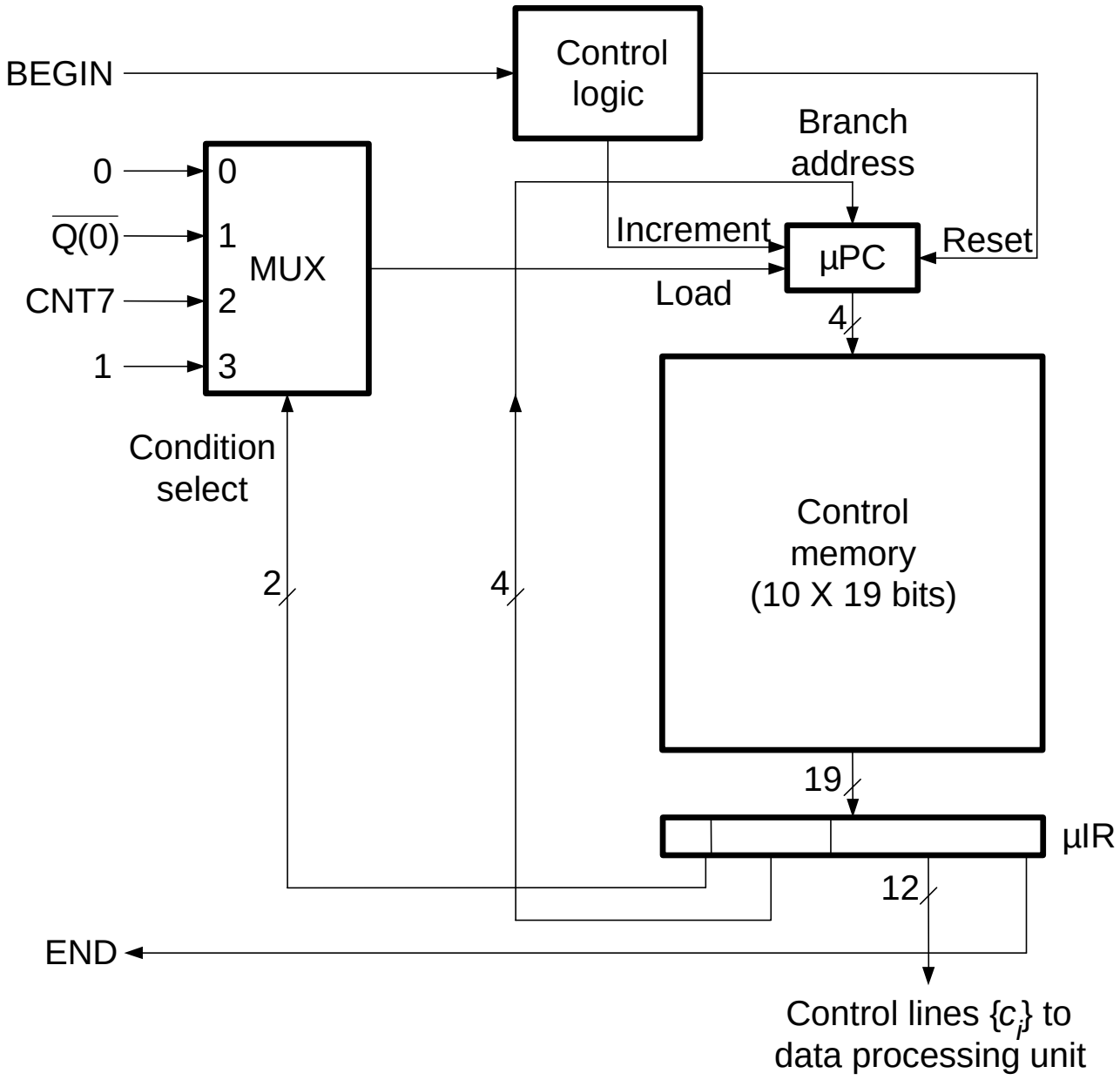
Damit kann das Mikroprogramm aus der RT-Darstellung unmittelbar codiert werden. (In der Praxis meist mittels eigener Mikro-Assembler, die sich flexibel für das jeweilige Mikrobefehlsformat konfigurieren lassen.)

Multiplikations-Mikroprogramm (horizontal codiert)

Microinstruction																
Addr. in CM	Condition select	Branch address	Control fields												END	Comments
			c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}		
0000	00	0000	0	0	0	0	0	0	0	0	0	1	1	0	0	A ← 0, COUNT ← 0, F ← 0, M←INBUS;
0001	00	0000	0	0	0	0	0	0	0	0	1	0	0	0	0	Q ← INBUS;
0010	01	0100	0	0	0	0	0	0	0	0	0	0	0	0	0	**if** Q(0) = 0 **then goto** 4 **fi**;
0011	00	0000	0	0	1	1	1	0	0	0	0	0	0	0	0	A ← A + M, F ← M(7) and Q(0) or F;
0100	10	0010	1	1	0	0	0	0	0	0	0	0	0	1	0	A(7) ← F, A(6:0).Q ← A.Q(7:1), COUNT ← COUNT + 1| **if** COUNT <> 7 **then goto** 2 **fi**;
0101	01	0111	0	0	0	0	0	0	0	0	0	0	0	0	0	**if** Q(0) = 0 **then goto** 7 **fi**;
0110	00	0000	0	0	1	1	1	1	0	0	0	0	0	0	0	A ← A – M, Q(0) ← 0;
0111	00	0000	0	0	0	0	0	0	1	0	0	0	0	0	0	OUTBUS ← Q;
1000	00	0000	0	0	0	0	0	0	0	1	0	0	0	0	0	OUTBUS ← A;
1001	11	1001	0	0	0	0	0	0	0	0	0	0	0	0	1	END ← 1| **goto** 9;

7.6.3 Mikroprogramm-Steuerwerk für den Zweierkomplement-Multiplizierer

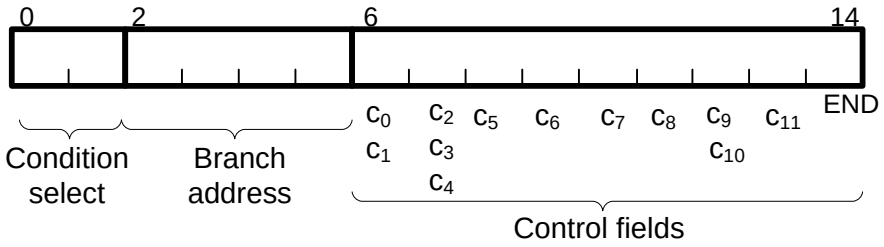
(vgl. ’modernes’ Mikroprogrammwerk aus Kap. 7.5.3 für horizontales Format)



7.6.4 Realisierung mit alternativen Mikrobefehlsformaten

Kompakteres horizontales Format

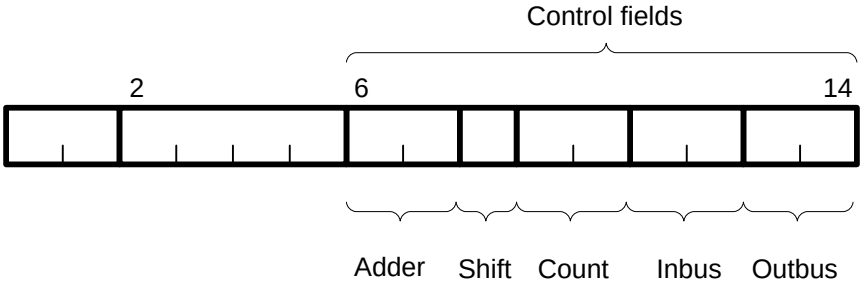
Zusammenfassung von stets gemeinsam auftretenden Mikrooperationen (z. B. c_0, c_1):



Problem:

Es sollten nur solche Mikrooperationen zusammengefasst werden, die auch bei einer evtl. Änderung oder Erweiterung des Mikroprogramms (oder der Hardware) nicht einzeln auftreten müssen (sonst Verlust an Flexibilität).

Quasi-horizontales Format (Gruppenweise Codierung)



Anmerkung:
END-Signal aus Signalkombination generiert

Bei dieser Realisierung ist zwar ein eigener, aber kleiner Decoder für jede Gruppe erforderlich. Aber insgesamt ist die Realisierung günstiger als ein großer Decoder bzgl. Anzahl und Größe der Gatter, Laufzeit und Verdrahtungsaufwand.

Beispiel für die Codierung der Gruppenfelder:

Control field	Bits used	Code	Microoperations specified	Control signals activated
ADDER	6, 7	00	No operation	-
		01	A ← A + M	c_2, c_3, c_4
		10	A ← A – M, Q(0) ← 0	c_2, c_3, c_4, c_5
		11	Unused	
SHIFT	8	0	No operation	-
		1	Right-shift A.Q and load A(7)	c_0, c_1
COUNT	9, 10	00	No operation	-
		01	Clear COUNT, A, F	c_{10}
		10	COUNT ← COUNT + 1	c_{11}
		11	Unused	
INBUS	11, 12	00	No operation	-
		01	Q ← INBUS	c_8
		10	M ← INBUS	c_9
		11	Unused	
OUTBUS	13, 14	00	No operation	-
		01	OUTBUS ← A	c_6
		10	OUTBUS ← Q	c_7
		11	Unused	

Durch das Zusammenfassen von Kontrollsignalen ergibt sich zwar i. allg. ein Gewinn an Speicherplatz für das Mikroprogramm, aber (im Vergleich zum vertikalen Format)

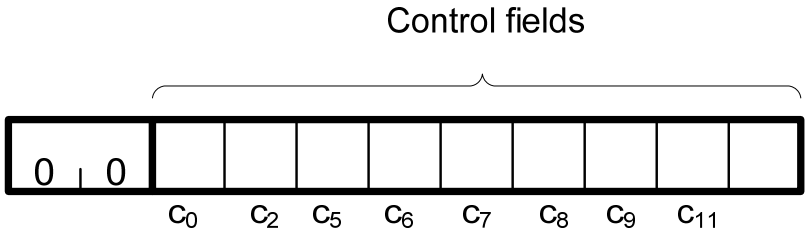
- zusätzlicher Zeitaufwand für die Decodierung der Gruppen,

- Gefahr des Verlustes an Flexibilität.

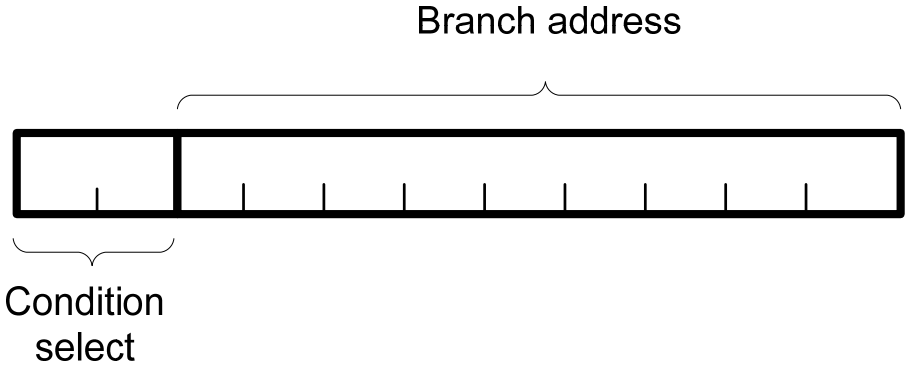
Variables Mikrobefehlsformat

Hier exemplarisch separate Branch-Befehle:

- Format Aktions-Mikrobefehl



- Format Branch-Mikrobefehl



(0 0 No branch; → Aktions-µBefehl vgl. oben)
0 1 Branch if Q(0) = 0
1 0 Branch if CNT7 = 1
1 1 Unconditional branch

Die Branch-Mikrobefehle sind ähnlich zu Sprungbefehlen auf Maschinenprogrammebene. Durch sie werden hier 4 von 15 Bits eingespart.

Aber: Branchbefehle müssen nun in einem eigenen Mikroprogrammzyklus ausgeführt werden und sind nicht mehr mit Aktions-Befehlen kombinierbar.

⇒ Erfordert i. Allg. partielles, aber relativ einfaches Umschreiben der Mikroprogramme, wenn diese in Registertransfer-Notation vorliegen.