Multiplikations-Mikroprogramm mit separaten Branch-Befehlen

Microinstruction												Comments
Addr.	Condition select		Branch address or control bits									
in CM	0	1	0	2	5	6	7	8	9	11	END	
0000	0	0	0	0	0	0	0	0	1	0	0	A ← 0, COUNT ← 0, F ← 0, M ← INBUS;
0001	0	0	0	0	0	0	0	1	0	0	0	Q ← INBUS;
0010	0	1	0	0	0	0	0	0	1	0	0	**if** Q(0) = 0 **then goto** 4 **fi**;
0011	0	0	0	1	0	0	0	0	0	0	0	A ← A + M, F ← M(7) and Q(0) or F;
0100	0	0	1	0	0	0	0	0	0	1	0	A(7) ← F , A(6:0).Q ← A.Q(7:1), COUNT ← COUNT + 1;
0101	1	0	0	0	0	0	0	0	0	1	0	**if** COUNT <> 7 **then goto** 2 **fi**;
0110	0	1	0	0	0	0	0	1	0	0	0	**if** Q(0) = 0 **then goto** 8 **fi**;
0111	0	0	0	1	1	0	0	0	0	0	0	A ← A – M, Q(0) ← 0;
1000	0	0	0	0	0	1	0	0	0	0	0	OUTBUS ← Q;
1001	0	0	0	0	0	0	1	0	0	0	0	OUTBUS ← A;
1010	0	0	0	0	0	0	0	0	0	0	1	END ← 1 ;
1011	1	1	0	0	0	0	0	1	0	1	0	**goto** 11;

Kürzere Mikrobefehle durch Einsparen der Branch-Adresse bei Aktions-Befehlen, dafür aber längerer Ausführungszeit der Programme durch separate Sprungbefehle.

Hier auch anderes Timing beim END-Signal (nicht mehr permanent 1).

7.7 CPU-Kontrolleinheiten

7.7.1 Vorüberlegung

In einem Mikroprogrammspeicher können mehrere Mikroprogramme abgelegt werden.

Die Mikroprogramme können als Interpretation von externen Instruktionen (Maschinenbefehl) aufgefasst werden, wenn sie

- von außen einzeln gezielt aktivierbar sind und
- sie Mikrobefehle und Hardware enthalten, um die Adresse des nächsten auszuführenden Mikroprogramms (externe Instruktion) von außen zu laden.

Mit einem einfachen Steuerwerk (z. B. dem modernen Mikroprogrammwerk) lassen sich dann Mikroprozessoren (CPUs - Central Processing Units) realisieren, die eine ISA-Architektur (*Instruction Set Architecture*) implementieren, die ein Maschinenprogramm als Sequenz von *Maschinenbefehlen* (Instruktionen) in jeweils eine Sequenz von Mikroprogrammen umsetzt.

Prinzipieller Ablauf der (Maschinen-)Befehlsverarbeitung

- Befehlsholphase (*Fetch Cycle*) für Maschinenbefehl
- Befehlsausführungsphase (*Execution Cycle* für µProgramm des Maschinenbefehtls)

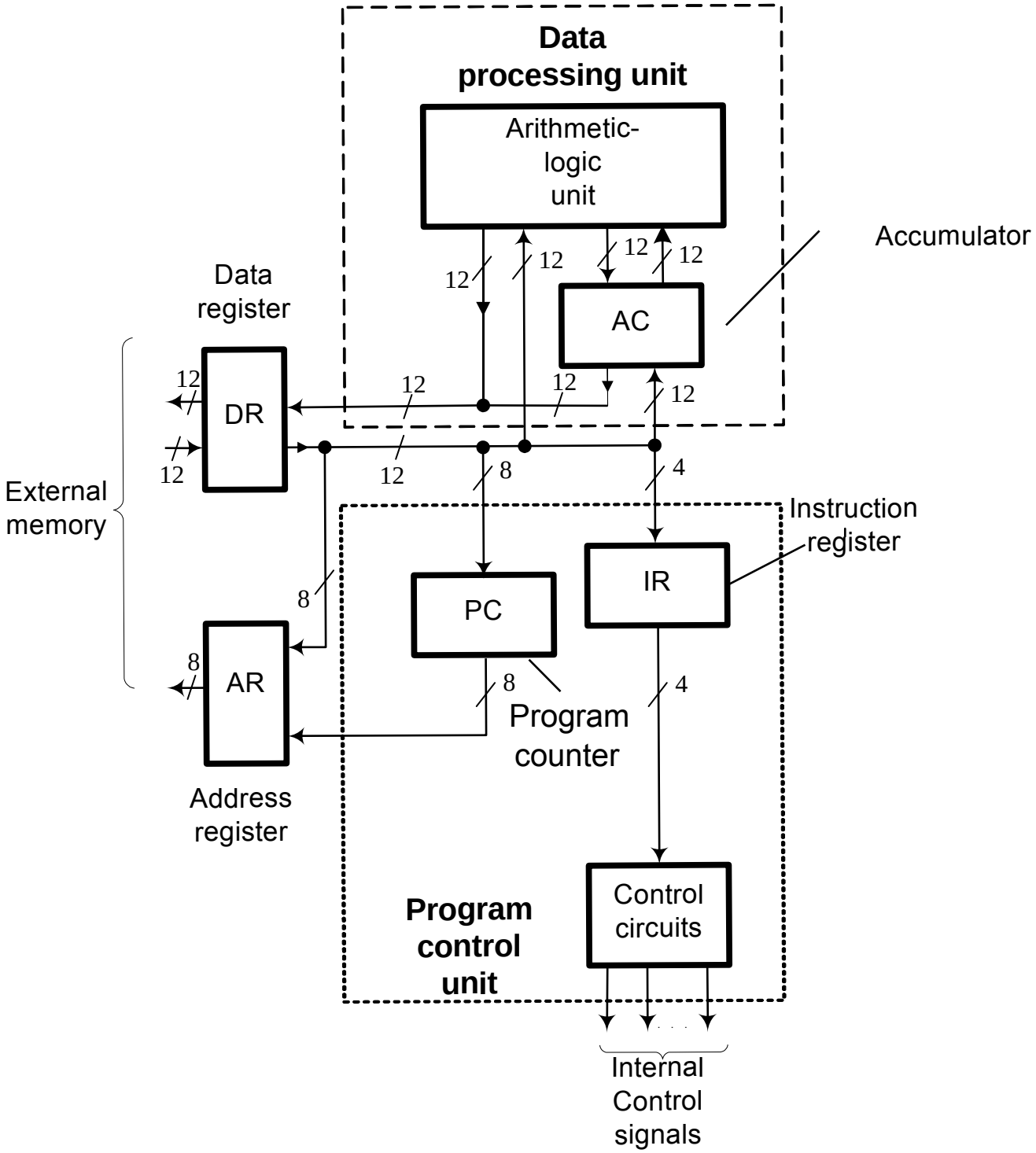
Die Realisierung der beiden Phasen erfolgt zyklisch durch die Kontrolleinheit jeweils über die Ausführung einer Folge von Mikrooperationen.

Hier: Zunächst nur Entwurf der Kontrolleinheit von Interesse

7.7.2 Einfache Beispiel-CPU

Hypothetische Akkumulator-Maschine „HAM“

HAM-Blockschaltbild:



Rechenwerk: Akkumulator AC (Register),
(Data proc. unit) ALU (Arithmetic-Logic Unit)

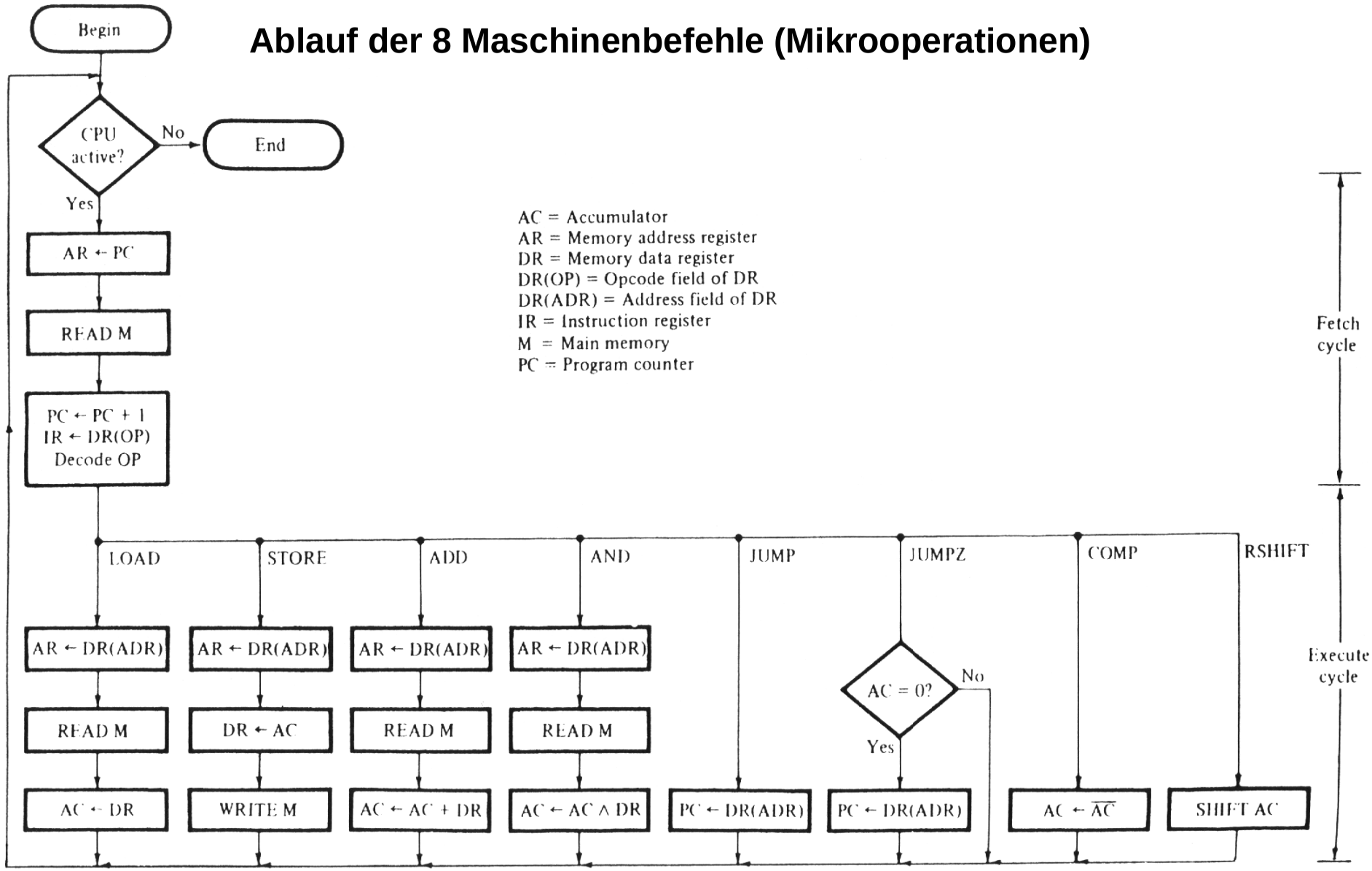
Steuerwerk: Befehlszähler PC,
(Prog. contr. unit) Instruktionsregister IR,
Kontrolleinheit

Speicher- und E/A-Schnittstelle: Datenregister DR,
Adressregister AR
(External memory) als Schnittstelle zum externen Speicher

Befehlssatz der Beispiel-CPU HAM:

Mnemonic	**Opcode**	**Description**	
LOAD X	1	AC ← M(*X*)	(transfer contents of external memory location *X* to the accumulator)
STORE X	2	M(*X*) ← AC	(store accumulator at memory location *X*)
ADD X	3	AC ← AC + M(*X*)	(two´s-complement addition)
AND X	4	AC ← AC $\wedge$ M(*X*)	(logical AND)
JUMP X	5	PC ← *X*	(unconditional branch to *X*)
JUMPZ X	6	**if** AC = 0 **then** PC ← *X*	(conditional branch)
COMP	7	AC ← $\overline{\mathrm{AC}}$	(1-complement accumulator)
RSHIFT	8	Ring-shift accumulator to the right	

Ablauf der 8 Maschinenbefehle (Mikrooperationen)



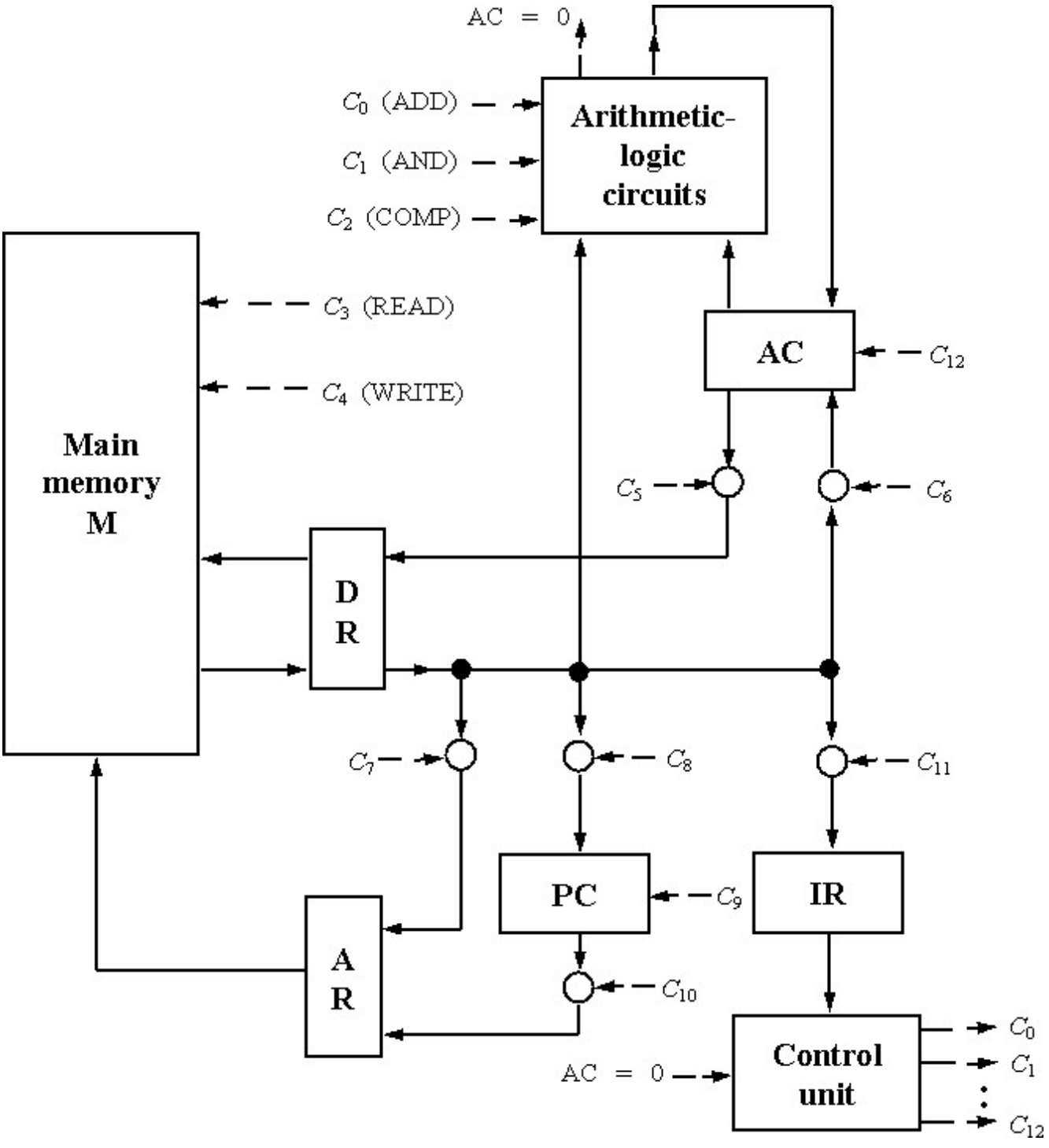
7.7.3 Mikroprogrammierte CPU-Kontrolleinheit

Emulation des Befehlssatzes der Beispiel-CPU durch Satz von Mikroprogrammen in RTeasy

declare register AC(11:0), DR(11:0), AR(7:0), PC(7:0), IR(3:0)
declare memory M(AR,DR)

```
FETCH:  AR<-PC;
        read M;
        IR<-DR(11:8), PC<-PC+1|
        if IR=1 then goto LOAD    else
        if IR=2 then goto STORE   else
        if IR=3 then goto ADD     else
        if IR=4 then goto AND     else
        if IR=5 then goto JUMP    else
        if IR=6 then goto JUMPZ   else
        if IR=7 then goto COMP    else
        if IR=8 then goto RSHIFT
        else goto FETCH  fi  fi  fi  fi  fi  fi  fi  fi;

LOAD:   AR<-DR(7:0);
        read M;
        AC<-DR | goto FETCH;

STORE:  AR<-DR(7:0);
        DR<-AC;
        write M | goto FETCH;

ADD:    AR<-DR(7:0);
        read M;
        AC<-AC+DR | goto FETCH;

AND:    AR<-DR(7:0);
        read M;
        AC<-AC and DR | goto FETCH;

JUMP:   PC<-DR(7:0) | goto FETCH;

JUMPZ:  if AC<>0 then goto FETCH fi;
        PC<-DR(7:0)| goto FETCH;

COMP:   AC<- not AC | goto FETCH;

RSHIFT: AC(11)<-AC(0), AC(10:0)<-AC(11:1) | goto FETCH;
```

Festlegung der Kontrollsignale (Mikrooperationen)



13 Mikrooperationen (Kontrollsignale) c_0, c_1, ..., c_{12}

1 Bedingung (Kriterium AC = 0)

Wirkung der Kontrollsignale

c_0	AC $\leftarrow$ AC + DR	
c_1	AC $\leftarrow$ AC $\wedge$ DR	
c_2	AC $\leftarrow$ $\overline{\text{AC}}$	
c_3	READ M	(DR $\leftarrow$ M(AR))
c_4	WRITE M	(M(AR) $\leftarrow$ DR)
c_5	DR $\leftarrow$ AC	
c_6	AC $\leftarrow$ DR	
c_7	AR $\leftarrow$ DR(ADR)	
c_8	PC $\leftarrow$ DR(ADR)	
c_9	PC $\leftarrow$ PC+1	
c_{10}	AR $\leftarrow$ PC	
c_{11}	IR $\leftarrow$ DR(OP)	
c_{12}	RING-SHIFT AC	(right)

Die Realisierung der Kontrollpunkte erfolgt analog zum Multiplizierwerk.

Eigenschaften

1 Maschinenbefehl = 1 Mikroprogramm
(Fetch-Zyklus zählt mit zum Mikroprogramm)

1 Mikrobefehl = 1 oder mehrere parallel (im gleichen Takt) auszuführende Mikrooperationen (Kontrollsignale)

Jeder Maschinenbefehl hat eine eindeutige Kennung, den *Maschinen*-, *Objekt*- oder kurz *Op-Code* (s. Befehlstabelle und RT-Programm).

Mit den von der Registertransfer-Programmierung bekannten Methoden kann die Anzahl Takte pro Maschinenbefehl und damit auch die Laufzeit von Maschinenprogrammen schon zur Entwurfszeit bestimmt werden.

Einfache Erweiterbarkeit des Befehlssatzes

Beispiel: Maschinenbefehl CLEAR (Clear Accumulator)

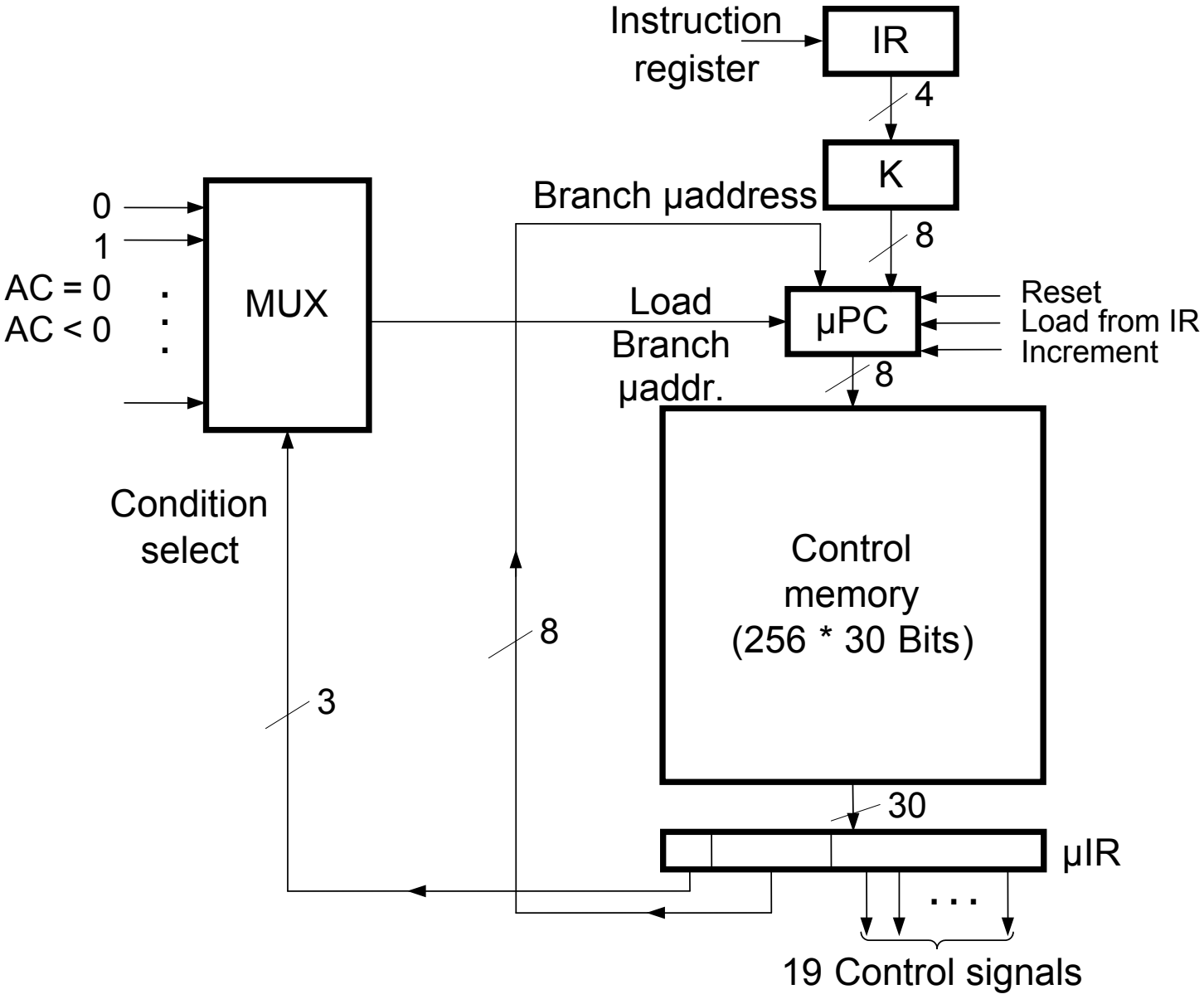
CLEAR: DR ← AC;
AC ← not AC;
AC ← AC and DR | **goto** FETCH;

Hierfür ist wegen der Mikroprogrammierung keine Hardware-Änderung, lediglich eine Erweiterung des Mikroprogramms erforderlich!!!

Mikroprogramm-Kontrolleinheit (mit µPC vgl. Kap. 7.5.3)

Um spätere Erweiterungen des Befehlssatzes zu erlauben, ist eine mächtigere Mikroprogrammsteuerung angezeigt, als für den einfachen Befehlssatz erforderlich wäre.

Erweiterte mikroprogrammierte CPU-Kontrolleinheit



Hier sind wegen der Erweiterbarkeit 19 statt 13 Kontroll-signale und 256 statt 19 Mikrobefehle vorgesehen.

Anmerkungen zur erweiterten CPU-Kontrolleinheit:

Die Mikrobefehle sind hier erweitert auf:

- bis zu 6 Kriterien (3 Bit Bedingungsteil)
- bis zu 19 Mikrooperationen (horizontal codiert)
- bis zu 256 Mikrobefehle (8 Bit µ-Adresse)

Das resultierende Kontrollwort ist damit 30 Bit breit.

Die Umsetzung des Op-Codes aus dem Instruktionsregister IR in die zugehörige Mikroprogramm-Startadresse, die in den µPC geladen wird, erfolgt mittels des Schaltnetzes K (Umcodierer, meist realisiert als ’Mapping ROM’).

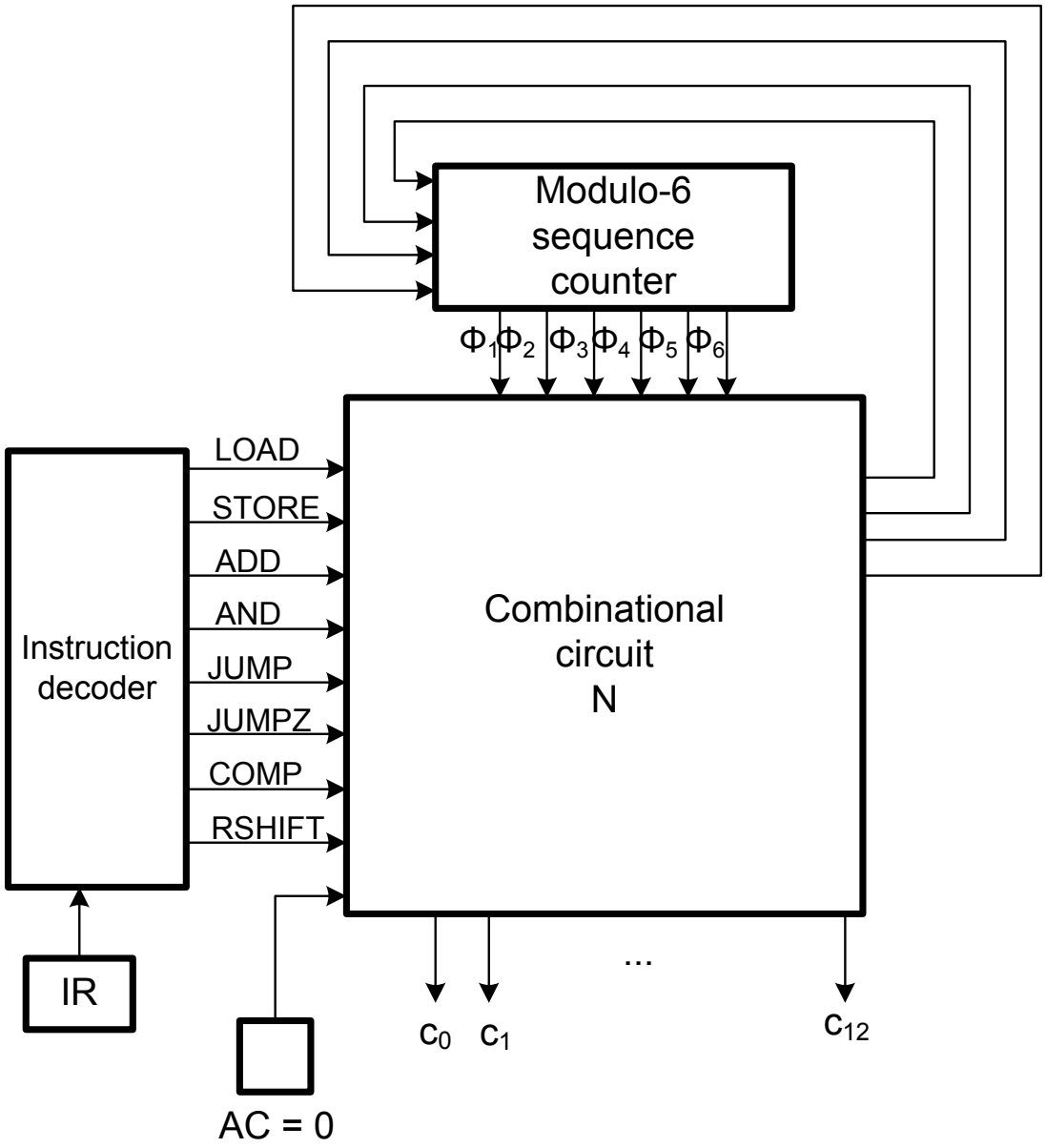
Es ist eine Konvention nötig, an welcher Stelle im externen Speicher der erste auszuführende Maschinenbefehl steht, z.B. an Adresse 0.

Diese Startadresse wird beim Reset automatisch im µPC eingestellt.

7.7.4 Alternative Realisierung als festverdrahtete CPU-Kontrolleinheit

Als alternative Realisierungsmöglichkeit bietet sich aufgrund des zyklischen Ablaufs von Befehlshol- und Befehlsausführungsphase eine Zählersteuerung besonders an.

Beispiel: Zählersteuerung für die Beispiel-CPU



Andere Realisierungen (Automaten, Verzögerungsketten) sind prinzipiell ebenfalls möglich.

Laut Flussdiagramm (Kap. 7.7.2) werden maximal 6 Schritte je Maschinenbefehl benötigt (z. B. LOAD, ADD, STORE).

Annahme: alle Mikrooperationen in einem Takt,
Speicherzugriff braucht Extratakt für READ M und WRITE M.

$\Rightarrow$ Maschinenbefehle brauchen maximal 6 Takte

$\Rightarrow$ Modulo-6 Zähler (Sequence Counter)

Bei Befehlen mit weniger als 6 Takten (z. B. JUMP) kann der Zähler früher zurückgesetzt werden, um Rechenzeit zu sparen.
→ etwas höherer Hardwareaufwand

Entwurf des Schaltnetzes N:

Der *Befehlsdecoder* (instruction decoder) liefert für jeden Maschinenbefehl ein Signal I_m, das mit den Zählersignalen Φ_i gemäß Flussdiagramm so verknüpft wird, dass die richtigen Kontrollsignale c_j für den betreffenden Schritt i erzeugt werden.

Beispiel: Kontrollsignal c_3 (READ M)

Wird erzeugt für $\Phi_2 = 1$ in jeder Befehlsholphase und für $\Phi_5 = 1$ in der Ausführungsphase der Befehle LOAD, ADD, AND, d. h.

$$c_3 = \Phi_2 + \Phi_5 \cdot (\text{LOAD} + \text{ADD} + \text{AND})$$

Allgemein gilt:

$$c_i = \sum_j \left(\Phi_j \cdot \sum_m I_m \right)$$

über alle *j* Takte und *m* Befehle

Bei bedingten Befehlen (z. B. JUMPZ) muss zusätzlich das zugehörige Kriterium K_I (z. B. AC=0) berücksichtigt werden.

$$c_i = \ldots + \Phi_j \cdot I_m \cdot K_I + \ldots$$

Anmerkung: Festverdrahtete Kontrolleinheiten sind zwar i. Allg. schneller, aber inflexibeler als mikroprogrammierte.

Zusätzliche Befehle wie CLEAR würden z. B. aber eine größere Hardware-Änderung in der festverdrahteten Kontrolleinheit erfordern.

7.8 Vorteile der Mikroprogrammierung

- Systematischer Entwurf des Steuerwerks (arbeitet ähnlich einem von Neumann-Rechner, nur dass Mikroprogramme ausgeführt werden).

- Regelmäßige, VLSI-freundliche Struktur (ROMs, PLDs).

- Nachträgliche Verbesserung/Korrektur oder Erweiterung der Ablaufsteuerung leichter durch Änderung der Mikroprogramme möglich (Flexibilität) als bei fest-verdrahteter Logik.

- Mikroprogrammierung sehr ähnlich der Assembler-programmierung (Mikro-Assembler als Werkzeuge zur Übersetzung von Mikroprogrammen).

- Einfache Emulation (= Nachahmung) anderer Maschinen in Hardware.

  ➔ schneller, sicherer, wartbarer, erweiterbarer **Entwurf**

  Mikroprogramme (Firmware) können als Interpretierer für den Maschinenbefehlssatz angesehen werden.

Nachteil:

I. Allg. langsamer (Anzahl und Dauer der Takte) und größerer Chipflächenbedarf als ein festverdrahtetes Steuerwerk

Varianten mikroprogrammierter CPUs

Mikroprogrammierbare Systeme

Bei einem „mikropogrammierbaren“ System ist im Gegensatz zu einem mikroprogrammierten System das Mikroprogramm vom Anwender im fertigen System veränderbar (programmierbar), z.B. um andere Hardware zu emulieren.

➔ Maschienen-Instruktionen durch den Benutzer definierbar.
(vgl. „*mikroprogrammierbare“* Rechner)

Dynamische Mikroprogrammierung:

Umschalten des Instruktionssatzes zur Laufzeit durch Neuladen des µ-Programmspeichers.

Nano-Programmierung

Um die Größe von Mikroprgrogrammspeichern zu reduzieren, werden Mikrooperationen bei der Nano-Programmierung nicht direkt ausgeführt, sondern greifen auf einen zweiten Kontrollspeicher zu (Nano-Programmspeicher), wo für jeden Mikrobefehl wiederum eine Sequenz von Nano-Anweisungen abgelegt ist, die die Hardware direkt kontrollieren (d. h. quasi ’zweistufige’ Mikroprogrammierung).

Die Nano-Programmierung wurde z. B. bei den Mikro-prozessoren Motorola 68000 und 68020 angewendet.