Erweiterungsschaltungen für Mikroprozessoren

Wolfgang Matthes

(c) 1978...1988

Inhalt

1.	Einführung	4
2.	Adressenerweiterungen für Sonderwirkungen	6
2.1.	Zugriffe über ein universelles Bussystem	6
2.2.	Blocktransporte zu E-A- Einrichtungen	6
2.3.	Transporte zur Umcodierung von Information	8
2.4.	Suchoperationen	8
2.5.	Aufteilung bzw. Erweiterung der CPU- Adresse	9
3.	Speicher für Zugriffe zu Einzelbits	12
4.	Beschleunigung von Programmabläufen	16
5.	Erweiterungen für externe Interfaces	19
6.	Erweiterungen des Operationscodes	24
6.1.	Grundlagen	24
6.2.	Ausgabe von Direktwerten	28
6.3.	Ausgabe von Registerbelegungen	30
6.4.	Eingaben in CPU- Register	30
6.5.	Transporte von Speicherinhalten	31
6.6.	Programmverzweigungen	33
6.7.	Weitere Sonderwirkungen	33
7.	Vorkehrungen für den Adressen- Vergleichsstop	35
8.	Rücklesbare Ausgaben	38
9.	Speicheranordnungen mit Fehlerkorrektur	40
9.1.	Fehlerkorrektur durch redundante Speicherung	41
9.2.	ECC- Anordnung für 8-bit- Mikrorechner	44
10.	Speicheranordnungen mit Videoanschluß	46
11.	Vermeidung von Wartezuständen in Mikrorechnern	48
12.	Literaturverzeichnis	51
	Bildteil	55

1. Einführung

Für die Lösung vieler Aufgaben der Prozeßsteuerung, der Datenverarbeitung usw. erscheint es beim gegebenen Stand der Technik naheliegend, Mikrorechneranordnungen einzusetzen.
Das führt oft zu überzeugenden Ergebnissen: die technische Aufgabe kann ohne aufwendige Entwicklung elektronischer Baugruppen durch Nutzung gegebener Funktionseinheiten und deren Programmierung gelöst werden.
Mit steigender Komplexität der Aufgabenstellung wird es zunehmend schwieriger, an einen solchen Lösungsansatz festzuhalten. Nach und nach stellen sich die Grenzen des gegebenen Mikrorechnersystems heraus (Verarbeitungsgeschwindigkeit, Speicherkapazität, E-A- Ausstattung). Dann werden üblicherweise folgende Alternativen erwogen:

1. Einsatz mehrerer Mikrorechner
2. Einsatz bzw. Entwickung einer Mikrorechneranordnung mit hinreichender Ausstattung und hinreichendem Leistungsvermögen
3. Entwicklung spezieller Schaltungsanordnungen.

Hier geht es im Sinne des zweiten Lösungsansatzes darum, das Leistungsvermögen eines Mikrorechners durch Erweiterungen, Zusatzschaltungen usw. für leistungsbestimmende Abläufe deutlich zu verbessern. Dieses Ziel beruht auf folgender Erfahrungstatsache: Wird die komplexe technische Aufgabenstellung in überschaubare Teilaufgaben zerlegt, so zeigt es sich oft, daß für die weitaus meisten Teilaufgaben die gegebene Mikrorechneranordnung in Ausstattung und Leistungsvermögen ohne weiteres ausreicht. Dann ist es sinnvoll zu untersuchen, ob sich die verbleibenden Anforderungen durch zielgerichtete Ergänzungen, Verfeinerungen usw. erfüllen lassen.

Dazu sollen nachfolgend einige Schaltungsvorschläge diskutiert werden, und zwar im einzelnen:

- Adressenerweiterungen für Sonderwirkungen
- Speicher für Zugriffe zu Einzelbits
- Beschleunigung von Programmabläufen
- Erweiterungen für externe Interfaces
- Erweiterungen des Operationscodes
- Vorkehrungen für den Adressen- Vergleichsstop
- Rücklesbare Ausgaben
- Speicheranordnungen mit Fehlerkorrektur
- Speicheranordnungen mit Videoanschluß
- Vermeidung von Wartezuständen.

Es geht dabei keineswegs um autonome Steuerschaltungen, Coprozessoren und dergleichen, sondern stets um vergleichsweise einfache Schaltungsmaßnahmen, die sich grundsätzlich auf elementare Mikrorechneranordnungen (Bild 1) beziehen. Den Daten-, Adressen- und Steuerleitungen einer solchen Anordnung werden entweder Ergänzungsschaltungen beigeordnet, oder an sich gegebene Schaltmittel (z. B. Speicher) werden in besonderer Weise ausgebildet.

Die Schaltungsanordnungen sind auf praktisch alle Mikroprozessor- Typen übertragbar. Hat der betreffende Mikroprozessor beispielsweise ein Multiplex- Bussystem (z. B. Z 8000, 8086), so bereitet es keine Schwierigkeiten, eine Struktur gemäß Bild 1 aufzubauen (Anordnung von Adreß- Latches).
Zur Illustration wird vorzugsweise auf das System Z 80 (U 880) zurückgegriffen. Das hat folgenden Grund: Für den Feinentwurf ist es unerläßlich, das Busprotokoll des Mikroprozessors in allen Einzelheiten zu kennen. Beim Z 80 sind die Abläufe noch recht einfach, und die Schaltungsanordnungen lassen sich vergleichsweise übersichtlich darstellen. Auch ist zu erwarten, daß die Einzelheiten hinreichend bekannt sind (siehe etwa /6/).

Für manche der angesprochenen Aufgaben erscheint es naheliegend, Mikroprozessorsysteme zu wählen, die von vornherein hinreichend leistungsfähig sind. Dazu einige Bemerkungen:

1. Die Wahl hat man nicht immer (Terminvorgaben, Verfügbarkeit, existierende Software).

2. Die Alternativen "einfacher Mikrorechner + Zusatzschaltungen" (z. B. mit 8-bit- CPU) und "Hochleistungs- Mikrorechner" (z. B. mit 32-bit- CPU) sind fallweise mit allen technisch und betriebswirtschaftlich bedeutsamen Tatsachen gegeneinander abzuwägen, namentlich dann, wenn das einfache System für die meisten Teilaufgaben des Einsatzfalles ausreicht (Hochleistungs- Mikrorechner haben vergleichsweise hohe Grundaufwendungen, z. B. für Mehrlagen- Leiterplatten, schnelle Speicherschaltkreise usw.).

3. Die Anforderungen an Einsatzlösungen werden weiterhin wachsen, so daß es auch künftig darum gehen wird, Leistungsgrenzen gegebener Hardware zu überwinden.

2. Adressenerweiterungen für Sonderwirkungen

Liefert der Mikroprozessor eine hinreichend lange Adresse (mehr Adressenbits, als zur Adressierung der Speicherausstattung benötigt werden), so lassen sich Sonderwirkungen recht einfach durch Adressendecodierung ableiten (Bild 2).
Beispiele für solche Sonderwirkungen sind:

1. Zugriffe über ein universelles Bussystem
2. Blocktransporte zu E-A-Einrichtungen
3. Transporte zur Umcodierung von Information
4. Suchoperationen
5. Zugriffe zu einzelnen Bits in Speichern
6. Hardwareseitige Überwachung von Zugriffen zwecks Auslösung von Sonderwirkungen, um die Software zu entlasten (im Sinne der Ablaufbeschleunigung).

Mit Ausnahme der Punkte 5. und 6. (s. dazu Abschnitt 3 bzw. 4) werden nachfolgend Einzelheiten erläutert.

2.1. Zugriffe über ein universelles Bussystem

Es handelt sich darum, daß nicht zu internen Speicheranordnungen des Mikrorechners, sondern zu Einrichtungen zugegriffen wird, die an einen Universalbus angeschlossen sind.
Der Mikrorechner wird dazu als Master am Universalbus wirksam. Grundsätzlich werden durch entsprechendes Decodieren der Adresse Master- Anforderungsschaltungen aktiviert. Wird daraufhin der Buszyklus zugeteilt, so veranlassen autonome Ablaufsteuerschaltungen (Bus- Sequencer) den Datentransport.
Da bei üblichen Bussystemen die Zeit von der Anforderung bis zur Zuteilung nicht vorhersagbar ist, muß die CPU sofort mit Erkennen der Anforderung in einen Wartezustand versetzt werden. Bild 3 gibt einen Überblick über die grundsätzliche Anordnung der Schaltmittel. Einzelheiten sind z. B. aus /8.2/ - /8.7/, /8.12/ - /8.14/, /15/ und /17/ ersichtlich.

2.2. Blocktransporte zu E-A- Einrichtungen

Für den schnellen Datenaustausch mit an sich beliebigen externen Interfaces wird ein Teil des Adressenraumes im Sinne von E-A- Zugriffen decodiert. Zugriffe mit diesen Adressen sind keine Speicherzugriffe, sondern sie aktivieren die E-A- Steuerschaltungen (Bilder 4 - 6; s. auch /8.16/).
Eine solche Lösung ist dann von Vorteil, wenn für das externe Signalspiel ein Übertragungsschema zu verwirklichen ist, für das die üblichen E-A- Schaltkreise nicht ausgelegt sind (etwa ein besonderes Handshake- Signalspiel). Es lassen sich die üblichen Blocktransportbefehle für die Datenübertragung nutzen. Allerdings bereitet es gewisse Schwierigkeiten, auf Sonderbedingungen zu reagieren (z. B. Abbruch der Übertragung durch eine besondere Interface- Schaltfolge, Paritätsfehler usw.). Solche Bedingungen müssen von weiteren Schaltmitteln erkannt werden, die ihrerseits die CPU beeinflussen (Alterna-

tiven: Rücksetzen, NMI, Interrupt). In diesem Zusammenhang ist zu gewährleisten, daß die jeweilige Bedingung programmseitig abgefragt werden kann. Als Beispiel ist in den Bildern 4 - 6 die Signalisierung mittels NMI und die Abfrage der Bedingung über Halteflipflops gezeigt, die an eine PIO angeschlossen sind.
Eine Weiterbildung des Prinzips ist in /18/ beschrieben: bestimmte Belegungen der Adresse bewirken, daß Speicher und E-A-Anschluß gleichermaßen angesteuert werden. Das ermöglicht den Datenaustausch zwischen Speicher und E-A- Anschluß in jeweils einem einzigen Zyklus.
Bild 7 zeigt eine entsprechende Adressenbelegung. Das höchstwertige Adressenbit dient zur Unterscheidung zwischen üblichen Speicherzugriffen und Interface- Zugriffen. Ein weiteres höherwertiges Adressenbit steuert den direkten Datenaustausch zwischen Speicher und Interface (DMA MODE). Weitere, hier nicht näher bezeichnete höherwertige Adressenbits sind bei Interface- Zugriffen für Sonderzwecke (z. B. zur Steuerung des Handshaking) vorgesehen.
Bild 8 gibt eine Übersicht über die grundsätzliche Anordnung. Wesentlich ist die steuerbare Pufferstufe 1 zwischen den Datenleitungen der CPU und dem gemeinsamen Datenpfad von Speicher und Interface. Weiterhin sind eine Interface- Koppelstufe 2, ein Speicher- Ausgangsregister 3 und ein Interface-Schreibregister 4 vorgesehen. Die Schaltmittel 1 - 4 sind in den Bildern 9, 10 im Einzelnen dargestellt.
Bei üblichen Speicherzugriffen (RAM ACCESS) gibt es keine Besonderheiten.
Beim direkten Datenaustausch Interface - Speicher (DMA MODE) ist hingegen zu beachten, daß die CPU sowohl die Speicheradresse liefert als auch die Richtung des Datentransports bestimmt.
Soll vom Interface gelesen werden (Richtung vom Interface zum Speicher), so ist in den Speicher zu schreiben; die CPU muß also einen Schreibzugriff zur betreffenden Adresse ausführen. Soll zum Interface hin geschrieben werden (die Information aus dem RAM wird im Interface- Schreibregister 4 bereitgestellt), so ist der Speicher zu lesen (Lesezugriff der CPU).
Weiterhin ist vorgesehen, daß die CPU direkt zum Interface zugreifen kann (INTERFACE ACCESS = 1; DMA MODE = 0). Dann wird der Speicher nicht angesteuert, und die CPU bestimmt direkt die Richtung des Informationstransports.
Die Hardware ist durch "lineare" Programmstrukturen ("loop unrolling") besonders gut nutzbar. So sind beim Z 80 nachstehende Befehlsfolgen für die Ein- bzw. Ausgabe von Datenblöcken (maximale Länge z. B. 256 Bytes) geeignet:

Eingabe (Interf. -> Speicher)	Ausgabe (Speicher -> Interf.)
LD (HL),A	LD A,(HL)
INC HL	INC HL
LD (HL),A	LD A,(HL)
INC HL	INC HL
...	...

(Die Anfangsadresse in HL ist gemäß Bild 7 aufzubereiten. Man

kann eine beliebige Anzahl von Bytes übertragen, wenn der betreffende Block oft genug aufgerufen wird bzw. wenn gemäß der gewünschten Bytezahl der letzte Teil des Blockes angesprungen wird.)
Damit erfordert die Übertragung eines Bytes 11 Maschinentakte gegenüber 21 Maschinentakten bei Nutzung von Blocktransportbefehlen (z. B. LDIR). Mit einer 4-MHz-CPU beträgt die maximale Datenrate 363 000 Bytes/s.

2.3. Transporte zur Umcodierung von Information

Sind Codewandlungen häufig auszuführen, können Umordnungsnetzwerke im Datenpfad die Leistung deutlich verbessern (vgl. /8.2/).
Solche Netzwerke können über entsprechende Adressendecodierung ein- oder ausgeschaltet werden, wobei sich erforderlichenfalls über weitere Adressenbits oder über ladbare Steuerregister die Art der jeweiligen Wandlung bestimmen läßt.
Die kombinatorischen Funktionen sind mit ROM- Schaltkreisen recht einfach realisierbar. Auch ist die Funktionsumsteuerung unproblematisch (die Bilder 11, 12 zeigen charakteristische Einzelheiten).
Die Speicherzugriffszeit verlängert sich um die Verzögerungszeit (Adresse zu Daten) des jeweiligen Wandlungs- ROM. Werden auch normale Zugriffe (ohne Wandlung) davon betroffen, so kann das von Nachteil sein. Soll dies vermieden werden, ist der Datenpfad so abzuwandeln, daß Zugriffe ohne Umcodierung über schnellere Schaltmittel (z. B. über einfache Koppelstufen) geführt werden (Bild 13). Erforderlichenfalls ist dann dafür zu sorgen, daß bei Zugriffen zwecks Codewandlung zusätzliche Wartezustände eingefügt werden.

2.4. Suchoperationen

Blocksuchbefehle sind für viele Suchaufgaben nicht ohne weiteres nutzbar (Beispiel: Suchen nach bestimmten Bitkombinationen). Ein Ausweg besteht darin, Netzwerke im Datenpfad anzuordnen, die bei erfüllter Suchbedingung ein charakteristisches Bitmuster zur CPU liefern. Damit werden Blocksuchbefehle für solche Durchmusterungsaufgaben nutzbar. Eine Anordnung ist ausführlich in /16/ bzw. /8.16/ dargestellt.
Die konkrete Schaltungsanordnung wird wesentlich durch die Art des Suchkriteriums und die Kompliziertheit der Verknüpfungen bestimmt. Allgemein ist zwischen drei Ausführungsformen zu unterscheiden:

1. Das Suchkriterium ist in der Adresse codiert; es betrifft nur das jeweils gelesene Byte. Es ist mithin eine Codewandlung beim Lesen auszuführen, und die grundsätzliche Anordnung entspricht Bild 11 bzw. Bild 13. Die ROMs im Datenpfad sind dabei auch kombiniert zu Wandlungs- und Suchzwecken nutzbar.
Als Beispiel mag eine Bildschirmansteuerung dienen, die Daten im EBCDIC- Code akzeptieren soll, aber intern einen 7-bit-Code verwendet, wobei das 8. Bit zur Unterscheidung zwischen Feld-

steuerzeichen und darstellbaren Zeichen dient. Es gibt dann zwei leistungsbestimmende Funktionen, nämlich die Codewandlung und das Suchen nach Feldsteuerzeichen.

2. Es sind gesonderte Register vorgesehen, und Suchbedingungen bzw. -kriterien sind vor Beginn des eigentlichen Suchens zu laden. Zusätzlich kann die Adresse weitere Steuerinformation (etwa zur Auswahl eines Suchkriteriums) enthalten (Bild 14; Einzelheiten z. B. in /16/).

3. In Verallgemeinerung des Prinzips werden an sich beliebige Verknüpfungs- bzw. Suchoperationen aus der CPU in zusätzliche Hardware verlagert, wobei die CPU selbst nur noch als Adressenlieferant sowie zur allgemeinen Ablaufsteuerung dient. Die Bilder 15, 16 veranschaulichen das Prinzip, das vielfältig abgewandelt werden kann. Zugriffe mit bestimmten Adressenbelegungen (SPECIAL MODE) veranlassen beispielsweise das Laden von Registern, denen Verknüpfungsschaltungen nachgeordnet sind. Die Verknüpfungsergebnisse sind wie folgt nutzbar:

- Lesen seitens der CPU
- Setzen von Bedingungs- Flipflops, die von der CPU aus abfragbar sind bzw. Sonderwirkungen auslösen (etwa NMI bei erfüllter Suchbedingung; vgl. auch Bild 6)
- direktes Rückschreiben in den Speicher (DMA MODE; vgl. die Bilder 8 - 10).

Die Anordnung kann so ausgebildet werden, daß bei den speziellen Zugriffen das Lesen bzw. Schreiben ausschließlich von der erweiterten Adresse aus gesteuert wird (die Pufferstufe zwischen CPU und Speicher- Datenpfad ist dann zu sperren; s. Bild 16). So lassen sich zeitaufwendige Programmschleifen durch Blockbefehle oder einfache "lineare" Befehlsfolgen ersetzen (Beispiel: byteweises Vergleichen von zwei Speicherbereichen mittels Blocktransportbefehl).

2.5. Aufteilung bzw. Erweiterung der CPU- Adresse

Die vorgeschlagenen Schaltungsanordnungen beruhen darauf, daß mehr Adressenbits zur Verfügung stehen als zur Adressierung der Speichermittel erforderlich sind. Ist dies nicht von vornherein gegeben, muß die Adresse erweitert werden.
Zunächst ist zu klären, ob die gesamte Mikrorechneranordnung ausschließlich dazu vorgesehen ist, mit gegebenen Schaltmitteln und spezifischer Software ein bestimmtes Feld von Aufgaben zweckmäßig zu lösen, ob Standard- Betriebssysteme sowie andere fremde Software lauffähig sein sollen bzw. ob das System für an sich beliebige, zur Entwicklungszeit nicht voraussehbare Erweiterungen offen sein muß.

Im ersten Fall kann man vergleichsweise unbedenklich vorgehen, d. h. die jeweils zweckmäßigste Lösung wählen, sollte allerdings die notwendige Speicherausstattung eher großzügig veranschlagen. Braucht man beispielsweise bei einer Z 80- Konfiguration "garantiert" nur je 4 kBytes ROM bzw. RAM, so ist eine

Adressenaufteilung gemäß Bild 17 praktisch risikolos; man hat für künftige Speichererweiterungen immer noch genügend Reserven. Wenn für die Sonderwirkungen nur wenige Adressenbits ausreichen, ist keine Adressenerweiterung notwendig. Faustregel: Die Hardware sollte so ausgelegt werden, daß sie wenigstens das 4- fache der an sich vorgesehenen Speicherkapazität adressieren kann, auch wenn die höhere Speicherkapazität zunächst nicht installiert wird: - man schafft aber damit gute Voraussetzungen für eine Modernisierung mit der jeweils nachfolgenden Generation von Speicherschaltkreisen.

Bei Forderungen nach Lauffähigkeit fremder Software bzw. nach praktisch unbeschränkten Erweiterungsmöglichkeiten sind zwei Prinzipien nutzbar:

1. Die Adresse wird so großzügig erweitert (z. B. auf 24...32 bit), daß die Sonderfunktionen und alle im Sinne von Kompatibilität erforderlichen Speicheraufteilungen gewissermaßen nebeneinander bestehen können.

2. Es werden Umschaltmöglichkeiten vorgesehen. Dabei wird die Tatsache ausgenutzt, daß E-A- Funktionen im Rahmen von Standard- Betriebssystemen mit Gerätetreiberroutinen gesteuert werden. So kann man z. B. über ein mit E-A- Befehlen ladbares Register bestimmen, ob die Speicheradresse im standardmäßigen Sinne oder im Sinne der Sonderfunktionen von der Hardware zu interpretieren ist. Das erfordert zwei Vorkehrungen:
a) Nach dem Anfangsrücksetzen muß die standardmäßige Interpretation erzwungen werden.
b) In den Treiberroutinen sind die jeweiligen Umschaltungen vorzusehen.

Im folgenden werden einige einfache Schaltungsanordnungen zur Adressenerweiterung vorgestellt.

Die Bilder 18 - 2Ø veranschaulichen die Adressenerweiterung durch einen Segmentadressenspeicher, der über E-A- Befehle geladen werden kann (das Rücklesen ist nicht unbedingt erforderlich, aber schon aus Gründen der Prüfbarkeit zu empfehlen). Die Anordnung gewährleistet die Segmentierung des gesamten Speicheradressenraumes (16 Segmente zu 4 kBytes). Damit läßt sich (bei entsprechender Speicherausstattung) eine Z 8Ø- Konfiguration so einrichten, daß wahlweise beliebige Sonderfunktionen und übliche Standardprogramme (etwa unter CP/M) lauffähig sind.
Wichtig ist, daß beim Anfangsrücksetzen das Segment adressiert wird, das die Startadresse der Rücksetzroutine enthält (Ø bei Z 8Ø; bei 8Ø86 aber FFFFØH). Siehe dazu besonders Bild 2Ø.
Zum Laden der Segmentadresse eignet sich folgender Programmablauf (Übergabe: Nummer des Segmentregisters in der höheren Tetrade von B, Adressenerweiterung in HL):

```
LD C, E-A- Adresse
LD A,L
CPL          -- wegen der Negation am RAM- Ausgang (Bild 2Ø)
OUT A        -- B adressiert Byte im Segment- RAM
```

```
INC C        -- zum 2. Teil des RAM
LD A,H
CPL
OUT A
```

In der Anordnung gemäß Bild 21 - 23 (/18/, /27/) ist ein Teil des an sich vorgesehenen RAM als Segmentadressenspeicher ausgebildet, und die von der CPU gelieferte Speicheradresse dient dazu, zwischen segmentierten und unsegmentierten Zugriffen zu unterscheiden. Das Umladen der Segmentadressen geschieht durch unsegmentierte Zugriffe zum betreffenden RAM- Bereich.

Vorteile:
- kombinierte Nutzung des RAM sowohl als Arbeitsspeicher als auch als Segmentadressenspeicher
- alle Speicherbefehle sind auf die Segmentadressen anwendbar (Erhöhen, Vermindern, Vergleichen usw.)
- keine Schwierigkeiten beim Anfangsrücksetzen.

Nachteil:
- Eingeschränkte Lauffähigkeit fremder Software (wegen der starren Aufteilung der CPU- Adresse).

Wesentlich ist die Auswahlschaltung vor den Adresseneingängen des RAM. Bei unsegmentierten Zugriffen müssen die niedrigstwertigen Adressenbits zum RAM durchgesteuert werden; hingegen sind bei segmentierten Zugriffen die zur Segmentauswahl vorgesehenen höherwertigen Adressenbits durchzusteuern, wobei unbenutzte Bits der RAM- Adrese mit einem Festwert zu beschalten sind (vgl. Bild 23; die 8 Segmentregister liegen im Adressenbereich 4ØFEH, 4ØFFH, 41FEH, 41FFH usw. bis 47FFH).
Weiterhin ist ersichtlich, daß der RAM zwecks hinreichender Adressenverlängerung mit doppelter Aufrufbreite ausgeführt ist, wobei das Adressenbit Ø zur Unterscheidung zwischen beiden Hälften benutzt wird (dies gestattet es, die Adressenerweiterung programmseitig als ein Wort zu behandeln).

In weiterentwickelten Mikrorechnern können solche Schaltungsmaßnahmen die an sich vorgesehenen Adressierungsmechanismen ergänzen. So ist es naheliegend, das Extrasegment beim 8Ø86 für Sonderfunktionen auszunutzen. Das kann aber Schwierigkeiten bringen, wenn beliebige Software lauffähig sein soll. Dann kann z. B. eine Anordnung ähnlich Bild 18 - 2Ø vorgesehen werden, die bei Zugriffen zum Extrasegment aktiviert wird.

3. Speicher für Zugriffe zu Einzelbits

In Mikrorechnerprogrammen geht es häufig darum, einzelne Bits abzufragen oder zu setzen. Sind einschlägige Befehle nicht vorgesehen (wie etwa beim 8086), so müssen logische Verknüpfungen benutzt werden (z. B. AND zum Löschen von Bits, OR zum Setzen von Bits usw.). Wesentlich ist, daß man die Einzelbits nicht adressieren kann, sondern daß jeweils ein entsprechender Binärvektor (Byte bzw. Wort) anzugeben ist.
In anderen Mikroprozessoren sind zwar Bitbefehle vorgesehen, aber die Position des betreffenden Bits (im Byte) ist jeweils direkt im Befehl angegeben (z. B. beim Z 80).

Manche Anwendungen erfordern es, mit Bitadressen arbeiten zu können, die in Listen gespeichert bzw. Rechenoperationen zugänglich sind (Erhöhen, Vermindern, Vergleichen usw.). Solche Abläufe können durchaus leistungsbestimmend sein. Als Beispiel sei ein Serviceprozessor für größere digitale Systeme (etwa EDVA) genannt. In solchen Systemen gehört es zum Stand der Technik, daß automatische Zugriffsmöglichkeiten auf interne Zustände vorgesehen sind. So werden beim bekannten "scan/set"-Prinzip alle Flipflops des Systems zu Schieberegistern zusammengeschaltet, um diagnostische Belegungen einspeisen und Zustände abfragen zu können. Die Zusammenschaltung wird dabei wesentlich von technologischen Gesichtspunkten bestimmt (z. B. durch die Belegung der gate array- Schaltkreise), so daß die Bitströme beim Datenaustausch völlig ungeordnet erscheinen. Der Serviceprozessor hat nun die Aufgabe, die Daten gemäß der logischen Struktur des Systems umzuordnen (im Sinne von Registerinhalten, Belegungen von Steuerflipflops usw.). Das läuft auf das Abarbeiten von Bitadressenlisten hinaus, die die Umordnungsangaben enthalten.
Ein weiteres typisches Beispiel ist der Bildwiederholspeicher eines graphischen Raster- Display. Jedes Bit repräsentiert einen Bildpunkt, und die Geschwindigkeit der wahlfreien Zugriffe zu den einzelnen Bildpunkten bestimmt wesentlich das Leistungsvermögen eines solchen Gerätes mit.

Derartige Aufgaben sind natürlich mit Unterprogrammen lösbar. Das ist aber zeitaufwendig.

Eine vergleichsweise naheliegende technische Lösung besteht darin, höhere (ggf. erweiterte) Adressenbits zu nutzen, um sowohl die Tatsache des Bitzugriffs an sich als auch die Bitadresse (Bit im Byte bzw. Wort) unterzubringen. Zum Abfragen ist dann eine Auswahlschaltung erforderlich, und zum bitweisen Schreiben müssen die Schreibimpulse der Speicherschaltkreise für jede Bitposition einzeln gebildet werden (Bilder 24, 25). Das betreffende Bit wird bei allen Zugriffen in der niedrigstwertigen Position des Datenpfades geführt, die verbleibenden Positionen sind bedeutungslos.

Nachteile:
1. Das Rechnen mit Bitadressen ist nach wie vor umständlich. Jede Bitadresse muß vor dem Zugriff in den höheren Teil der Adresse gebracht werden; das erfordert u. U. das Umladen eines

Segmentregisters, eines Segmentadressenspeichers o. dergl.).

2. Bitweises Schreiben erfordert Speicherschaltkreise mit 1 bit Aufrufbreite; andere Typen sind nicht nutzbar.

3. Maßnahmen zur Fehlerkontrolle (Paritätsprüfung oder ECC) sind nicht nutzbar.

Die Ausführung nach /19/ vermeidet diese Nachteile. Es sind "echte" Bitadressen vorgesehen (Bitauswahl durch die niedrigstwertigen Adressenpositionen), und einzelne Bits werden mit "read-modify-write"- Zyklen in den Speicher geschrieben. Bild 26 zeigt ein Beispiel für die Adressenbelegung. Grundsätzlich ist eine Adresse bereitzustellen, die bei einer für Bitzugriffe vorgesehenen Speicherkapazität von k Worten zu b Bits eine Länge von wenigstens ld k + ld b + 1 Bit hat; letzteres dient zur Unterscheidung zwischen normalen und bitorientierten Zugriffen (BIT ACCESS).

Bild 27 veranschaulicht den Aufbau einer Speicheranordnung. Die Adresse wird dem RAM über eine Auswahlschaltung zugeführt, und zwar bei normalen Zugriffen direkt und bei Bitzugriffen (BIT ACCESS) mit einer Rechtsverschiebung, die dem Zweierlogarithmus der Aufrufbreite entspricht (3 bit beim Z 8Ø). Der Datenpfad der Anordnung ist mit den Datenleitungen der CPU über eine Koppelstufe verbunden, um bei den Schreibzugriffen mit Einzelbits das Durchschalten der CPU- Daten unterdrücken zu können. Die arithmetisch- logische Einheit (ALU) vor den Dateneingängen des Speichers hat 3 Aufgaben:

1. Durchschalten der Datenbytes bei normalen (byteorientierten) Schreibzugriffen
2. Modifizieren des Speicherinhaltes bei bitorientierten Schreibzugriffen
3. Bitabfrage bei bitorientierten Lesezugriffen.

Die grundsätzliche Wirkungsweise bei Bitzugriffen ist folgende:
Es ist ein Decoder vorgesehen, der aus der Bitadresse (Bit im Byte) einen invertierten Binärvektor erzeugt (Beispiel: Bitadresse 2 ergibt Binärvektor 1111 1Ø11). Dieser Vektor sei mit B bezeichnet, der jeweils (mit der verschobenen CPU- Adresse) gelesene Speicherinhalt mit A. Damit werden die Bitzugriffe folgendermaßen ausgeführt:

1. Bit setzen: A v /B; Beispiele:

A	Ø11Ø 1ØØ1	Ø11Ø 11Ø1
B	1111 1Ø11	1111 1Ø11
A v /B	Ø11Ø 11Ø1	Ø11Ø 11Ø1

2. Bit löschen: A & B; Beispiele:

A	Ø11Ø 1ØØ1	Ø11Ø 11Ø1
B	1111 1Ø11	1111 1Ø11
A & B	Ø11Ø 1ØØ1	Ø11Ø 1ØØ1

3. Bit lesen: A v B; Beispiele:

A	Ø11Ø 1ØØ1	Ø11Ø 11Ø1
B	1111 1Ø11	1111 1Ø11
A v B	1111 1Ø11	1111 1111

Beim Lesen führt ein Resultat, das nur aus Einsen besteht, dazu, daß der Wert "1" zur CPU geliefert wird; bei jedem anderen Resultat wird der Wert "Ø" geliefert.
Weitere Einzelheiten sind in Bild 28 dargestellt. Daraus geht hervor, daß die Anordnung mit herkömmlichen Schaltmitteln recht einfach aufzubauen ist. Die folgende Tafel gibt einen Überblick über die einzelnen Funktionen und die zugehörige Ansteuerung der ALU- Schaltkreise.

Funktion	Verknüpfung	MODE CONTROL 3	2	1	Ø
Bytezugriff	F = A.	1	1	1	1
Bit abfragen	F = A v B	1	1	1	Ø
Bit setzen	F = A v /B	1	1	Ø	1
Bit löschen	F = A & B	1	Ø	1	1

Vor der weiteren Funktionserläuterung sollen die Voraussetzungen für die zeitlichen Abläufe veranschaulicht werden (Bild 29): In jedem Speicherzyklus werden zeitversetzte Impulse A, B, C gebildet. Mit A wird der Speicherinhalt in das Ausgangsregister übernommen. B wird aktiviert, wenn das Ergebnis des Bit- Lesens feststeht. Dieses Ergebnis wird mit C auf die niedrigstwertige Datenleitung geschaltet.
Solche zeitversetzten Impulse sind mit Schieberegistern oder auch Laufzeitketten recht einfach zu erzeugen. Dynamische RAMs erfordern derartige Schaltmittel ohnehin (vgl. /8.12/, /17/, /18/, /22/, /26/).

Bei normalen byteorientierten Zugriffen gibt es keine Besonderheiten. Die ALU wird so angesteuert, daß die Datenleitungen vom A- Eingang direkt zum Ausgang durchgeschaltet werden.

Bei bitorientierten Schreibzugriffen sind die Datenwege von CPU und Speicheranordnung voneinander entkoppelt (vgl. Bild 28), so daß der Speicherdatenweg für den "read-modify-write"-Zyklus frei ist.

Wesentlich ist, daß das zu schreibende Bit vom CPU- Datenpfad (also vor der Koppelstufe) entnommen wird. Wert Ø bedeutet "Bit löschen", Wert 1 "Bit setzen".
Das betreffende Byte wird aus dem RAM gelesen und im Ausgangsregister gehalten. Nach der Modifikation über die ALU wird es in den RAM zurückgeschrieben.

Bei bitorientierten Lesezugriffen dient die ALU zur Bitabfrage. Dazu wird deren Vergleicherausgang (A=B) genutzt. Er wird aktiv, wenn das Resultat der Verknüpfungsoperation ausschließlich Einsen enthält.
Der so gelesene Wert wird in einem Halteflipflop gespeichert und am Ende des Zyklus auf die Datenleitung Ø geschaltet. Zum Aufschalten ist ein "open collector"- Gatter vorgesehen (vgl. Bild 28). Vor dem Aufschalten ist das Asgangsregister zu deaktivieren (mit Zeitintervall B; s. Bild 27 u. 29).

Die beschriebenen Funktionen können natürlich auch mit anderen Schaltmitteln ausgeführt werden. So veranschaulicht Bild 3Ø ein kombinatorisches Netzwerk, das die ALU ersetzen kann.

4. Beschleunigung von Programmabläufen

Die effektive Arbeitsgeschwindigkeit eines Programmablaufs wird oft durch dessen innerste Schleife maßgeblich bestimmt (Beispiel: Datenaustausch mit einem externen Interface). Die genauere Untersuchung zeigt, daß eine solche Schleife im wesentlichen drei Aktivitäten enthält:

1. die eigentlichen Operationen mit den zu verarbeitenden bzw. zu transportierenden Daten
2. Adressenrechnungen
3. Endeprüfung und Verzweigung.

In manchen Fällen ist es mit einfachen Schaltungsmaßnahmen möglich, die Software von Teilen der Aktivitäten 2. und 3. zu entlasten und damit die Leistung deutlich zu erhöhen.
Als Beispiel, das Anregung zu vielfältigen Abwandlungen geben soll, sei eine Anordnung für die interruptgesteuerte Datenübertragung (/24/) näher beschrieben.
Das Blockschaltbild (Bild 31) zeigt zunächst eine elementare Mikrorechneranordnung, die über einen E-A- Schaltkreis (PIO) mit einem externen Interface verbunden ist.
Die Aufgabe, die dem Beispiel zugrunde liegt, besteht darin, daß vom Interface fortlaufend Bytes angeboten werden, die in den RAM zu schreiben sind, und zwar in einen Bereich von 192Ø Bytes Länge. Bei Überschreiten dieses Bereichs ist eine besondere Maßnahme auszulösen (Beispiele: Übertragung abbrechen oder auch Übergang zum Anfang des Speicherbereichs - "wrap around"). Das Interface beruht auf einem Handshake- Signalspiel mit unvorhersagbaren Antwortzeiten der sendenden Einrichtung, und der Mikrorechner soll zusätzlich (zeitmultiplex) weitere Aufgaben übernehmen. Deshalb ist die Bedienung des Interfaces als Interrupt- Routine vorgesehen. Bild 32 zeigt eine typische Befehlsfolge mit den Ausführungszeiten (Anzahl der CPU- Takte). Die einzelnen Aktivitäten haben folgende Anteile:

1. Datenübertragung:	17 Takte
2. Erhöhen der Adresse:	6 Takte
3. Test auf Bereichsüberschreitung:	62 Takte
4. Eintritt und Rückkkehr:	34 Takte.

Der Bereichstest, der nur nach jeweils 192Ø übertragenen Bytes zu einer Modifikation des Programmlaufs führt, hat an der gesamten Laufzeit mehr als 5Ø% Anteil!

Gemäß Bild 31 wird diese Aktivität in die Hardware verlagert. Mit einer einfachen Erkennungsschaltung wird die Speicheradresse überwacht. Bei einem Zugriff mit einer Adresse außerhalb des Bereichs wird ein NMI ausgelöst.
Bild 33 zeigt ein Beispiel für die Adressenstruktur. Im konkreten Fall gestaltet sich die Erkennungsschaltung besonders einfach. Die Bereichsüberschreitung kann nur durch +1- Zählung ausgelöst werden (von 1919 auf 192Ø bzw. von 7FFH auf 78ØH). Der Speicherbereich beginnt an einer 2k- Grenze (Anfangsadresse der Form Øxxx xØØØ ØØØØ ØØØØ). Um die Bereichsüberschrei-

tung zu erkennen, reicht es deshalb völlig aus, die Adressenbits 10 - 7 konjunktiv zu verknüpfen. Adressenbit 15 kennzeichnet die Erlaubnis zur NMI- Auslösung.
Mit diesen Voraussetzungen ist ein Programmablauf nach Bild 34 möglich, der im Vergleich zu Bild 32 weniger als 50% der Laufzeit erfordert.
Weiterhin erlaubt die Anordnung ohne besondere Vorkehrungen das zeitmutiplexe Betreiben von drei derartigen Interfaces (bei entsprechender Ausstattung mit Speichern und E-A- Schaltkreisen). Es sind dann drei Abläufe ähnlich Bild 34 vorzusehen, wobei jeweils eines der Register BC, DE, HL als Adressenregister verwendet wird. Die einzelne Routine ist in sich nicht unterbrechbar, so daß der NMI- Ablauf ohne weiteres feststellen kann, welches Interface zu behandeln ist (Vergleich der Registerbelegungen mit den zugehörigen Endadressen). Ein herkömmlicher Ablauf würde gegenüber Bild 32 noch länger dauern, da bei mehreren Interfacefunktionen die Register über den Stack gerettet werden müßten.
Für die allgemeine Nutzung und Abwandlung des Schaltungsprinzips gelten folgende Grundsätze:

1. In den leistungsbestimmenden Schleifen sind jene Testabläufe zu bestimmen, die anteilig viel Laufzeit benötigen, aber in den weitaus meisten Schleifendurchläufen keine Sonderwirkungen veranlassen.

2. Um zu erkennen, ob die Sonderwirkungen auszulösen sind, werden Schaltmittel vorgesehen.

3. Diese Schaltmittel werden so angeordnet, daß sie einen höher priorisierten Interrupt auslösen, sobald eine der Sonderbedingungen auftritt. Der Interrupt muß sofort wirksam werden können, d. h. praktisch: NMI oder Auslösung über E-A-Schaltkreis mit höherer Priorität (die E-A- Routine muß dann aber selbst unterbrechbar sein).

4. Bei mehreren Sonderbedingungen muß es möglich sein, die jeweils wirksame Bedingung programmseitig bestimmen zu können (z. B. über abfragbare Flipflops).

5. Es sollte programmseitig steuerbar sein, ob erkannte Sonderbedingungen wirksam werden oder nicht, z. B. über höherwertige (ggf. erweiterte) Adressenbits (vgl. Bilder 33, 34) oder auch über ladbare Steuerregister o. ä.

6. Der Effekt der Maßnahmen läßt sich durch probeweises Programmieren und Auszählen der Maschinentakte ermitteln. Es sei
- t_S die Laufzeit der Schleife ohne Hardwaremaßnahmen, wenn keine Sonderbedingung wirksam ist
- t_W die Laufzeit der Schleife ohne Hardwaremaßnahmen bei wirksamer Sonderbedingung
- t_h die Laufzeit der Schleife mit Hardware- Unterstützung
- t_I die Laufzeit der Interruptroutine zur Behandlung der Sonderbedingungen
- n_S die Anzahl der Durchläufe ohne Sonderwirkungen
- n_W die Anzahl der Durchläufe mit Sonderwirkungen.

Die Hardware- Maßnahmen sind effektiv, wenn gilt

$$n_s \cdot t_h + n_w \cdot t_i \ll n_s \cdot t_s + n_w \cdot t_w .$$

Ein verallgemeinerter Einsatzfall ist in Bild 35 dargestellt. Zur Erkennung von drei Sonderbedingungen sind drei Decodierschaltungen vorgesehen, denen jeweils ein programmseitig abfragbares und löschbares Flipflop nachgeordnet ist. Alle Bedingungen veranlassen die Auslösung eines NMI. Für jede Bedingung ist eine gesonderte Erlaubnisleitung vorgesehen, die von erweiterten Adressen oder ladbaren Steuerregistern geschaltet werden kann.

5. Erweiterungen für externe Interfaces

Bitserielle Interfaces und solche, deren Breite der Verarbeitungsbreite des Mikrorechners angepaßt ist, bieten keine grundsätzlichen Schwierigkeiten. Sie können in den weitaus meisten Fällen über E-A- Schaltkreise angeschlossen werden. Es gibt aber auch Einsatzfälle, wo die Interfaces weder bitseriell noch byte- oder wortorientiert sind. Üblicherweise wird dann eine entsprechende Anzahl programmierbarer E-A-Schaltkreise vorgesehen. So zeigt Bild 36 eine Mikrorechneranordnung mit einem einwärts- und einem auswärtsführenden Interface von jeweils bis zu 16 zusammengehörenden Informationsleitungen. Für eine Ausgabe ist beispielsweise folgender Ablauf notwendig:

```
LD A, niederes Byte
OUT PIO 1 PORT A
LD A, höheres Byte
OUT PIO 1 PORT B
```

Dabei stehen die LD- Befehle für an sich beliebige Transporte, um die betreffenden Bytes heranzuschaffen. Handelt es sich um Speicherzugriffe der Form LD A, (HL) o. ä., so erfordert der Ablauf 46 CPU- Takte. Zu beachten ist, daß die Interfaceleitungen byteweise nacheinander erregt werden. Bei manchen Interfaces ist das nicht zulässig, so daß zusätzliche hardware- und programmseitige Maßnahmen erforderlich sind (z. B. ein Zwischenregister ausreichender Breite, dessen Ladetakt von einem weiteren E-A- Befehl ausgelöst wird).

Die Anordnung nach /25/ vermeidet solche Schwierigkeiten von vornherein und erlaubt zudem eine deutlich höhere Geschwindigkeit.
Gemäß Bild 37 kann eine 8-bit-Mikrorechner- Konfiguration auswärtsführende Interfaces mit bis zu 24 zusammengehörenden parallelen Interfaceleitungen steuern, und es können einwärtsführende Interfaces mit bis zu 16 parallelen Informationsleitungen angeschlossen werden.

Das Prinzip der Ausgabe beruht darauf, daß die Adressenleitungen wenigstens teilweise als Datenleitungen mitbenutzt werden. Bei bestimmten Transportbefehlen des Typs Register - Speicher (z. B. LD (HL), A) stammt sowohl die Adresse als auch das Datenbyte aus programmseitig ladbaren Registern der CPU, und während des eigentlichen Speicherzugriffs sind diese Werte auf den Daten- und Adressenleitungen der CPU für externe Schaltmittel zugänglich. Sie sind mithin für Ausgabezwecke nutzbar, wenn folgende Vorkehrungen getroffen werden:

1. Die Hardware muß erkennen, daß die betreffende Wirkung gewünscht ist.
2. Das Schreiben in den Speicher muß verhindert werden.
3. Statt in den Speicher muß die Information in ein Interfaceregister übernommen werden (im Beispiel bis zu 24 Bit parallel!).

Grundsätzlich kann man mit einem Mikroprozessor, der a Adressenbits liefert und eine Verarbeitungsbreite von b Bits hat, parallele Interfaces mit maximal a + b zusammengehörenden Informationsleitungen ansteuern.

Das Prinzip der Eingabe beruht darauf, daß die Interfacebelegung parallel durch autonome Schaltmittel übernommen wird und daß die CPU diese Belegung abschnittsweise über die Datenleitungen liest. Dazu werden bestimmte Befehle ausgenutzt, die einen Direktwert entsprechender Länge abschnittsweise aus dem Speicher in ein CPU- Register übertragen (z. B. LD HL, nm). Das erfordert:

1. Die Hardware muß erkennen, daß die betreffende Wirkung gewünscht ist.
2. Nach dem Lesen des Operationscodes sind alle Speicherzugriffe zu unterdrücken. Bis zum Ende des Befehls darf kein Speicherausgang am Datenbus der CPU aktiv sein.
3. Bei jedem der Zugriffe zum Direktwertlesen wird eines der Interfaceregister auf den Datenbus geschaltet. Der Speicherinhalt wird ignoriert, stattdessen gelangt die Interfacebelegung in das CPU- Register.

Grundsätzlich kann ein paralleles einwärtsführendes Interface soviele zusammengehörende Informationsleitungen haben, wie Datenbits mit Direktwert- Ladebefehlen in ein programmseitig zugängliches Register der CPU transportiert werden können.

Die entscheidende Aufgabe für den Entwurf der zusätzlichen Hardware besteht darin, die Befehle zu erkennen, die die jeweils gewünschten Wirkungen veranlassen sollen. Grundsätzliche Lösungen sind:

1. programmseitige Modussteuerung (z. B. mittels Steuerregister)
2. Adressendecodierung, ggf. zusammen mit Adressenerweiterung
3. Beobachtung des Befehlsstromes
4. Erweiterung des Operationscodes.

Die letztgenannte Lösung wird in Abschnitt 6 ausführlich vorgestellt. Deshalb werden nachfolgend nur die ersten drei Punkte erläutert.

1. Programmseitige Modussteuerung

Es ist eine generelle Erlaubnisleitung vorgesehen, beispielsweise als Ausgangsleitung eines programmseitig ladbaren Steuerregister oder einer PIO (in den Darstellungen ab Bild 39 ist diese Leitung mit MODE CONTROL bezeichnet).
Die Vorkehrung ist auch bei allen anderen Lösungen eine praktische Notwendigkeit: allein um den Mikrorechner prüfen zu können, muß es möglich sein, die Sonderfunktionen außer Kraft zu setzen.
Ob diese Maßnahme allein ausreicht, hängt davon ab, ob die

zusätzliche Laufzeit (der Umschaltbefehle) tragbar ist.
Man braucht nur einen einzigen zusätzlichen Befehl pro Interfacezugriff, wenn man die Konvention einführt, daß der Zugriffsbefehl für das Interface dem Umschaltbefehl unmittelbar nachfolgen muß. Dann wird beim Lesen des übernächsten Befehls die Erlaubnis automatisch zurückgenommen. Als Umschaltbefehl eignet sich eine Ausgabe mit einer bestimmten Adresse, wobei die ausgegebenen Daten ignoriert werden (11 zusätzliche Maschinentakte beim Z 8Ø). Wesentlich ist, daß alle Unterbrechungsanforderungen zwischen Umschalt- und Zugriffsbefehl hardwaremäßig blockiert werden. Bild 38 zeigt eine entsprechende Schaltung.

2. Adressendecodierung bzw. -erweiterung

Für Ausgabezwecke wird die Lösung recht einfach, wenn die CPU-Adresse mehr Bits hat als auszugeben sind und als zur Speicheradressierung benötigt werden.
Ist z. B. bei einer Z 8Ø- Konfiguration der Speicheradressenraum auf 32 kBytes beschränkt, so ist das höchstwertige Adressenbit zur Steuerung der Sonderfunktionen nutzbar. Das Interface kann dann 15 - 23 Leitungen umfassen (letzteres bei Mitnutzung des Datenpfades). Bild 39 zeigt eine solche Adressenstruktur.
Beim 8Ø86 wäre es denkbar, den Adressenraum des Extrasegmentes für die Ausgabesteuerung zu nutzen; man könnte dann bis zu 32 bit mit einem Befehl ausgeben.
Für Eingabezugriffe ist die Adressendecodierung nur dann geeignet, wenn es höherwertige Adressenbits gibt, die bei Befehlszugriffen nicht genutzt werden. Dann wäre beispielsweise beim Z 8Ø die Adressenstruktur gemäß Bild 39 weiterhin nutzbar (man muß erforderlichenfalls nur dafür sorgen, daß bei Befehlslesen das Adressenbit 15 am Speicher auf Ø gehalten wird).
Durch Decodieren solcher Adressenbits läßt sich eine Ablaufsteuerung aktivieren, die in den verbleibenden Zyklen des Befehls wirksam ist.
Bild 4Ø zeigt ein Beispiel für die Steuerschaltungen. Diese sind für Ein- und Ausgabezyklen vorgesehen (die Abläufe sind in den Bildern 41, 42 dargestellt). Die Schaltung gemäß Bild 4Ø interpretiert bei aktivem Erlaubnissignal SPECIAL ACCESS ENABLE den folgenden Befehl als einen Befehl mit Sonderwirkungen (SPECIAL ACCESS). Daraufhin wird bei einem folgenden Schreibzugriff nicht in den Speicher, sondern in das Interfaceregister geschrieben. Ist der dem Operationscodelesen folgende Zugriff ein Lesezugriff, wird angenommen, daß es sich um einen Direktwert- Lesebefehl mit Eingabefunktion handelt. Deshalb wird der Speicher für diesen und den nachfolgenden Zugriff deaktiviert (MEMORY DISABLE). Stattdessen werden nacheinander die Interface- Eingaberegister auf den Datenbus geschaltet (ENABLE LO INPUT, ENABLE HI INPUT). Dazu sind Vorkehrungen notwendig, die Bild 43 veranschaulicht (bei allen Sonderzugriffen sind alle Speicher zu deaktivieren).
Die Schaltung gemäß Bild 4Ø kann sowohl durch Umschaltbefehle (Bild 38) als auch durch Adressendecodierung (Bild 39) ange-

steuert werden.

Bei Steuerung von Eingabezugriffen mittels Adressendecodierung ist der Befehlszähler zu modifizieren (wegen des Zeitverlustes stellt dies einen gewissen Nachteil dar). Dazu eignet sich z. B. folgenden Ablauf:

```
          ...
          ...                  -- Befehle ohne Sonderwirkung

          JMP SW+8000H         -- Adressenbit 15 setzen
SW:       ...                  -- Programmstück, in dem
          ...                  -- Befehle mit Eingabezugriffen
          ...                  -- vorkommen
          ...
          JMP NORMAL           -- Sonderwirkung ausschalten
NORMAL:   ...                  -- Fortsetzen ohne Sonderwirkung
```

Die Umschaltverluste lassen sich praktisch vermeiden, wenn die Eingabefunktionen in Unterprogramme eingebaut werden, in denen die betreffenden Befehlstypen ausschließlich für Sonderfunktionen genutzt werden. Dann reicht ein CALL mit der erhöhten Anfabgsadresse (+ 8000H) zum Setzen von Adressenbit 15, und der RETURN- Befehl gewährleistet automatisch das Zurückschalten.

3. Beobachtung des Befehlsstromes

Es ist ein Steuerautomat vorgesehen, der den Datenbus der CPU ständig überwacht und beim Holen der betreffenden Befehle die Sonderwirkungen auslöst. Die besondere Schwierigkeit besteht darin, daß es sich um Befehle handelt, die auch für allgemeine Programmabläufe recht nützlich sind. Deshalb wäre es sehr nachteilig, diese Befehle nicht freizügig anwenden zu können. Bei Mikroprozessoren, die viele Universalregister haben, kann man zeitweilig (gesteuert über eine programmseitig schaltbare Erlaubnisleitung) bestimmte Operationscodes mit bestimmten Registeradressen für die Interfacezugriffe reservieren. Da solche Mikroprozessoren (z. B. Z 8000) bei jedem Speicherzugriff eine umfassende Zustandsinformation liefern, gestaltet sich die zusätzliche Hardware recht einfach (Grundprinzip: Decodierung der betreffenden Operationscodes einschließlich der Registeradressen, dann sinngemäß weiter ähnlich Bild 40).

Ansonsten bietet ein Steuerautomat die Möglichkeit, das befehlsweise Umschalten (vgl. Bild 38) zu verkürzen, indem etwa anstelle eines Ausgabebefehls ein NOP- Befehl vorgesehen wird (/25/). Damit verlängert sich beim Z 80 der Interfacezugriff um lediglich 4 Maschinentakte. Bild 44 veranschaulicht das Prinzip, Bild 45 zeigt die Zustandsübergänge für die Befehlsanalyse beim Z 80.

Vorteile:

1. Der Speicheradressenraum ist ohne Einschränkung nutzbar.
2. Die Adresse steht in voller Länge für Ausgabezwecke zur Verfügung.
3. Gar keine oder nur geringe Umschaltzeiten.
4. Größere Funktionssicherheit: nur Befehle, die ausdrücklich für Interfacezugriffe vorgesehen sind, üben auch Sonderwirkungen aus. Im Gegensatz dazu wirken in den Anordnungen nach Bild 38 - 40 verschiedene nicht ausdrücklich zugelassene Befehle bei erlaubten Interfacezugriffen in unvorhersagbarer Weise (Programmierfehler, mit denen immer gerechnet werden muß, führen zu schwerwiegenden Fehlfunktionen).
5. Nutzung weiterer Befehlsabläufe für Interfacezugriffe. So kann beim Z 80 eine Folge NOP; LD A,(nm) zur Ausgabe des Direktwertes nm auf die Adressenleitungen genutzt werden (in Bild 45 bereits berücksichtigt).

Diesen Vorteilen steht allerdings ein nicht unbeträchtlicher Aufwand gegenüber.

6. Erweiterungen des Operationscodes

6.1. Grundlagen

Die Befehlsliste einer Mikroprozessor- CPU ist als unabänderlich gegeben hinzunehmen. Sollen zusätzliche Wirkungen erzielt werden, so ist dies nur im Rahmen des gesamten Mikrorechner-Komplexes möglich, beispielsweise über ladbare Steuerregister, E-A- Schaltkreise, Coprozessoren oder andere zusätzliche Schaltmittel.

Hier soll ein weiteres Prinzip beschrieben werden (/23/). Es besteht darin, daß durch zusätzliche Speichermittel der Operationscode der Maschinenbefehle erweitert wird. Diese Erweiterungen können natürlich nur außerhalb der CPU zur Wirkung kommen, sie können aber den Programmablauf in der CPU beeinflussen: durch Belegen des Datenbus, Modifizieren der Adresse, Auslösen von Interrupts usw.
Bild 46 zeigt das Blockschaltbild eines solchen Mikrorechners. Neben Speichern in üblicher Anordnung und Schaltmitteln zum Anschluß externer Interfaces ist zusätzlich ein Steuerspeicher mit nachgeordnetem Steuerregister vorgesehen. An dieses sind Steuerschaltungen angeschlossen, die Steuerleitungen für verschiedene Sonderwirkungen erregen, etwa zum Betreiben externer Interfaces oder auch zum Aufschalten bestimmter Belegungen auf den Datenbus.
Der Steuerspeicher wird genauso adressiert wie alle anderen Speichermittel. Sonderwirkungen werden nur bei Befehlszugriffen zu speziellen Adressenbereichen ausgeübt. Dann wird die Belegung des Steuerspeichers in das Steuerregister übernommen. Der Steuerspeicher kann in beliebiger Weise ausgeführt sein (ROM/RAM) und eine an sich beliebige Aufrufbreite haben. Sind RAMs vorgesehen, so müssen sie geladen werden können (z. B. im Rahmen von Initialisierungsabläufen). Es ist vom Prinzip her nicht notwendig, den Steuerspeicher von der CPU aus lesen zu können, aber entsprechende Vorkehrungen sind schon aus Gründen der Prüfbarkeit zweckmäßig (z. B. für Testprogramme, die den Speicherinhalt kontrollieren).
Bild 47 gibt einen ersten Überblick über eine so ausgelegte Speicheranordnung. Neben dem Steuerspeicher ist jener Speicher dargestellt, der die zu erweiternden Befehle enthält, sowie ein Adressendecoder für Lesezugriffe. Hier sind drei Adressenbereiche von Bedeutung:

1. Adressenbereich a für übliche Zugriffe zum dargestellten Befehlsspeicher
2. Adressenbereich b für Zugriffe mit Befehlserweiterung
3. Adressenbereich c für übliche Zugriffe zum Steuerspeicher, der dann als normaler ROM bzw. RAM von der CPU angesprochen wird.

Bei Zugriffen zum Adressenbereich b werden die aus dem Befehlsspeicher gelesenen Befehle mit Steuerworten aus dem Steuerspeicher erweitert. Diese Betriebsweise heißt im folgenden CONTROL MODE. Das entsprechende Steuerflipflop wird bei jedem

Lesen eines Operationscodes ein- oder ausgeschaltet, je nachdem, mit welcher Adresse der Zugriff ausgeführt wird.
Bei Zugriffen zum Adressenbereich b sind beide Speicher zu aktivieren (disjunktive Verknüpfungen vor den "chip enable"-Eingängen), aber nur der Befehlsspeicher darf auf den Datenbus geschaltet werden.
Jeder Lesezugriff zum Adressenbereich b veranlaßt, daß die adressierte Belegung des Steuerspeichers in das Steuerregister übernommen wird.
Ist CONTROL MODE inaktiv (d. h. der Operationscode wurde nicht vom Adressenbereich b gelesen), so wird das Steuerregister zurückgesetzt. Dabei geht es um folgendes:
Es muß eine Belegung des Steuerregisters definiert werden, die - auch bei aktivem CONTROL MODE - keinerlei Sonderwirkung veranlaßt (im folgenden: NO-OP- Modus). Diese Belegung muß bei ausgeschaltetem CONTROL MODE hardwaremäßig erzwungen werden (das ist zweckmäßigerweise eine Belegung ØØ..ØH bzw. FF..FH).
Bild 48 veranschaulicht die Speicherbelegung im CONTROL MODE. Der Befehl besteht hier aus einem Operationscode- und zwei Operandenbytes. Jedem Byte steht ein Steuerwort gegenüber. Bei jedem Lesezugriff gelangt das betreffende Byte des Befehls zur CPU und das betreffende Wort des Steuerspeichers in das Steuerregister. Sind bestimmte Zugriffe im Rahmen eines Befehls keine Lesezugriffe zum Adressenbereich b, so bleibt der Inhalt des Steuerregisters erhalten. Das veranschaulicht Bild 49 am Beispiel eines Schreibbefehls (z. B. LD (HL), A). Der Befehl hat zwei Zyklen, aber nur beim Operationscodelesen gelangt ein Steuerwort ins Steuerregister. Es bleibt beim darauf folgenden Schreibzyklus erhalten und kann dort Sonderwirkungen veranlassen.
Haben Steuerworte eine NO-OP- Belegung, so werden die betreffenden Befehle auch im CONTROL MODE auf herkömmliche Weise (also ohne jegliche Sonderwirkung) ausgeführt. Das gestattet es, recht umfangreiche Routinen im CONTROL MODE abzuarbeiten, die Befehle mit und ohne Sonderwirkung in beliebiger Folge enthalten. Damit entfällt die Notwendigkeit des häufigen Umschaltens (vgl. hingegen Abschnitt 5).

Für die Auslegung von Mikorechneranordnungen nach den beschriebenen Prinzipien haben sich folgende Grundsätze als zweckmäßig erwiesen:

1. Speicher sind reguläre und verhältnismäßig leicht prüfbare Strukturen, die sich mit kostengünstigen LSI- Schaltkreisen verwirklichen lassen.
Für Steuerlogik steht üblicherweise nur ein Sortiment von SSI- und MSI- Bausteinen zur Verfügung; LSI (gate array) wird bei geringen Stückzahlen sehr teuer. Eine umfangreiche Steuerlogik ist wenig übersichtlich und im Rahmen von Selbsttests nur schwer prüfbar. Deshalb:

- ausreichend Steuerspeicher vorsehen (betrifft sowohl die Länge der Steuerworte als auch deren Anzahl)
- Steuerworte so belegen, daß die zusätzliche Steuerlogik minimal wird; Bequemlichkeit beim Programmieren spielt demgegenüber eine untergeordnete Rolle.

2. Der Steuerspeicher sollte so groß sein, wie mit den schnellsten verfügbaren Speicherschaltkreisen noch wirtschaftlich realisierbar. Je größer der Steuerspeicher, desto geringer die Umschaltverluste. Kann hingegen der Steuerspeicher nicht alle Routinen aufnehmen, so muß man entweder umladen (RAM ist notwendig), oder man muß alle Abläufe, die tatsächlich Sonderfunktionen enthalten, in kurzen Unterprogrammen zusammenfassen.
Vergleichsweise langsame Speicherschaltkreise können unter Umständen dazu zwingen, zusätzliche Wartezustände einzufügen.
Faustregel: Der Steuerspeicher sollte wenigstens 1k Steuerworte umfassen. 4k Steuerworte (bei einem Gesamtumfang des Programmspeichers von 16 kBytes) haben sich für einen sehr anspruchsvollen Einsatzfall (vgl. /23/ und /21/, wo das zu steuernde Objekt beschrieben ist) als völlig ausreichend erwiesen. Solche Speicherkapazitäten sind noch mit schnellen statischen RAMs realisierbar (einfache Steuerung, keine Zeitverluste durch Wartezustände).
Ein RAM als Steuerspeicher hat - selbst wenn der eigentliche Befehlsspeicher ein ROM ist - noch weitere Vorteile:

- zu Test- und Erprobungszwecken ist es leicht möglich, im Steuerspeicher zu ändern
- der Steuerspeicher ist auch als normaler Arbeitsspeicher nutzbar (für Variable, für den Stack usw.); ein Bereich von einigen hundert Bytes reicht dafür zumeist aus.

Obwohl die Steuerworte beliebig (nötigenfalls auch extrem) lang sein können, empfiehlt es sich, bei einer solchen kombinierten Nutzung des Steuerspeichers die Länge des Steuerwortes an die Byte- bzw. Wortstruktur der CPU anzupassen.

3. Grundsätzlich wirken die Steuerworte ähnlich wie Mikrobefehle in den bekannten Mikroprogrammsteuerwerken, und es können alle einschlägigen Prinzipien bzw. Erfahrungen zur Gestaltung der Steuerwortformate herangezogen werden (s. etwa /1/, /4/, /8.16/ - /8.19/, /11/).
Ist die Länge des Steuerwortes einmal festgelegt, so kommt es oft nicht darauf an, bei der Codierung der einzelnen Sonderwirkungen Bits einzusparen. Dann erweist sich die einfache 1-aus-n- Codierung (1 bit pro Steuerwirkung) als durchaus zweckmäßig.
Das Steuerwort sollte - wenn immer möglich - so formatiert werden, daß wenigstens 1 Bit bzw. eine Codeposition für künftige Erweiterungen frei bleibt ("Escape"- Vorkehrung).

Im folgenden solllen typische Einzelheiten an einem Beispiel (Basis: Z 8Ø) näher erläutert werden. Dazu zeigt Bild 5Ø eine Mikrorechneranordnung, in der die Möglichkeiten des CONTROL MODE weitgehend genutzt sind. Bild 51 veranschaulicht die Adressenstruktur und Bild 52 die Belegung eines Steuerwortes. Der Mikrorechner hat drei Speicher A, B, C zu je 16 kBytes (8k ROM + 8k RAM). Diese Speicher sind alle in üblicher Weise nutzbar. Ein 16k- Adressenbereich ist für den CONTROL MODE vorgesehen. Dann werden die Speicher A, B, C parallel genutzt,

und zwar A als Befehlsspeicher und B, C zusammen als Steuerspeicher. Das Steuerwort ist also 16 bit lang. Bild 52 zeigt dessen grundsätzliche Aufteilung in ein Funktionscodefeld, ein Steuerfeld und ein Direktwertfeld (EMIT- Feld). Das Funktionscodefeld wird grundsätzlich so interpretiert, wie in Bild 52 dargestellt (8 Belegungen genutzt, 8 als Reserve freigehalten); die Belegung der anderen Felder wird im einzelnen durch den Funktionscode bestimmt.
Aus den Bildern 53, 54 sind wesentliche Einzelheiten der Speicheranordnung ersichtlich (die ROM/RAM- Kombination jedes Speichers ist jeweils zusammengefaßt dargestellt). Schreibimpulse werden nur bei RAM- Zugriffen mit Adressen außerhalb des CONTROL MODE- Bereichs abgegeben. Hingegen werden die "chip enable"- Eingänge der Speicherkomplexe sowohl bei üblichen Zugriffen aktiviert als auch bei Lesezugriffen im Adressenbereich des CONTROL MODE (Schreibzugriffe zu diesem Bereich sind wirkungslos). Bei Lesezugriffen außerhalb des CONTROL MODE-Bereichs wird der jeweils adressierte Speicher auf den Datenbus der CPU geschaltet, bei Lesezugriffen im CONTROL MODE-Bereich stets der Speicher A, damit die CPU die Befehle lesen kann. Bild 55 veranschaulicht anhand einer Ausgestaltung der CONTROL MODE- Steuerung die wesentlichen Einzelheiten:

1. CONTROL MODE muß beim Operationscodelesen (M1- Zugriffe zum entsprechenden Adressenbereich) einschalten und so lange aktiv bleiben (im Beispiel über Rückführung), bis ein Operationscode außerhalb des CONTROL MODE- Adressenbereichs gelesen wird.

2. Kritisch sind Zugriffe, die bei aktivem CONTROL MODE von der CPU aus anderen Ursachen (z. B. Interrupt, NMI) eingefügt werden.
Ist das System so ausgelegt, daß solche Bedingungen nie eintreten, kann man auf Sondermaßnahmen verzichten. Ansonsten ist dafür zu sorgen, daß zu Beginn des Behandlungsablaufs CONTROL MODE ausgeschaltet wird. Das ist im Beispiel wie folgt gelöst: CONTROL MODE wird bei E-A- Zugriffen (IORQ) gelöscht (E-A-Zugriffe sind grundsätzlich frei von Sonderwirkungen!).
Da der Interrupt mit M1 und IORQ quittiert wird, ist das gewünschte Verhalten der Schaltung gewährleistet.
NMI wird der CPU über ein Zwischenflipflop zugeführt, das die Anforderung mit dem Beginn des jeweils nächsten Zyklus synchronisiert. Ist das Zwischenflipflop einmal gesetzt, so wird NMI garantiert am Ende des aktuellen Befehle behandelt; der nächste Zyklus mit M1 und MREQ ist also der NMI- Quittungszyklus - und da schaltet CONTROL MODE aus, weil das Zwischenflipflop die Rückführung auftrennt.
Hinweise:

- Die NMI- Anforderung muß gehalten werden bis zur Annahme durch die CPU.
- Befehle mit zwei M1- Zyklen dürfen nie Sonderwirkungen ausüben, da die Schaltung nach Bild 55 bei aktivem Zwischenflipflop den 2. M1- Zyklus als NMI- Quittungszyklus interpretiert. Sollte die Programmiereinschränkung nicht tragbar sein, so muß die Schaltung entsprechend erweitert werden, z. B. gemäß Bild 56.

Bild 57 zeigt das Steuerregister zusammen mit der Decodierung des Funktionscodefeldes. Das Register wird nur bei Lesezugriffen mit einer Adresse im CONTROL MODE- Bereich geladen. Es ist bei inaktivem CONTROL MODE stets zurückgesetzt (NO-OP- Belegung).
CONTROL MODE schaltet stets zu Beginn von MREQ ein, gibt das Register also rechtzeitig frei. Der weitere Zeitablauf im Zyklus muß folgenden Bedingungen gerecht werden:

- das Steuerregister kann erst geladen werden, wenn die Daten an den Speicherausgängen bereitstehen
- wirkt die Befehlserweiterung auf die CPU unmittelbar (im selben Zyklus) zurück, so darf der Zyklus erst dann beendet werden, wenn die Rückwirkung eingetreten ist (betrifft z. B. Aufschalten von Information auf den Datenbus, Stellen von Unterbrechungsanforderungen usw.).

Mit schnellen Bauelementen für Steuerspeicher und Zusatzlogik (z. B. Steuerregister mit 74S175) ist es kaum erforderlich, zusätzliche Wartezustände einzufügen. Oft reicht es aus, gegen Ende des Zyklus zwei zeitversetzte Impulse zu erzeugen:

- LOAD CTL REG zum Laden des Steuerregisters
- LOAD PULSE als Taktimpuls für die Wirkungen, die vom Steuerregister ausgehen (der Abstand gegenüber LOAD CTL REG ist gegeben durch Verzögerungszeit des Steuerregisters + Decodierzeit + Vorhaltezeit der anzusteuernden Flipflops bzw. Register).

Derartige Impulse sind z. B. mit einer Laufzeitkette oder mit einem Schieberegister leicht zu bilden (Bild 58).
Erweist es sich als erforderlich, Wartezustände einzufügen, so ist zu beachten, daß viele Mikroprozessoren eine solche Entscheidung relativ früh im Zyklus verlangen (Z 8Ø: fallende Flanke des 2. CPU- Taktes; vgl. auch Abschnitt 11).
Man kann den Wartezustand deshalb kaum vom Steuerregister aus beeinflussen (z. B. durch besondere Steuerbits); es verbleibt oft nur, zu Beginn des Zyklus den Wartezustand gewissermaßen prophylaktisch einzuschalten und bei eingetretener Steuerwirkung zurückzunehmen. Die Bilder 59, 6Ø veranschaulichen eine solche Schaltungslösung, weitere Einzelheiten sind z. B. in /8.12/, /8.13/, /17/ und /26/ zu finden.

Nachfolgend werden die Sonderwirkungen im einzelnen beschrieben.

6.2. Ausgabe von Direktwerten

Teile des aktuellen Steuerwortes können ohne besondere Schwierigkeiten direkt auf Interfaceleitungen geschaltet werden. Da die Aufrufbreite an sich beliebig sein kann, lassen sich auch extreme Anforderungen erfüllen, so z. B. das gleichzeitige Erregen von über 4Ø Steuerleitungen in einem einzigen Befehl (z. B. ein NOP, mit dem zusammen ein entsprechend breites Steuerwort gelesen wird; das kostet beim Z 8Ø nur 4 Maschinen-

takte).
Von besonderem Interesse ist es, einen Steuerspeicher mit üblicher Breite (8 oder 16 bit) nutzen zu können, um möglichst viele Interfaceleitungen anzusteuern. Bild 61 zeigt dazu eine Struktur im Rahmen des Beispiels von Bild 5Ø. Zur Programmierung sind Direktwert- Ladebefehle (z. B. LD HL, nm) vorgesehen, denen Steuerworte so zugeordnet sind, wie dies in Bild 62 gezeigt ist. Zum Interface hin werden die Direktwertbytes m, n sowie die EMIT- Bytes der drei Steuerworte ausgegeben (4Ø bit; deren Übertragung erfordert nur 11 Maschinentakte). Diese Belegung steht am Ausgang eines Interfaceregisters parallel zur Verfügung. Zum Puffern der Bytes, die im ersten und zweiten Zyklus des Befehls geliefert werden, sind Zwischenregister vorgesehen. Das Interfaceregister wird im dritten Zyklus geladen, wobei der Ladeimpuls die Gültigkeit der neuen Belegung anzeigt.
Das Laden der Register wird durch Steuerwortbelegungen im 1-aus-n- Code gesteuert. Dadurch wird die Zusatzlogik sehr einfach.
Von Nachteil ist, daß ein CPU- Register verfügbar sein muß, um die vom Befehl gelieferten Datenbytes m,n aufzunehmen.

Ein erster Ausweg besteht darin, auf die Nutzung der Datenbytes zu verzichten und nur das EMIT- Feld des Steuerwortes auszugeben. Dann wäre die Steuerwirkung des Beispiels mit 5 aufeinanderfolgenden NOP- Befehlen (2Ø Maschinentakte) zu erzielen. Es sind dazu vier Zwischenregister anzuordnen. Die Information zum selektiven Laden dieser Register muß im Steuerwort codiert werden (im beispielgemäßen Format ist dann der 1-aus-n- Code nicht mehr anwendbar).

In manchen Anwendungsfällen kann es erforderlich sein, Interrupts für die CPU zu sperren, wenn, wie eben beschrieben, die Interface- Ansteuerung auf mehrere Befehle aufgeteilt ist. Dann sind die entsprechenden CPU- Eingänge (INT, NMI) mit Sperrgattern zu beschalten, die wiederum an das Steuerregister angeschlossen sind (direkt an eine besondere Bitposition bzw. an Verknüpfungen mehrerer Bits, wenn das Verhindern von Unterbrechungen bei verschiedenen Funktionen erforderlich ist).

Ein weiteres Prinzip der Funktionssteuerung, das für mehrere Sonderwirkungen von grundsätzlicher Bedeutung ist, zeigt Bild 63.
Über eine Codeposition des Steuerwortes wird veranlaßt, daß der CPU eine Datenbusbelegung zugeführt wird, die einem NOP-Befehl entspricht (ØØH beim Z 8Ø). Das Steuerwort kann weiterhin beliebige Wirkungen ausüben, auch kann die ursprünglich aus dem Befehlsspeicher gelesene Datenbelegung auf beliebige Weise genutzt werden (so lassen sich gemäß Bild 63 mit 3 Bytes im Befehlsstrom, die an der CPU durch NOPs ersetzt werden, in 12 Maschinentakten 48 Interfaceleitungen erregen). Das Prinzip der NOP- Aufschaltung erfordert folgende Wirkungen von der zusätzlichen Hardware:

1. Nutzung der gelesenen Datenbusbelegung (z. B. Übernahme in Register), wenn erforderlich

2. Freigeben des Datenbus
3. Aufschalten eines Festwertes, der dem NOP- Operationscode entspricht
4. Beenden des Speicherzugriffs.

Erforderlichenfalls muß die CPU so lange im Wartezustand gehalten werden, bis die NOP- Belegung aufgeschaltet ist. Zum Freigeben des Datenbus gibt es zwei Schaltungslösungen:

1. Die Speicher selbst oder deren leseseitigen Buskoppelstufen werden deaktiviert. Bild 64 zeigt dazu die Modifikation des Decoders gemäß Bild 54.
Nachteil: sequentielle Arbeitsweise notwendig (erst Belegung übernehmen, dann freischalten).

2. Entkopplung des Speicherdatenpfades von der CPU durch gesonderte Buskoppelstufe (in größeren Mikrorechner- Konfigurationen ohnehin notwendig); Aufschalten des NOP an der CPU-Seite (Bild 65).
Vorteil: Informationsübernahme und Aufschalten können parallelisiert werden.

6.3. Ausgabe von Registerbelegungen

Um Registerinhalte der CPU auf Interfaceleitungen auszugeben, wird die Adresse benutzt, die von entsprechenden Befehlen (z. B. LD (HL), A) beim Datentransport geliefert wird. Das Datenbyte selbst kann zusätzlich mit ausgegeben werden (vgl. Abschnitt 5, /23/ und /25/).
Die grundsätzliche Schaltungsanordnung zeigt Bild 66; zum Ablauf s. Bild 49 und auch Bild 41.
Im Steuerwort sind viele Bitpositionen frei, so daß weitere Steuerwirkungen angebbar sind, so z. B. die Auswahl des Ziels der Information im EMIT- Feld. Bild 66 veranschaulicht, daß mehrere Registerinhalte aufeinanderfolgend ausgegben werden können, die dann gemeinsam über ein Interfaceregister wirksam werden. Mit zwei Befehlen (z. B. LD (HL), B; LD (DE), A) kann man bis zu 48 bit in 14 Maschinentakten ausgeben.

Entscheidend ist, daß vom Steuerwort aus alle Speicherzugriffe im Schreibzyklus verhindert werden. Dazu zeigt Bild 67 die Modifikation des Schreibimpulsdecoders von Bild 53. In manchen Mikrorechneranordnungen kann es notwendig sein, alle Speicheradressendecoder zu sperren, so daß auch kein "chip enable"- Eingang der Speicher aktiviert wird.

6.4. Eingaben in CPU- Register

Dazu werden Befehle ausgenutzt, die Direktwerte mit einer entsprechenden Anzahl von Datenbits in CPU- Register transportieren (vgl. Abschnitt 5, /23/ und /25/). Gleichzeitig mit den Speicherzugriffen zu diesen Direktwerten werden Steuerworte gelesen, die veranlassen, daß anstelle der Speicherinhalte die Interfacebelegungen abschnittsweise auf den CPU- Datenbus

geschaltet werden (Bild 68). Die Speicher selbst sind am Datenbus zu deaktivieren, beispielsweise über eine Koppelstufe zwischen Speicher und CPU- Datenweg (s. Bild 65; zur Freigabe muß dann REGISTER INPUT sinngemäß wirksam werden).
Bild 68 veranschaulicht, daß es auf einfache Weise möglich ist, zwischen mehreren Interfaceregistern auszuwählen. Die Zuordnung von Befehlsbytes und Steuerworten zeigt Bild 69.

Hinweise:

- In manchen Anwendungsfällen sind anstelle der Interfaceregister einfache Koppelstufen ausreichend. Um eine von mehreren Interfacebelegungen auszuwählen, können dann auch Multiplexerschaltkreise mit "tri state"- Ausgängen eingesetzt werden.
- Sind Interfaceregister notwendig, so ist zu beachten, daß in bestimmten Konfigurationen der Übernahmetakt zu Störungen führen kann (z. B. bei Einsatz von Registern 8282/8283, die ausgangsseitig direkt an allgemein benutzte Datenleitungen angeschlossen sind). Auswege:
  a) Informationsübernahme nur außerhalb von CPU- Zyklen (MREQ bzw. IORQ), so daß Störimpulse an den Datenausgängen nicht schaden
  b) weitere Koppelstufe zwischen Interfaceregistern und Datenbus, die nur bei REGISTER INPUT aktiviert wird.

6.5. Transporte von Speicherinhalten

In manchen Anwendungsfällen geht es darum, Datenblöcke zwischen einem Interface und einem Speicher auszutauschen, der weder Befehls- noch Steuerspeicher ist (er kann praktisch als Teil der vom Mikrorechner zu steuernden Einrichtung angesehen werden). Es ist dann möglich, Teile der für Befehlszugriffe von der CPU gelieferten Adresse zur Datenadressierung zu nutzen und mit den Steuerworten einen DMA- Datenaustausch zwischen Interface und Datenspeicher zu steuern (vgl. Abschnitt 2.2. und /18/).
Bild 70 zeigt das Prinzip, Bild 71 veranschaulicht die Zuordnung der Speicherbereiche. Ein üblicher Einsatzfall ist z. B. der Datenaustausch zwischen einem Interface und einem fest reservierten Pufferbereich im Datenspeicher. Zu Befehlsspeicher, Steuerspeicher und Datenspeicher wird dann gemeinsam mit dem entsprechenden niederen Teil der CPU- Adresse zugegriffen. Sind die zu übertragenden Datenblöcke z. B. 1 kBytes lang, so sind alle Bereiche an eine 1k- Grenze zu legen, und die 10 niedrigstwertigen Adressenbits werden von der CPU an alle Speicher geliefert. Die verbleibende Adresse für den Datenspeicher kann auf an sich beliebige Weise bereitgestellt werden; im Beispiel ist gezeigt, wie das EMIT- Feld des Steuerwortes dazu ausgenutzt werden kann. Die Einzelheiten der Speicheradressierung sind aus Bild 72 ersichtlich, Bild 73 zeigt die Steuerschaltung für die Datenübertragung.
Für die Übertragung von 1024 Bytes sind im Befehlsspeicher 1024 NOPs vorzusehen. Pro Byte werden dann 4 Maschinentakte benötigt (gegenüber 11 bei DMA- Steuerung mit Adressenerwei-

terung nach Abschnitt 2.2.).
Der Platz im Befehlsspeicher kann weitgehend für andere Zwecke zur Verfügung gestellt werden, wenn das bereits beschriebene Prinzip der NOP- Aufschaltung angewendet wird (in Bild 69 bereits vorgesehen, vgl. Bild 63 - 65). Da der CPU so lange NOPs angeboten werden, wie dies in den Steuerworten angegeben ist, kann der Befehlsspeicher einen an sich beliebigen Inhalt haben, z. B. mit Programmen belegt sein, die nicht im Adressenbereich des CONTROL MODE abgearbeitet werden.

Hinweise:

- Der Datenspeicher kann grundsätzlich eine beliebige Aufrufbreite haben.
- Die Richtung der Übertragung wird vom Steuerwort bestimmt (Bit DIR).
- In den Bildern 72, 73 ist weiterhin angedeutet, daß zum Datenspeicher auch übliche CPU- seitige Zugriffe vorgesehen sind (CPU ACCESS). Dabei wird vorausgesetzt, daß die CPU eine hinreichend lange Adresse liefert, etwa über Erweiterungsmaßnahmen, wie sie in Abschnitt 2 beschrieben sind.
- Der Datenaustausch zwischen Interface und Datenspeicher kann erst dann vollzogen werden, wenn das Steuerregister geladen wurde. Der niedere Teil der CPU- Adresse muß bis zum Ende des Datenspeicherzugriffs gehalten werden. Bei extremen Anforderungen ist zu empfehlen, für den niederen Teil der Datenspeicheradresse ein Halteregister vorzusehen; dann läßt sich der CPU- Zyklus bereits nach dem Steuerwortlesen beenden, und der interfaceseitige Datentransport findet im Zeitabschnitt zwischen zwei CPU- Zugriffen statt (bei dynamischen RAMs können zusätzliche Maßnahmen für das Auffrischen notwendig werden).
- Die Transportabläufe sind an sich unterbrechbar; es ist deshalb nicht notwendig, daß INSERT NOP unmittelbar die Unterbrechungsanforderungen sperrt (wie etwa in Bild 65). Vielmehr ist es zweckmäßig, für das Sperren eine besondere Bitposition im Steuerregister zu reservieren (wie in Bild 66).

In der Ausgestaltung nach Bild 74 ist das Prinzip auch direkt in der Konfiguration von Bild 50 nutzbar. Da das EMIT- Feld im Steuerwort an sich nicht benötigt wird, lassen sich die Speicher A, B als Datenspeicher verwenden. So bietet es sich an, in deren RAM- Teilen feste Pufferbereiche für den schnellen Datenaustausch (bis zu zwei Bytes parallel) vorzusehen. Der Speicher C liefert dann die Steuerworte, die bewirken, daß der CPU NOPs zugeführt werden und die den Datentransport zwischen den Speichern A, B und dem Interface steuern. Die Decoder in Bild 53 sind dazu mit disjunktiven Verknüpfungen so zu beschalten, daß bei Lesezugriffen zum Interface (DIR = 0) Schreibimpulse zu den Speichern A, B abgegeben werden. Bei Schreibzugriffen zum Interface (DIR = 1) ist das Interfaceregister mit den gelesenen Belegungen der Speicher A, B zu laden; das erfordert keine besonderen Vorkehrungen, da bei CONTROL MODE- Zugriffen alle drei Speicher aktiviert sind.

6.6. Programmverzweigungen

Um auf Bedingungen zu reagieren, die von externen Interfaces herrühren, sind vergleichsweise zeitaufwendige Programmabläufe notwendig: Holen der Signalbelegung (Eingabe, erforderlichenfalls mit vorangehender Ausgabe zwecks Bedingungsauswahl), Abfragen (z. B. Bittest), Verzweigen.
Diese Aktivitäten können weitgehend in zusätzliche Hardware verlagert werden, wie in den Bildern 75 und 76 veranschaulicht. Es wird die bereits beschriebene NOP- Aufschaltung benutzt, die hier in Abhängigkeit von einer auswählbaren externen Bedingung wirksam wird. Zur Auswahl dient im Beispiel das EMIT- Feld, wodurch bis zu 256 externe Signale der Auswahl zugänglich werden. Eine weitere Bitposition des Steuerregisters (CPL) entscheidet darüber, ob der Signalwert 1 oder der Signalwert Ø die Bedingung aktivieren soll. Bei gesetzter Bedingung wird NOP aufgeschaltet; ansonsten wird das gelesene Befehlsbyte der CPU zugeführt, und es gibt keine Sonderwirkungen. Für diese Abläufe sind folgende Anforderungen von der zusätzlichen Hardware zu erfüllen:

1. Die Bedingung darf nur gestellt werden, wenn dies im Steuerwort ausdrücklich angegeben ist.
2. Während des Ablaufs darf die einmal gestellte Bedingung nicht umschalten.
3. Während des Ablaufs sind die Unterbrechungsanforderungen zu sperren.

Das Prinzip besteht im Ersatz von an sich beliebigen Befehlen durch NOPs. Das erlaubt ein bedingtes Überspringen (SKIP) von an sich beliebigen Programmabschnitten bzw. einzelnen Befehlen. Im besonderen kann es sich dabei um Verzweigungsbefehle handeln (JMP, CALL). Diese stehen als unbedingte Verzweigungen im Programm. Sie werden durch die zugehörigen Steuerworte (s. Bild 76) zu bedingten Verzweigungen. Ist das Bedingungsflipflop gesetzt, so wird der betreffende Befehl von der CPU gelesen und ausgeführt. Ist das Bedingungsflipflop nicht gesetzt, werden NOPs aufgeschaltet, es findet somit keine Verzweigung statt. Beim Lesen des ersten Operationscodebytes muß über die Bedingung entschieden werden. Die einmal getroffene Entscheidung muß so lange gültig sein, bis alle Bytes des Befehls bzw. Programmabschnittes von der CPU adressiert wurden. Deshalb wird das Bedingungsflipflop nur gestellt, wenn im Steuerwort die Erlaubnis dazu angegeben ist (COND SET), und das ist nur im ersten Steuerwort des Ablaufs der Fall. Solange weitere Steuerworte mit dem Funktionscode JUMP/SKIP gelesen werden, entscheidet die Stellung des Bedingungsflipflops darüber, ob der NOP aufgeschaltet wird oder nicht.

6.7. Weitere Sonderwirkungen

Es ist wenig sinnvoll, mit den Prinzipien, die bisher beschrieben wurden, eine weitgehend universelle Schaltungsanordnung zu verwirklichen; stattdessen sollten für den konkreten Anwendungsfall zweckmäßige Lösungen ausgewählt und mit jeweils

angemessenen Schaltmitteln realisiert werden. Man erhält so mit recht geringen Zusatzaufwendungen eine sehr flexible Hardware, die weniger aufwendig ist als eine reine Sonderschaltung, aber deutlich leistungsfähiger als ein herkömmlicher Mikrorechner. In /23/ ist dargestellt worden, daß typische Steuerungsabläufe gegenüber der konventionellen Programmierung eines nach CPU- Ausstattung und Taktfrequenz vergleichbaren Mikrorechners etwa um das 3 - 10fache beschleunigt werden können, wobei ein 8-bit- Mikrorechner einem 16-bit- Mikrorechner (dessen Grundaufwendungen höher sind) noch deutlich überlegen (d. h.: 2 - 3 mal schneller) ist.
In diesem Sinne läßt die Erweiterung des Operationscodes noch vielfältige Abwandlungen zu. So können z. B. andere Schaltungsvorschläge mit ähnlicher Zielstellung, die auf Erweiterungen der Adresse, Umschaltbefehlen usw. beruhen, ohne besondere Schwierigkeiten einbezogen werden (vgl. die Abschnitte 2 - 5; die in Abschnitt 5 bzw. in /25/ erläuterten Prinzipien werden erst in der hier angegebenen Form wirklich praktikabel).
Von besonderem Vorteil ist, daß die Zusatzaufwendungen nur gering sind und daß alle Sonderwirkungen durch gespeicherte Angaben bestimmt werden. Damit sind Fragen der Prüfung, Diagnose und Änderung mit programmtechnischen Mitteln vergleichsweise (gegenüber einer reinen Spezialhardware) einfach lösbar.
Im Rahmen dieses Prinzips wurden praktisch alle bisher beschriebenen Konzepte zur beschleunigten Ein- und Ausgabe eingesetzt, um mit einem 8-bit- Mikrorechner eine Videosteuerschaltung nach /21/ zu betreiben (s. weiterhin /23/). Dazu hat sich ein Steuerwort von nur einem Byte als ausreichend erwiesen (s. Bild 77).

7. Vorkehrungen für den Adressen- Vergleichsstop

Maßnahmen zur Beobachtung von Programmabläufen sind notwendig, wenn es darum geht, neu entwickelte Mikrorechneranordnungen in Betrieb zu nehmen bzw. umfangreiche Programmkomplexe zu erproben. Besonders bei Systemen, die unter Echtzeitbedingungen arbeiten, treten gelegentlich noch nach längerer Praxisbewährung Fehler auf. Programmtechnische Fehlersuchhilfen allein sind nicht in allen Fällen ausreichend; sie sind manchmal gar nicht nutzbar oder verändern das Echtzeitverhalten so, daß das System nicht unter Einsatzbedingungen erprobt werden kann. Übliche Hardware- Fehlersuchhilfen reichen von einfachen Adressenvergleichseinrichtungen bis zu industriell gefertigten Logikanalyzern. Solche Mittel sind oft vollkommen ausreichend; manchmal zeigen sich aber die Nachteile:

- die Anordnungen müssen extern angeschlossen werden (das betrifft eine Vielzahl von Adressen- und Datenleitungen)
- sie sind nicht immer an Ort und Stelle verfügbar (namentlich zur Fehlersuche unter Einsatzbedingungen)
- Logikanalyzer sind teuer; ihre funktionellen Möglichkeiten werden beim Suchen von Softwarefehlern nur zu einem geringen Teil genutzt
- zur Suche von Fehlern beim Zusammenwirken mehrerer Funktionsmoduln sind die Aufwendungen kaum tragbar (z. B. in Multimikrorechnersystemen ein Logikanalyzer an jedem verdächtigen Mikrorechner; vgl. /8.1/ und /8.11/).

Eine einfache Schaltungsanordnung, die in jedem Mikrorechner fest vorgesehen werden kann, ist in /13/ dargestellt (s. auch /8.11/ und /8.14/). Die Anordnung ist sinngemäß an Speichern mikroprogrammierter Steuerwerke einsetzbar (/14/; /8.19/, /8.20/).
Damit ist es möglich, Aussagen darüber zu erhalten, ob Zugriffe auf bestimmte Speicherplätze bzw. -bereiche ausgeführt wurden oder nicht. Gemäß Bild 78 wird die betreffende Speicheranordnung mit zusätzlichen Speicherschaltkreisen erweitert (Einzelheiten sind in den Bildern 79 - 81 dargestellt).
Es können sowohl ROM- als auch RAM- Anordnungen erweitert werden; die zusätzlichen Speicher sind aber immer mit RAM-Schaltkreisen aufgebaut. Sie sind alternativ für zwei Aufgaben vorgesehen:

1. Während des normalen Betriebes können sie zur Fehlerkontrolle der zugeordneten Informationsspeicher genutzt werden. Das kann eine Paritätsprüfung sein; Fehlerkorrekturmaßnahmen (ECC) bereiten keinerlei Schwierigkeiten (sie sind beispielsweise im Mikrorechner nach /26/ verwirklicht, s. auch Abschnitt 9).

2. Zwecks Fehlersuche (Ablaufverfolgung) werden die Zusatzspeicher umgeladen und als Adressenvergleicher genutzt. In diesem Modus werden die Zusatzspeicher nur gelesen (auch bei Schreibzugriffen). Werden dabei bestimmte Belegungen vorgefunden, so wird eine Vergleichsbedingung signalisiert. Dazu wird üblicherweise ein NMI oder ein ähnlicher hoch priorisierter

Interrupt ausgelöst.

Wesentlich ist das Umstellen der Speicherinhalte beim Wechsel zwischen den Nutzungsweisen. Diese Aufgabe wird programmtechnisch gelöst. Dazu müssen in der Hardware 6 Betriebszustände einstellbar sein (üblicherweise mittels eines ladbaren Steuerregisters wie in Bild 78 und 79):

Betriebsart 1 - Speicherinhaltsüberwachung (Normalbetrieb)

Der Inhalt des Informationsspeichers wird durch zusätzliche Kontrollbits (Parität oder ECC) überwacht. Dabei werden alle Lese- und Schreibzugriffe gleichzeitig zum Informations- und Zusatzspeicher geführt.
Belegung des Steuerregisters (Bild 79): Es sind ENABLE PARITY CHECK und ENABLE ERROR SIGNALIZATION aktiv, d. h. die Kontrollinformation wird sowohl eingeschrieben als auch bei Lesen ausgewertet.

Betriebsart 2 - Speicherzugriffsüberwachung (Adressenvergleichsstop)

Die Zusatzspeicher werden nur gelesen (auch bei Schreibzugriffen). Die gelesenen Bits werden in der Hardware verglichen. Die Stopbedingungen können auf vielfältige Weise festgelegt werden; die Bilder 798 und 8Ø zeigen lediglich ein einfaches Beispiel. Im besonderen bietet sich bei ECC- Speichern die Möglichkeit, in den einzelnen Zusatzbits selektive Stopbedingungen zu codieren (z. B. Stop bei Lesen, Stop bei Schreiben, Stop bei Befehlszugriffen, Stop bei Datenzugriffen usw.). Im Beispiel ist das Steuerregister so belegt: ENABLE COMPARE STOP ist ständig aktiv; je nach Bedarf werden STOP ON READ sowie STOP ON WRITE angeschaltet (ist weder STOP ON READ noch STOP ON WRITE gesetzt, so findet auch bei eingestellten Stopadressen nie ein Vergleichsstop statt) .

Betriebsart 3 - Umsteuerung in den Vergleichsstopmodus

Die Zusatzspeicher werden völlig gelöscht. Das heißt:
- im Steuerregister wird nur ENABLE INJECT gesetzt
- der gesamte Speicherinhalt wird gelesen und wieder zurückgeschrieben (z. B. mit einem Blocktransportbefehl).

Betriebsart 4 - Einstellen von Stopadressen

Im Steuerregister wird nur ENABLE INJECT gesetzt. INJECT HI wird gesetzt, wenn die Stopadresse(n) aktiviert werden sollen. Dann werden Zugriffe der Form "Lesen - Zurückschreiben" mit allen Adressen ausgeführt, wo eine Stopbedingung aktiviert bzw. gelöscht werden soll. (INJECT HI repräsentiert den Wert, der in das Zusatzbit eingeschrieben wird).

Betriebsart 5 - Übergang zum Normalbetrieb

Die Speicherinhaltskontrolle (Parity bzw. ECC) wird eingeschaltet, aber die Fehlersignalisierung verhindert (ENABLE

PARITY ein, ENABLE ERROR SIGNALIZATION aus). Dann wird der gesamte Speicherinhalt gelesen und wieder zurückgeschrieben. So wird die Fehlerkontrollinformation in den Zusatzspeicher eingetragen. Der gelesene Inhalt des Zusatzspeichers bleibt aber wirkungslos.

Betriebsart 6 - keine Wirkung des Zusatzspeichers

Bei völlig gelöschtem Steuerregister hat der Zusatzspeicher keinerlei Wirkung. Diese Betriebsart wird von der Testsoftware bei der Hardware- Diagnose benutzt.

Hinweise:

1. Mit den beschriebenen Vorkehrungen sind sowohl Stopbedingungen für einzelne Adressen als auch für ganze Adressenbereiche einstellbar (betrifft Befehls- und Datenzugriffe). So kann ein befehlsweiser Betrieb eingestellt werden, indem der betreffende Programmabschnitt vollständig mit Stopbits im Zusatzspeicher belegt wird.

2. Ist der zu überwachende Speicher ein ROM, so wirken Schreibzugriffe beim Umstellen der Betriebsart nur für den Zusatzspeicher. Es muß dann schaltungstechnisch dafür gesorgt werden, daß der ROM nicht den Datenbus belegt.

3. Das Umsteuern der Betriebsarten und das Auswerten von Stopbedingungen kann oft der betreffende Mikrorechner selbst übernehmen. Mit modernen Bedien- und Anzeigemitteln (Bildschirm, Tastatur) läßt sich ein hoher Bedienkomfort gewährleisten. Es muß aber dafür gesorgt werden, daß Stackbereiche und Arbeitsbereiche der betreffenden Software grundsätzlich nicht selbst mit Stopbedingungen belegt werden (Prüfung und nötigenfalls Rückweisung der gewünschten Stopadressen).

8. Rücklesbare Ausgaben

Gelegentlich sind in Mikrorechneranordnungen Hardwareregister vorzusehen, die programmseitig zugänglich sein müssen und unmittelbar Steuerwirkungen veranlassen. Beispiele sind Register zur Adressenverlängerung (wie das Busadressenregister in /17/ bzw. /8.11/ - /8.15/ oder das Steuerregister von Bild 79). An sich kann man solche Register mit E-A- Zugriffen laden, und es sind passende E-A- Schaltkreise für solche Zwecke nutzbar (z. B. Z 8Ø PIO).
Im Unterschied zu "normalen" Ausgaben, wo Datenströme abgeschickt werden, die dann gleichsam "vergessen" werden können (etwa bei der zeichenweisen Ausgabe zu einem Drucker), üben solche Register beständig Steuerwirkungen aus, und es ist notwendig, daß der Registerinhalt der Software immer zugänglich ist. Von besonderem Vorteil wäre es, ließe sich der Registerinhalt wie eine beliebige Speicherzelle behandeln, also testen, bitweise setzen, erhöhen, vermindern usw. Übliche Auswege dafür sind:

- softwareseitiges Verwalten einer gespeicherten Kopie; das heißt aber: Zugriff nur über entsprechende Unterprogramme, also Zeitverlust
- Adressierung des Registers über den Speicheradressenraum und Nachschalten eines Bustreibers, der bei Lesezugriffen den Registerinhalt auf den Datenbus zurückführt; das ist aber aufwendig, besonders dann, wenn mehrere solche Register vorgesehen sind.

Eine weiter Möglichkeit besteht darin, die genannte RAM- Kopie automatisch zu führen (/17/, /8.11/).
Die betreffenden Register sind über den Speicheradressenraum zu adressieren ("memory mapped I/O"), und es ist dafür zu sorgen, daß bei allen Zugriffen zu den Registern parallel ein RAM- Bereich adressiert wird. Bei Schreibzugriffen wird gleichzeitig in RAM und Register geschrieben, bei Lesezugriffen wird der RAM gelesen. Damit werden alle inhaltsmodifizierenden sowie Testbefehle direkt auf die Registerinhalte anwendbar (Erhöhen, Vermindern, Vergleichen, Bits setzen/löschen/testen usw.).
Bild 82 zeigt das Prinzip. Es ist an sich lediglich eine zweckgerechte Auslegung des Adressendecoders notwendig (Registeradressen sind besondere RAM- Adressen, die zusätzlich decodiert werden, um bei Schreibzugriffen Ladeimpulse zum jeweiligen Register zu liefern).

Hinweise:

1. Bei einem Steuerregister gemäß Bild 79 ist dafür zu sorgen, daß bei den Registerzugriffen die Vergleichsstopbedingung nie wirksam werden kann (konjunktive Verknüpfung mit /REGISTER ACCESS).

2. Für den Speichertest ist es vorteilhaft, den parallelen Registerzugriff abschalten zu können, um Nebenwirkungen beim

Schreiben von Testbelegungen zu verhindern. Lösungen:

- programmseitig stellbare Erlaubnisleitung
- besonderer Adressenraum für die Registerzugriffe (wie in /17/ bzw. /8.11/), so daß unabhängige RAM- Zugriffe ausführbar sind, die die Register nicht beeinflussen (geringerer Aufwand; es geht aber ein beträchtlicher Teil des Adressenraums - z. B. 16 k - der anderweitigen Verwendung verloren).

9. Speicheranordnungen mit Fehlerkorrektur

Die Nutzung von Fehlerkorrekturcodes (ECC) ist allgemein bekannt (s. beispielsweise /5/), und seit einiger Zeit sind LSI-Schaltkreise verfügbar, so daß man solche Speicheranordnungen recht einfach aufbauen kann. Die Mehraufwendungen an Speicherschaltkreisen sind aber prinzipbedingt. Sie fallen um so stärker ins Gewicht, je geringer die Aufrufbreite des Speichers ist:

Aufrufbreite	Zusatzbits	Mehraufwand in %
8 bit	5	62,5
16 bit	6	37,5
32 bit	7	21,9
64 bit	8	12,5

(Die Zusatzbit- Angaben gelten für die Korrektur von 1-bit-Fehlern und die Erkennung von 2-bit- Fehlern. Bei Verzicht auf letzteres ist jeweils 1 bit weniger erforderlich.)

Gerade bei Speichern für kleinere Mikrorechner wird der Zusatzaufwand extrem hoch. Andererseits ist es unbedingt wünschenswert, auch in solchen Systemen eine vergleichsweise großzügige Speicherausstattung vorzusehen (Leistungsgewinn durch unmittelbare Verfügbarkeit aller Programme und Daten, beträchtliche Vereinfachung der Programmierarbeit). Die Größenordnung dessen, was als "großzügige" RAM- Ausstattung anzusehen ist, betrifft etwa den Bereich von 32 kBytes bis 2 MBytes (im einzelnen stark vom Einsatzgebiet abhängig). Aus praktischer Sicht geht es also um Speicher, die einerseits zu groß sind, um sie noch mit statischen RAMs kostengünstig verwirklichen zu können, aber andererseits so klein, daß Mehraufwendungen für die Fehlerkorrektur nur in besonderen Fällen akzeptiert werden können. Die üblichen Alternativen werden nachfolgend kurz erläutert:

1. Verzicht auf Kontrollmaßnahmen. Der wichtigste Grund für Fehlerkorrektur- Hardware besteht darin, daß in dynamischen RAMs mit sporadischen Bitverlusten ("soft errors") gerechnet werden muß. Die Fehleranfälligkeit wächst mit zunehmender Speicherkapazität. Es wird aber von den Herstellern viel getan, um die Fehlerrate zu senken (Gehäusematerial, Gestaltung des Layout usw.). Hat man die Möglichkeit, Speicherschaltkreise mit hinreichender Spezifikation stabil zu beziehen, kann auf Kontrollmaßnahmen verzichtet werden. Man sollte aber Vorkehrungen treffen, um den Speicher testen zu können (ROM-residente Prüfroutinen) und sichere Betriebsbedingungen gewährleisten (Zeitverhältnisse der Speicher- Ansteuerung, des Auffrischens usw. so gestalten, daß gegenüber den Grenzwert-Angaben des Datenblattes deutliche Reserven bleiben).

2. Paritätskontrolle. Damit kann der Speicherinhalt lediglich auf 1-bit- Fehler überwacht werden. Die Mehraufwendungen sind

vergleichsweise gering; auch ist die zusätzliche Bitposition des Speichers alternativ für Vergleichsstopzwecke nutzbar (s. Abschnitt 7). Im Fehlerfall lassen sich Maßnahmen einleiten, die weitere Schäden begrenzen bzw. vermeiden. Eine vollautomatische Fehlerbehandlung (Wiederanlauf und Fortsetzung der Arbeit) läßt sich aber nicht ohne weiteres organisieren, zumindest nicht bei Mikroprozessoren, die nur zwischen den einzelnen Maschinenbefehlen unterbrechbar sind. Dann bleibt an sich nur ein Hardware- Rücksetzen, um eine Fehlerbehandlung auszulösen (die Möglichkeiten der Ablaufbeeinflussung einer Mikroprozessor- CPU und deren Nutzbarkeit für die Fehlerbehandlung sind in /8.8/ und /8.9/ dargestellt).
In Multimikrorechnersystemen kann die Fehlerbehandlung einem der Mikrorechner fest zugeordnet werden, die fehlerverdächtige Funktionseinheit braucht ihre Fehler also nicht selbst zu behandeln. Nach diesem Prinzip sind das Bussystem nach /15/ und der Mikrorechner nach /17/ ausgelegt. Im Mikrorechner sind verschiedene Betriebsarten einstellbar. Im Normalfall bewirkt ein Paritätsfehler, daß die CPU im Wartezustand verbleibt. Die Fehlerbedingung wird über den Systembus signalisiert (Busleitung ERROR) und von einem speziellen Mikrorechner (BUS CONTROL; s. auch /9/) behandelt (die genaue Fehlerursache ist aus einem abfragbaren Fehlerregister ersichtlich). Dabei werden die Umstände des Fehlers analysiert, und im verdächtigen Mikrorechner wird eine spezifische Behandlungsroutine durch ein Selektives Rücksetzen (besondere Steuerfolge des Systembus nach /15/) gestartet. Da die CPU- Register nicht zugänglich sind, muß sich die Fehleranalyse darauf beschränken, einen von wenigen fest vorgesehenen Wiederanlaufpunkten auszuwählen.
Bei Mikroprozessoren, die es gestatten, den einzelnen Buszyklus zu unterbrechen (z. B. beim 68 Ø2Ø durch Erregen der BUS ERROR- Leitung), läßt sich hingegen, ein befehlsgenauer Wiederanlauf vorsehen, sofern es möglich ist, den Speicherinhalt wiederherzustellen (z. B. durch Nachladen).

Im folgenden werden 2 Schaltungslösungen zur Fehlerkorrektur vorgestellt. Das erste Prinzip beruht auf einer Kombination von Paritätsprüfung und redundanter Speicherung; im 2. Abschnitt wird eine einfache ECC- Schaltung für 8-bit- Mikrorechner beschrieben, die ohne Spezialschaltkreise auskommt und beispielsweise zur Modernisierung bzw. Ergänzung vorhandener Systeme geeignet ist.

9.1. Fehlerkorrektur durch redundante Speicherung

Das Prinzip ist in /22/ ausführlich beschrieben. Es ist besonders dazu vorgesehen, moderne DRAM- Schaltkreise in kleineren Mikrorechneranordnungen verwenden zu können. Das Einsatzgebiet ist folgendermaßen gekennzeichnet:

- es geht wesentlich um die Korrektur der "soft errors"; "harte" Ausfälle in Speicherschaltkreisen sollen zwar erkannt werden, es ist aber nicht vorgesehen, in Anwesenheit solcher Fehler die Arbeit fortzusetzen

- geringfügige Leistungsminderungen sind tragbar (verursacht durch zusätzliche Wartezustände bei manchen Zugriffen)

- moderne Speicherschaltkreise bieten eine höhere Speicherkapazität an, als oft ausgenutzt werden kann. Die Speicherkapazität erhöht sich bei jeder neuen Schaltkreisgeneration um das 4-fache. Will man eine gegebenes System modernisieren, so gewährleistet die Lösung, daß mit etwas mehr als einem Viertel an IC- Gehäusen (Speicher + Zusatzschaltungen) die verfügbare Speicherkapazität wenigstens verdoppelt wird. Das ist oft völlig ausreichend.

Bild 83 zeigt den grundsätzlichen Aufbau einer solchen Speicheranordnung. Der Speicher ist mit Paritätskontrolle versehen und wird folgendermaßen betrieben:

1. In einem Prüfmodus ist der gesamte Speicher in üblicher Weise nutzbar (ohne Kontrolle und Korrektur). Dazu muß die programmseitig schaltbare Leitung DIAG CORRECTION DISABLE aktiviert sein.

2. Ist diese Leitung inaktiv, so werden normale Zugriffe nur zur "niederen Hälfte" des Speichers geführt (das höchstwertige Bit der Speicheradresse ist stets Ø).

3. Jeder Schreibzugriff der CPU wird durch die Hardware in 2 aufeinanderfolgende Schreibzugriffe zum Speicher umgesetzt. Dabei wird die CPU im Wartezustand gehalten. Der erste Zugriff erfolgt mit der Adresse, wie sie von der CPU geliefert wird. Beim 2. Zugriff wird von der Hardware das höchstwertige Adressenbit auf 1 gesetzt. Damit wird die Information doppelt gespeichert (im Originalbereich und im Korrekturbereich).
Zwecks Zeitersparnis wird für den 2. Zugriff die Betriebsart "column mode" der RAM- Schaltkreise ausgenutzt, d. h. RAS bleibt aktiv, und nach dem Umschalten der Adresse wird lediglich CAS nochmals erregt.

4. Lesezugriffe ohne Paritätsfehler laufen ohne Besonderheiten (also ohne Zeitverlust) ab.

5. Wird bei einem Lesezugriff ein Paritätsfehler festgestellt, so startet die Hardware einen zweiten Zugriff, diesmal mit gesetztem höchstwertigen Adressenbit zum Korrekturbereich. Tritt dabei kein Paritätsfehler auf, so wird die gelesene Information in einem dritten Zyklus in den Originalbereich zurückgeschrieben. Für diese Zugriffe wird der "column mode" genutzt, und die CPU wird im Wartezustand gehalten.
Die Tatsache der Korrektur wird in einem programmseitig abfragbaren Flipflop (CORRECTION LATCH) registriert.

6. Wird beim 2. Lesezugriff auch ein Paritätsfehler erkannt, so handelt es sich um einen nicht korrigierbaren Fehler. Dies wird in einem weiteren abfragbaren Flipflop (ERROR LATCH) registriert. Die weitere Behandlung hängt vom Einsatzfall ab (Auslösen eines Hardware- Rücksetzens, Halten im Wartezustand o. ä.).

Bild 84 veranschaulicht das Funktionsprinzip anhand der Zustandsübergänge.
Die Bilder 85 - 88 zeigen Schaltungseinzelheiten:
Die Ablaufsteuerung ist gemäß Bild 85 mit einer Schieberegisterkette aufgebaut. Das erste Schieberegister liefert seine Impulse zu Beginn jedes CPU- Zugriffs; dann bleiben alle Ausgänge bis zum Verlassen des Wartezustandes aktiv (RAS und SWITCH ADRS für die Adressenumschaltung am Speicher). Das zweite Schieberegister läuft bei jedem Teilzugriff neu an (es wird zwischenzeitlich mit END OF ACCESS zurückgesetzt), um die CAS- und Lade- bzw. Schreibimpulse zu liefern.
Gemäß Bild 86 wird END OF ACCESS nach jedem Teilzugriff aktiv. Damit wird auch das Setzen der Fehlerflipflops gesteuert. CLEAR LATCHES ist ein programmseitig auslösbarer Rückstellimpuls.
Die Korrektursteuerung besteht nach Bild 87 aus einem Funktionszuordner und 2 Statusflipflops (STAT Ø, STAT 1). Die angegebenen kombinatorischen Gleichungen ergeben sich unmittelbar aus den Zustandsdiagrammen (Bild 84).
Bild 88 zeigt die Beschaltung der RAM- Eingänge für einen 8-bit- Mikrorechner. Bei Schreibzugriffen der CPU wird deren Datenbus zum RAM durchgeschaltet, bei Korrekturzugriffen wird das gelesene Byte über das Zwischenregister dem RAM wieder zugeführt (bei aktivem END OF ACCESS werden beide Baustufen ausgangsseitig deaktiviert, so daß während des Umschaltens die tri state- Verbindung inaktiv ist).
Die wesentlichen Abläufe sind in Taktdiagrammen dargestellt (Bild 89 - 91).

Weitere Hinweise:

1. In der gezeigten Anordnung kann die CPU mit gesetztem höchstwertigen Adressenbit direkt zum Korrekturbereich zugreifen. Bei solchen Zugriffen ist die Korrekturhardware unwirksam.

2. In manchen Anwendungsfällen ist es ausreichend, nur besonders kritische Speicherbereiche mit Fehlerkorrektur zu versehen. Das betrifft Programmbereiche (wenn das Befehlslesen der CPU nicht unterbrechbar ist), Stackbereiche usw. Für Datenbereiche, deren Inhalt im Fehlerfall leicht wiederhergestellt werden kann (z. B. durch Nachladen) reicht eine einfache Paritätskontrolle. Ob für bestimmte Speicherbereiche eine Fehlerkorrektur vorgesehen ist oder nicht, läßt sich beispielsweise durch die CPU- Adresse steuern, die erforderlichenfalls erweitert wird (s. Abschnitt 2). Bild 92 veranschaulicht eine solche Speicheraufteilung.

3. In der gezeigten Anordnung wird ein fehlerhaftes Bit mit Information aus dem selben Speicherschaltkreis korrigiert (bei "soft errors" in 64 kbit- bzw. 256 kbit- Schaltkreisen nach bisherigen Erkenntnissen unkritisch). Das wird vermieden, wenn die Korrekturinformation zyklisch in jeweils andere Speicherschaltkreise eingetragen und beim Lesen wieder sinngemäß umgeordnet wird. Bild 93 zeigt die notwendigen Schaltmittel.
Bei Einsatz mehrerer Speicherschaltkreise pro Bit läßt sich

die Umordnung vermeiden, indem Original- und Korrekturbereiche von vornherein verschiedenen Gruppen von RAM- Schaltkreisen zugeordnet werden. Das erfordert aber u. U. eine kompliziertere Steuerung für die Teilzyklen beim Schreiben und Korrigieren.

4. Jede Korrekturschaltung, die gegen "soft errors" wirken soll, muß durch programmseitige Maßnahmen unterstützt werden, nämlich:
- zyklisches Inspizieren des gesamten Speichers, da sich sonst in Bereichen, zu denen längere Zeit nicht zugegriffen wird, "soft errors" ansammeln können (in der beschriebenen Anordnung muß dazu der Korrekturbereich von der CPU aus direkt zugänglich sein; vgl. Hinweis 1)
- Registrieren der korrigierten Fehler (wenigstens überschlagsmäßig), um zwischen sporadischen "soft errors" und Anzeichen von Schaltkreis- Ausfällen unterscheiden zu können.

5. Die Korrekturhardware sollte zu Prüfzwecken abschaltbar und ihrerseits prüfbar sein. Dazu hat es sich bewährt (/17/), eine programmseitig steuerbare Antivalenzverknüpfung in die Paritätsbit- Eingangsleitung des Speichers einzufügen (in Bild 88 bereits angedeutet). Bei aktivierter Steuerleitung DIAG PARITY INVERT lassen sich so Paritätsfehler im Speicher erzeugen.

9.2. ECC- Anordnung für 8-bit- Mikrorechner

Die Einbußen an Verarbeitungsleistung und nutzbarer Speicherkapazität sind in manchen Fällen nicht tragbar (das betrifft z. B. den Mikrorechner nach /26/; s. auch Abschnitt 11). Dann ist gelegentlich eine "echte" ECC- Anordnung notwendig. Bild 94 gibt einen Überblick über eine solche Schaltung. 8 Datenbits werden mit 4 ECC- Bits ergänzt. Damit können 1-bit-Fehler korrigiert, aber 2-bit- Fehler nicht erkannt werden (vertretbarer Kompromiß für viele Einsatzfälle). Die Bildung des Fehlerkorrekturcodes ist in Bild 95 gezeigt. Gemäß Bild 96 läßt sich dazu ein 512 x 4 bit- PROM (Schottky- TTL) vorteilhaft einsetzen. Wesentlich ist (s. obenstehenden Hinweis 5), die ECC- Bitpositionen des Speichers ihreseits prüfen zu können. Dazu ist an den 9. Adresseneingang des PROM eine programmseitig schaltbaren Steuerleitung angeschlossen, und die ECC- Bits sind in einem Sondermodus über eine zusätzliche Auswahlschaltung (s. Bild 94) von der CPU rücklesbar. Der PROM- Inhalt bestimmt sich folgendermaßen:

- bei inaktivem ECC DATA ACCESS müssen die ECC- Bits gemäß Bild 95 gebildet werden
- bei aktivem ECC DATA ACCESS sind die niederen 4 Datenbits der CPU direkt zu den Speichereingängen durchzusteuern.

Bild 97 zeigt das Korrekturnetzwerk, das den RAM- Ausgängen nachgeordnet ist. Für jede der 4 ECC- Bitpositionen ist ein

Paritätsprüfer so beschaltet, wie dies in Bild 95 angegeben ist. Jede Ausgangsbelegung der Paritätsprüfer bezeichnet somit genau eine fehlerhafte Bitposition. Diese binär codierte Angabe wird mittels Decoder in eine 1-aus-n- Angabe umgeschlüsselt. Das durch eine Ø bezeichnete fehlerhafte Bit wird durch das jeweilige Antivalenzgatter invertiert.
Mit ECC EFFECTIVE wird den Steuerschaltungen des Mikrorechners angezeigt, daß eine Korrektur erfolgt ist (die Hardware korrigiert nur Fehler im Datenbyte; ein Fehler in den ECC- Bits wird lediglich signalisiert).

Eine solche RAM- Anordnung wird vorzugsweise dann eingesetzt, wenn es gilt, Wartezustände zu vermeiden. Deshalb ist ein automatisches Rückschreiben korrigierter Bytes nicht möglich. Stattdessen ist folgende Verfahrensweise vorgesehen:
ECC EFFECTIVE veranlaßt, daß der höhere Teil der Speicheradresse in einem programmseitig lesbaren BACKUP- Register gerettet wird. Weiterhin wird ein NMI ausgelöst. So wird ein Fehlerbehandlungsprogramm aktiviert, das den geretteten Abschnitt der Adresse liest und den betreffenden Speicherbereich mit korrekten Daten lädt; und zwar einfacherweise durch einen Blocktransport (Daten lesen - dabei werden Fehler korrigiert - und wieder zurückschreiben - damit gelangen sie korrekt wieder in den Speicher). Für solche Abläufe muß die NMI- Auslösung programmseitig unterdrückbar sein.
Der Ausfall eines Speicherschaltkreises äußert sich in ständigen Korrekturen. In einem solchen Fall kann eine Fehlerbehandlungsroutine veranlassen, daß der NMI seitens der ECC-Anzeige ständig verhindert bleibt, so daß die Arbeit trotzdem (mit gewissem Funktionsrisiko) fortgesetzt werden kann. Eine solche Betriebsart sollte aber eindeutig angezeigt werden (LED, Hinweis auf Bildschirm usw.).

10. Speicheranordnungen mit Videoanschluß

Das Prinzip, einem Speicher zusätzliche Bitpositionen beizuordnen, kann auch genutzt werden, um Mikrorechner für graphische Anzeigen auf Raster- Sichtgeräten auszubilden, ohne dazu Spezialschaltkreise (CRT- Controller) einzusetzen.
In /12/ ist die Lösung für einen Heimcomputer beschrieben, und in /21/ ist dargestellt, wie das Prinzip in einer Anordnung genutzt wird, die gleichzeitig 2 Sichtgeräte betreibt und eine Vielzahl von Spezialeffekten (Fensterdarstellungen, Überblendungen usw.) ermöglicht.

Bild 98 zeigt eine einfache Ausführung eines solchen Mikrorechners. Für dessen RAM sind 2 Zugriffsmöglichkeiten vorgesehen:
1. übliche Lese- und Schreibzugriffe der CPU
2. Lesezugriffe für die Bilddarstellung, wobei die Adresse von einem besonderen Adressenzähler geliefert wird und die Daten zu einer Videosteuerschaltung gelangen.

Der RAM wird von einer Zeitteilungsvermittlung in einem festen Rhythmus angesteuert und zyklisch zwischen CPU- und Videozugriffen umgeschaltet. Die CPU wird durch Wartezustände mit diesem zeitstarren Zugriffsschema synchronisiert (zur Zeitteilungsvermittlung s. auch /8.15/, /8.16/ und /16/). Die Bilder 99 und 100 zeigen Taktdiagramm und Schaltungseinzelheiten.
Der RAM ist durch ein Zusatzbit erweitert. Bei CPU- Zugriffen kann dies über ein Modusregister geladen werden (Bild 101). Bei Lesezugriffen der CPU wird es nicht berücksichtigt (Vorkehrungen zum diagnostischen Rücklesen bleiben hier außer Betracht).
Bei Videozugriffen kennzeichnet das Zusatzbit, daß der RAM an der betreffenden Position ein Video- Steuerwort enthält, das in das Steuerregister der Video- Steuerschaltung (Bild 102) übernommen wird.
Enthält der RAM kein Video- Steuerwort (Zusatzbit = 0), so wird dessen Inhalt im Verlauf von 2 Zugriffs- Zyklen seriell als Videoinformation zum Sichtgerät übertragen (sofern im Steuerregister erlaubt).
Mit dem Video- Steuerwort läßt sich unmittelbar beeinflussen:
- das Synchronsignal für das Sichtgerät (SYNC)
- das Rücksetzen des Adressenzählers (FLYBACK)
- das Erlauben bzw. Verhindern der Darstellung (UNBLANK CRT)
- verschiedene Darstellattribute, wie Helligkeit bzw. Farbe.

Das jeweilige Bildraster wird also ausschließlich vom RAM-Inhalt bestimmt!
Nach dem Einschalten muß der Mikrorechner die Steuerworte entspechend in den RAM laden. Dazu sind zunächst alle Zusatzbits zu löschen. Anschließend müssen die Steuerworte für die Bildsynchronisation und das Rücksetzen des Adressenzählers geladen werden (vorheriges Löschen des eigentlichen Bildbereichs im RAM ist zu empfehlen).
Erst nach dem Laden ist durch Setzen des Modusregisters die Auswertung der Videozugriffe seitens der Videosteuerschaltungen zu erlauben (VIDEO ENABLE).

Der grundsätzliche Vorteil einer solchen Anordnung besteht im geringen Aufwand und der freien Programmierbarkeit. Für Einsatzfälle, die nur eine geringe Auflösung der Bildschirm-Darstellung erfordern (Heimcomputer, manche Prozeß- bzw. Maschinensteuerungen), lassen sich so befriedigende Schaltungslösungen schaffen.
Zu beachten ist, daß auch die Austastlücken den Bildspeicher belegen. Das kann ein gewisser Nachteil sein, weil die Zugriffsroutinen darauf Rücksicht nehmen müssen. Der Mehrbedarf an Speicherplatz fällt nicht ins Gewicht, zumal die Lücken mit anderweitiger Information (Programmstücke, Daten, Arbeitsbereiche) belegt werden können (Bild 103).
Die einfache Anordnung ist nicht geeignet, wenn eine höhere Auflösung der Darstellung gewünscht ist. Der Ausweg besteht in der Erhöhung der Aufrufbreite. Um die Verhältnisse an einem einfachen Zahlenbeispiel zu verdeutlichen, sei angenommen, daß eine Bildschirmzeile mit 640 Punkten aufzulösen ist, und zwar in 32 Mikrosekunden. Für einen Bildpunkt stehen dann 50 ns zur Verfügung, für 8 Bildpunkte also 400 ns. Das ist auch die Größenordnung der Zykluszeit kostengünstiger DRAMs. Bei gleichberechtigten CPU- und Videozugriffen sind also für reine Schwarz- Weiß- Darstellungen 16 Bit parallel zu lesen (als Hinweis für die Praxis: die Anordnung sollte von vornherein für eine 16-bit- CPU ausgelegt werden). Für die Darstellung von Grauwerten bzw. Farben sind dann weitere parallele Speicher vorzusehen. Es ist aber grundsätzlich nur 1 Zusatzspeicher notwendig!
Hinweis: Der Vorteil der Lösung besteht in ihrer schaltungstechnischen Einfachheit. Es sollte nicht versucht werden, die Nachteile durch weitere Schaltungsmaßnahmen zu beseitigen; für höhere Anforderungen ist die Nutzung von Videocontroller-Schaltkreisen sicherlich der bessere Ausweg.

11. Vermeidung von Wartezuständen in Mikrorechnern

Wenn das potentielle Leistungsvermögen einer Mikroprozessor-CPU voll ausgenutzt werden soll, dürfen die Speicherzugriffe nicht durch Wartezustände verlängert werden. In einfachen Mikrorechnern ist dieses Ziel durch Einsatz ausreichend schneller Speicherschaltkreise zu erreichen. Hingegen sind bei umfangreicheren Systemen Wartezustände in 2 Fällen kaum vermeidbar:

1. Die Speicheranordnung ist mit DRAM- Schaltkreisen großer Speicherkapazität aufgebaut, die eine sequentielle Ansteuerung erfordern (Zuführung der Adresse in 2 Abschnitten), und die Zeitbedingungen gestatten es nicht, den gesamten Zugriff in der Zeit der aktiven CPU- Speicheranforderung (MEMORY REQUEST) zu beenden.

2. Es sind verschiedene unabhängige Zugriffe zu der Speicheranordnung vorgesehen (vom Systembus, von einem autonomen Refresh- Adressenzähler usw.). Ein typisches Beispiel für einen solchen Mikrorechner ist in /17/ beschrieben (vgl. auch /8.11/ - /8.14/). In einer solchen Anordnung kann nicht garantiert werden, daß jede CPU- Anforderung unverzüglich ausgeführt wird.
Es ist deshalb notwendig, auf die Anforderung hin sofort den Wartezustand einzuschalten (Aktivierung des WAIT- Einganges der CPU). Dieser darf erst dann wieder inaktiv werden, wenn eindeutig feststeht, daß der Zugriff im Rahmen des Zeitschemas der CPU zu Ende geführt werden kann (im einfachsten und sichersten Fall also erst nach Vermittlung und Ausführung des Speicherzugriffs, wie in /8.13/ erläutert).

Für die Entscheidung über sofortiges Ausführen oder zeitweiliges Zurückstellen des Zugriffs wird eine gewisse Zeit benötigt. Die meisten CPU- Typen erwarten diese Entscheidung (durch Abfrage des mit WAIT, READY o. ä. bezeichneten Einganges) bereits kurz nach Stellen der Speicheranforderung. In einem solchen Zeitintervall können die Vermittlungsschaltungen nicht über die Vergabe des nächsten Speicherzyklus entscheiden, so daß die besagte Leitung prophylaktisch erregt werden muß. Dadurch wird pro Zugriff wenigstens ein Taktzyklus mehr benötigt (Verlangsamung um etwa 20 - 30%). Somit ist die potentielle Geschwindigkeit gerade moderner schneller CPUs nicht voll ausnutzbar. Darauf wird in /7/ hingewiesen, und es wird ein neuer CPU- Typ vorgestellt, bei dem die entsprechende Abfrage zeitlich verschoben ist. So verbleibt für die Entscheidung ein nennenswertes Zeitintervall, wodurch ein Warten in der Regel vermieden werden kann.
Um mit herkömmlichen CPU- Typen ähnliche Effekte zu erreichen, kann die Mikrorechneranordnung nach Prinzipien gestaltet werden, wie sie in /26/ beschrieben sind.

Bild 104 zeigt das Blockschaltbild einer solchen Anordnung, die in der konkreten Ausführung als Modernisierung der Lösung gemäß /17/ angesehen werden kann. Die gewünschte Wirkung wird

hier durch 2 Maßnahmen erreicht:

1. Die Adressen- und Datenwege von der CPU zum betreffenden Speicher (hier: RAM) werden so kurz wie möglich gehalten, d. h. der niedere Teil der CPU- Adresse, der sofort zu Beginn des Zugriffs benötigt wird, gelangt unmittelbar von den CPU- Datenleitungen über eine einzige Auswahlschaltung zu den Speicherschaltkreisen; gelesene Daten werden über ein besonderes Ausgangsregister unter Umgehung von Pufferstufen unmittelbar auf den Datenbus der CPU geschaltet.

2. Die Vermittlungs- und Ablaufsteuerschaltungen für den RAM sind so erweitert, daß gewissermaßen vorbeugend Anforderungen für CPU- Zugriffe gestellt werden, sobald die CPU eine neue Adressenbelegung liefert. Liegen keine Anforderungen höherer Priorität vor, so wird ein solcher Zugriff vermittelt, und die Warteleitung wird sofort wieder zurückgesetzt, noch bevor die CPU deren Belegung abfragt. Hat die CPU tatsächlich einen RAM-Zugriff angefordert, so wird ein Gültigkeitsflipflop gesetzt und der Zugriff wirklich ausgeführt, also die Datenbusbelegung der CPU in den RAM geschrieben bzw. die gelesene Information auf den Datenbus der CPU geschaltet. Andernfalls wird der vorbeugend gestartete RAM- Zugriff als Lesezugriff zu Ende geführt, die gelesenen Daten werden aber nicht weiterverwendet.
Sind Zugriffe höherer Priorität (Systembus- bzw. Refreshzugriffe) anhängig, so werden zunächst diese vermittelt, und die CPU wird im Wartezustand gehalten. Die CPU- Anforderung unterliegt dann ebenfalls der üblichen Vermittlung (wie in /17/; s. auch /8.12/).

Die Bilder 1Ø5 - 113 zeigen die wesentlichen Schaltungsdetails. Besonderheiten werden im folgenden erläutert:

1. Eine Anforderung für vorbeugende Speicherzugriffe (LOOKAHEAD REQUEST) wird immer dann gestellt, wenn die CPU keine Anforderungsleitung aktiviert hat (weder Speicher- noch E-A-Anforderungen). Das zeigt an, daß die CPU im nächsten Taktzyklus eine neue Adresse liefern wird (Bild 1Ø5). Die Anforderung wird verhindert, wenn bereits ein Slave- Zugriff über den Systembus eingeleitet wurde (SLAVE ACCESS). Weiterhin ist es möglich, die vorbeugenden RAM- Zugriffe mittels einer programmseitig stellbaren Erlaubnisleitung (ENABLE LOOKAHEAD) zuzulassen oder grundsätzlich zu verhindern (für diagnostische Zwecke und Leistungsvergleiche von Vorteil).

2. In den Anforderungschaltungen hat die LOOKAHEAD- Anforderung die niedrigste Priorität. Jede Anforderung wird in das Anforderungsregister aufgenommen. Ein Prioritätsnetzwerk bewirkt, daß jeweils nur ein Flipflop des Zyklusregisters eingeschaltet wird. Wurde ein Zyklus vermittelt (CYCLE RUNNING), so wird das Anforderungsregister für die Dauer des Zyklus zurückgesetzt (Bild 1Ø6).

3. Die Ablaufsteuerschaltung liefert mittels eines Schieberegisters zeitversetzte Impulse (Bild 1Ø7). Daraus werden die

Ansteuersignale für den RAM abgeleitet (Bild 108), und zwar nach einem von 2 Zeitschemata:
Bei LOOKAHEAD- Zugriffen wird die RAS - SWITCH ADRS - CAS-Impulsfolge für den RAM früher gestartet, und der Zyklus wird eher beendet. Das ist durch die beschriebene Minimierung der kritischen Signalwege zwischen CPU und RAM möglich.
Hingegen muß bei Zyklen, die auf übliche Weise vermittelt wurden, mehr Zeit eingeräumt werden, damit die Adressen- und Datenwege entsprechend umgesteuert werden können.
Bild 112 zeigt den zeitlichen Ablauf eines LOOKAHEAD- Zugriffs, Bild 113 den eines "üblichen" Zugriffs.
Aus Bild 108 ist weiterhin ersichtlich, daß bei einem ungültigen LOOKAHEAD- Zugriff (LOOKAHEAD VALID inaktiv) weder Schreibimpulse für den RAM noch Ladeimpulse für dessen Ausgangsregister abgegeben werden.

4. Bild 109 zeigt, daß das Warteflipflop sofort zu Beginn der Speicheranforderung (MREQ) aktiviert wird. Ist es aktiv, so veranlaßt die Vorderflanke des folgenden CPU- Taktes das Einschalten des Gültigkeitsflipflops (LOOKAHEAD VALID), sofern der LOOKAHEAD- Zyklus zwischenzeitlich gestartet wurde. Daraufhin wird das Warteflipflop sofort wieder gelöscht . Die CPU findet dann zum Auswertezeitpunkt (fallende Flanke des besagten Taktes) keine Wartebedingung mehr vor.
Wurde die CPU- Anforderung auf übliche Weise vermittelt (LOCAL CYCLE), so wird das Warteflipflop ausgeschaltet, wenn der Zyklus soweit fortgeschritten ist, daß er in der Zeit vom Erkennen des gelöschten Wartezustandes bis zum Abschalten der Speicheranforderung (d. h. bis die CPU MREQ deaktiviert) sicher beendet werden kann (im Gegensatz zu /17/ wird also nicht auf das vollständige Ende des Zugriffs gewartet).

5. Der niedere Teil der CPU- Adresse ist ständig zum RAM durchgeschaltet, sofern nicht die Adressenumschaltung (SWITCH ADRS) oder ein anderer Zyklus außer LOOKAHEAD aktiv ist. In diesen anderen Zyklen wird auch der niedere Adressenteil über eine vorgeordnete Auswahlschaltung eingespeist (Bild 110).

12. Literaturverzeichnis

/1/ Boulaye, G. G.: Microprogramming. Carl Hanser Verlag München; MacMillan London 1975.

/2/ Brennenstuhl, H.: 16-Bit-Mikroprozessorsystem U 8000. Verlag Technik, Berlin 1987.

/3/ Grafik, W.: Aufbau und Arbeitsweise von 16-Bit-Mikroprozessoren. Verlag Technik, Berlin 1987 (Reihe Automatisierungstechnik, Band 226).

/4/ Grass, W.: Steuerwerke. Entwurf von Schaltwerken mit Festwertspeichern. Springer- Verlag, Berlin Heidelberg New York 1978.

/5/ Hedtke, R.: Mikroprozessorsysteme. Zuverlässigkeit, Testverfahren, Fehlertoleranz. Springer- Verlag, Berlin Heidelberg New York Tokyo 1984.

/6/ Kieser, H.; Meder, M.: Mikroprozessortechnik. Verlag Technik, Berlin 1986.

/7/ Mathon, J. C.: 32-bit-Mikroprozessor der 2. Generation. Elektronik, 29. 11. 85, S. 71 - 81.

/8/ Matthes, W.: Multimikrorechnersysteme. Beitragsfolge in radio fernsehen elektronik von Heft 4/1984 (33) bis Heft 12/1985 (34). Darauf wird in der Form /8.n/ Bezug genommen (n ist die Nummer der Folge). Die einzelnen Folgen sind nachstehend angegeben:

1 -	4/84,	231 - 234	11 -	3/85,	163 - 166
2 -	5/84,	299 - 302	12 -	4/85,	231 - 234
3 -	6/84,	367 - 370	13 -	5/85,	299 - 302
4 -	7/84,	435 - 438	14 -	6/85,	367 - 370
5 -	8/84,	503 - 506	15 -	7/85,	435 - 438
6 -	9/84,	571 - 574	16 -	8/85,	503 - 506
7 -	10/84,	639 - 642	17 -	9/85,	571 - 574
8 -	11/84,	707 - 710	18 -	10/85,	639 - 642
9 -	12/84,	773 - 776	19 -	11/85,	707 - 710
10 -	2/85,	95 - 98	20 -	12/85,	767 - 768.

/9/ Marschner, R.: Bedien- und Serviceprozessor EC7069.M. Rechentechnik und Datenverarbeitung, Heft 8, 1984, S. 88 - 93.

/10/ Strauss, E.: Inside the 80286. Prentice- Hall, New York 1986.

/11/ Wendt, S.: Entwurf komplexer Schaltwerke. Springer-Verlag Berlin Heidelberg New York 1974.

/12/ WP 151 237. Einrichtung zur Informationsdarstellung auf Fernsehgeräten.

/13/ WP 154 244. Speicheranordnung mit Fehlererkennungs- und Diagnoseeigenschaften, vorzugsweise für Mikrorechner.

/14/ WP 155 458. Adapteranordnung zum Anschluß an ein Standard- Interface.

/15/ WP 156 743. Bussystem zur Verbindung von logischen Funktionsmoduln.

/16/ WP 157 482. Anordnung zur Darstellung von Information auf Fernsehsichtgeräten.

/17/ WP 159 916. Mikrorechneranordnung, vorzugsweise für den Einsatz in Multimikrorechnersystemen.

/18/ WP 233 435. Speicheranordnung mit Eingabe-/Ausgabeanschluß, vorzugsweise zum Einsatz in Multimikrorechnersystemen.

/19/ WP 234 955. Speicheranordnung, vorzugsweise zum Einsatz in Mikrorechneranordnungen.

/20/ WP 244 431. Refreshanordnung für dynamische RAMs in Mikrorechnern.

/21/ WP 246 855. Ansteuerschaltung für Fernsehsichtgeräte.

/22/ WP 246 857. Speicheranordnung mit Fehlerkorrektur, vorzugsweise für Mikrorechner.

/23/ WP 246 858. Mikrorechneranordnung mit erweiterten Steuerwirkungen.

/24/ WP 246 859. Mikrorechneranordnung für interruptgesteuerte Datenübertragung.

/25/ WP 246 860. Mikrorechneranordnung mit programmgesteuertem Interfaceanschluß.

/26/ WP 254 077. Mikrorechneranordnung zum Einsatz in Multimikrorechnersystemen.

/27/ WP 258 300. Elektronische binäre Steuereinrichtung, vorzugsweise zum Einsatz in Mikrorechnersystemen.

- Für Notizen -

(Absichtlich frei gelassen: Deckblatt bei abgetrenntem Bildteil.)

Zusatzschaltungen für Mikroprozessoren

- Bildteil -

Bild 1: 64
Struktur einer elementaren Mikrorechneranordnung mit Ergänzungsschaltungen

Bild 2: 65
Prinzip der Ableitung von Sonderwirkungen durch Adressendecodierung

Bild 3: 66
Anschluß eines Mikrorechners an ein universelles Bussystem (für Masterzugriffe)

Bild 4: 67
Anschluß eines externen Interfaces für den Datenaustausch mit der CPU

Bild 5: 68
Details der Ablaufsteuerung nach Bild 4

Bild 6: 69
Details der Ablaufsteuerung nach Bild 4 (Erkennen und Signalisieren von Sonderbedingungen)

Bild 7: 70
Adressenbelegung für den direkten Datenaustausch zwischen externem Interface und Speicher

Bild 8: 71
Anschluß eines externen Interfaces für den direkten Datenaustausch mit dem Speicher

Bild 9: 72
Einzelheiten zu Bild 8 (Pufferstufe, Interface- Koppelstufe)

Bild 10: 73
Einzelheiten zu Bild 8 (Speicher- Ausgangsregister, Interface-Schreibregister)

Bild 11: 74
Anordnung zur Codewandlung

Bild 12: 75
Beschaltung eines ROM zur Codewandlung

Bild 13: 76
Anordnung für schnelle direkte Zugriffe und (wahlweise) Codewandlung

Bild 14: 77
Anordnung für Suchoperationen mit zusätzlichem Register für die Suchbedingung

Bild 15: 78
Anordnung für universelle Verknüpfungsoperationen außerhalb der CPU

Bild 16: 79
Einzelheiten zu Bild 15

Bild 17: 80
Beispiel für eine Adressenaufteilung ohne Erweiterung

Bild 18: 81
Adressenerweiterung mittels vollständiger Segmentierung

Bild 19: 82
Anordnung zur Adressenerweiterung (vollständige Segmentierung)

Bild 20: 83
Einzelheiten des Segmentadressenspeichers

Bild 21: 84
Adressenerweiterung bei teilweiser Segmentierung

Bild 22: 85
Anordnung zur Adressenerweiterung mit kombiniert genutztem RAM bei teilweiser Segmentierung

Bild 23: 86
Einzelheiten zu Bild 22 (Adressenauswahlschaltung, RAM)

Bild 24: 87
Prinzip einer einfachen Adressenerweiterung für Bitzugriffe

Bild 25: 88
Anordnung für bitorientierte Zugriffe mit einer Adresse gemäß Bild 24

Bild 26: 89
Beispiel einer Adressenstruktur, die Zugriffe mit "echten" Bitadressen ermöglicht

Bild 27: 90
Blockschaltbild einer Speicheranordnung für die Adressierung gemäß Bild 26

Bild 28: 91
Schaltungseinzelheiten zu Bild 27

Bild 29: 92
Schema des Zeitablaufs für einen Speicherzugriff

Bild 30: 93
Prinzipielle Struktur eines Gatternetzwerkes für die kombinatorischen Verknüpfungen

Bild 31: 94
Mikrorechneranordnung für die interruptgesteuerte Datenübertragung

Bild 32: 95
Beispiel einer Befehlsfolge für die Übertragung eines Datenbytes vom Interface in den RAM

Bild 33: 96
Die Adressenstruktur, die dem Beispiel zugrunde liegt

Bild 34: 97
Befehlsfolge für die Datenübertragung (wie Bild 32); die zusätzlichen Schaltungsmaßnahmen sind wirksam

Bild 35: 98
Beispiel für einen verallgemeinerten Einsatzfall (3 Sonderbedingungen werden von Schaltmitteln erkannt)

Bild 36: 99
Elementare Mikrorechneranordnung mit externen Interfaces

Bild 37: 100
Mikrorechneranordnung mit erweiterten Interfaceanschlüssen

Bild 38: 101
Erlaubnissteuerung durch Ausgabebefehl

Bild 39: 102
Beispiel einer Adressenstruktur, die zur Erlaubnissteuerung geeignet ist

Bild 40: 103
Steuerschaltung für die Interfacezugriffe des Mikrorechners

Bild 41: 104
Ablauf einer Ausgabe

Bild 42: 105
Ablauf einer Eingabe

Bild 43: 106
Einzelheiten der Speicheranordnung des Mikrorechners

Bild 44: 107
Prinzipielle Struktur eines Steuerautomaten zur Beobachtung des Befehlsstromes

Bild 45: 108
Zustandsdiagramm eines Steuerautomaten (Beispiel für Z 80)

Bild 46: 109
Blockschaltbild eines erweiterten Mikrorechners

Bild 47: 110
Überblick über die Speicheranordnung

Bild 48: 111
Speicherbelegung im CONTROL MODE

Bild 49: 112
Ablauf eines Schreibbefehls im CONTROL MODE

Bild 5Ø: 113
Struktur einer Mikrorechneranordnung, in der die Möglichkeiten des CONTROL MODE weitgehend genutzt sind

Bild 51: 114
Adressenstruktur zu Bild 5Ø

Bild 52: 115
Belegung eines Steuerwortes

Bild 53: 116
Decodierung der Speicheradressen

Bild 54: 117
Einzelheiten der Speicheranordnung

Bild 55: 118
Einfache Ausführung der CONTROL MODE- Steuerung

Bild 56: 119
Erweiterte CONTROL MODE- Steuerung (Befehle mit 2 M1- Zyklen dürfen Sonderwirkungen ausüben)

Bild 57: 12Ø
Steuerregister und Decodierung des Funktionscodefeldes

Bild 58: 121
Zeitablauf eines Zyklus sowie Erzeugung der Steuerimpulse

Bild 59: 122
Ablaufsteuerung, die Wartezustände einfügt

Bild 6Ø: 123
Zeitablauf zu Bild 59

Bild 61: 124
Ansteuerung auswärtsführender Interfaceleitungen

Bild 62: 125
Befehl mit Steuerworten zu Bild 61

Bild 63: 126
Prinzip der NOP- Aufschaltung

Bild 64: 127
NOP- Aufschaltung durch Modifikation des Adressendecoders

Bild 65: 128
NOP- Aufschaltung durch gesonderte Buskoppelstufe

Bild 66: 129
Ausgabe von Registerbelegungen der CPU auf Interfaceleitungen

Bild 67: 130
Modifikation des Schreibimpulsdecoders von Bild 53

Bild 68: 131
Eingaben von Interfacebelegungen in Register der CPU

Bild 69: 132
Zuordnung von Befehlsbytes und Steuerworten für Bild 68

Bild 70: 133
Transporte von Speicherinhalten zwischen Speicher und Interface

Bild 71: 134
Zuordnung der Speicherbereiche

Bild 72: 135
Einzelheiten der Speicheradressierung

Bild 73: 136
Steuerschaltung für die Datenübertragung

Bild 74: 137
Modifikation der Anordnung gemäß Bild 50 für die beschleunigte Datenübertragung

Bild 75: 138
Programmverzweigung mit direktem Abfragen von Interfaceleitungen

Bild 76: 139
Befehls- und Steuerwortformat zu Bild 75

Bild 77: 140
Belegung eines Steuerwortes von einem Byte Breite

Bild 78: 141
Speicheranordnung mit Zusatzspeicher für Inhaltskontrolle und Adressenvergleich

Bild 79: 142
Belegung des Steuerregisters

Bild 80: 143
Einzelheiten der Speicheransteuerung

Bild 81: 144
Einzelheiten der Vergleichsstop- Signalisierung

Bild 82: 145
Rücklesbare Ausgaberegister (Prinzipschaltung)

Bild 83: 146
Speicheranordnung mit Fehlerkorrektur auf Grundlage doppelter Speicherung und Paritätskontrolle

Bild 84: 147
Zustandsdiagramme der Steuerschaltungen

Bild 85: 148
Ablaufsteuerung

Bild 86: 149
Ende- und Fehlerflipflops

Bild 87: 150
Korrektursteuerung

Bild 88: 151
Beschaltung der RAM- Dateneingänge

Bild 89: 152
Schreibablauf

Bild 90: 153
Leseablauf mit Fehlerkorrektur

Bild 91: 154
Leseablauf bei Doppelfehler

Bild 92: 155
Speicheraufteilung für bereichsweise Fehlerkorrektur

Bild 93: 156
Schaltmittel zum Umordnen der Datenbits, so daß Original- und Korrekturbits jeweils in anderen Speicherschaltkreisen untergebracht werden

Bild 94: 157
Speicheranordnung mit ECC für 8-bit- Mikrorechner

Bild 95: 158
Bildung des fehlerkorrigierenden Codes

Bild 96: 159
Beschaltung der RAM- Dateneingänge

Bild 97: 160
Korrekturnetzwerk, das den RAM- Ausgängen nachgeordnet ist

Bild 98: 161
Mikrorechneranordnung mit Videoanschluß

Bild 99: 162
Taktdiagramm der Speicheransteuerung

Bild 100: 163
Einzelheiten der Steuerschaltungen

Bild 101: 164
Belegung des Modusregisters und besondere RAM- Belegungen

Bild 102: 165
Einzelheiten der Video- Steuerschaltungen

Bild 103: 166
Besonderheiten der RAM- Belegung

Bild 104: 167
Blockschaltbild eines Mikrorechners, der für Multimikrorechnersysteme vorgesehen ist und weitgehend ohne Wartezustände arbeitet

Bild 105: 168
Erzeugung der CPU- Anforderungen

Bild 106: 169
Überblick über die Vermittlungsschaltung

Bild 107: 170
Überblick über die Ablaufsteuerschaltung

Bild 108: 171
Einzelheiten der RAM- Ansteuerung

Bild 109: 172
Einzelheiten der Warte- und Gültigkeitsflipflops

Bild 110: 173
Adressenweg zum RAM

Bild 111: 174
Datenwege des RAM

Bild 112: 175
Ablauf eines LOOKAHEAD- Zugriffs

Bild 113: 176
Ablauf eines "normalen" CPU- Zugriffs

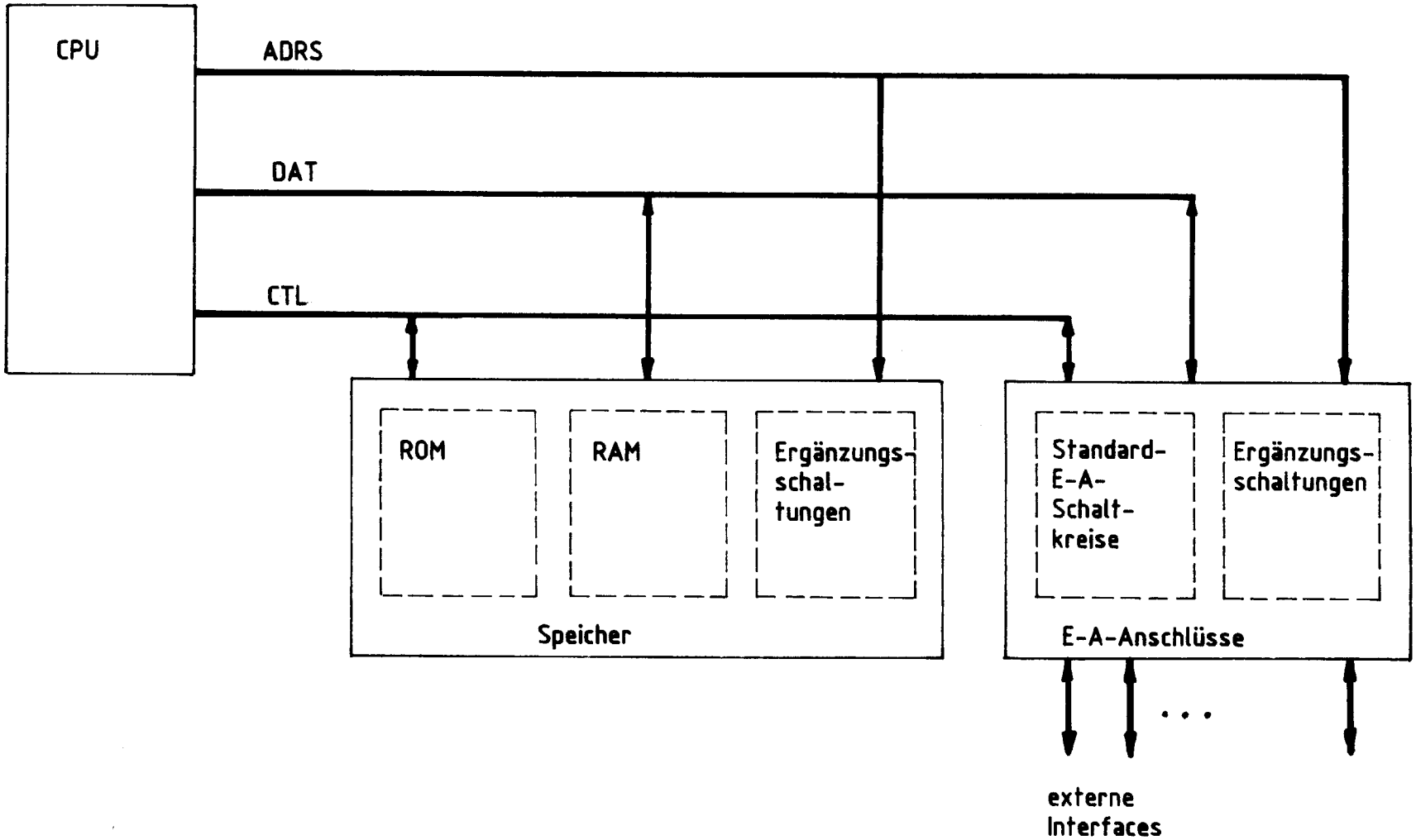

CPU
ADRS
DAT
CTL
ROM
RAM
Ergänzungs-
schal-
tungen
Speicher
Standard-
E-A-
Schalt-
kreise
Ergänzungs-
schaltungen
E-A-Anschlüsse
...
externe
Interfaces

Bild 1

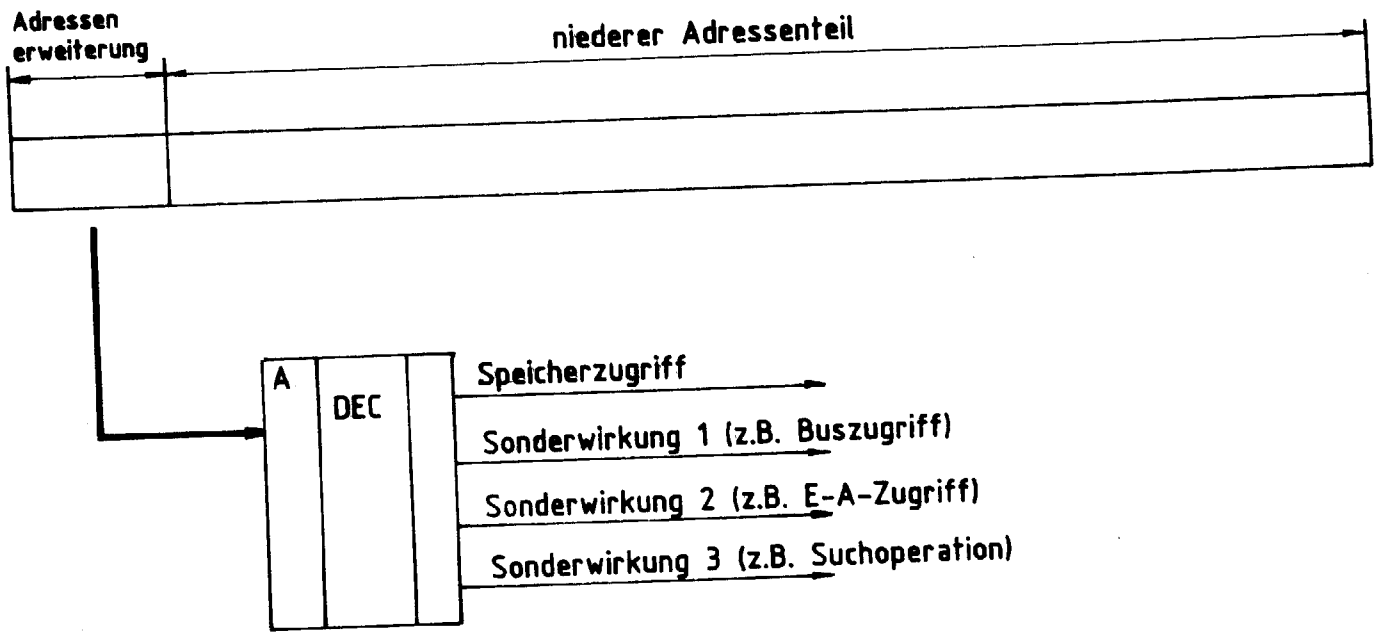

Adressen
erweiterung
niederer Adressenteil
A
DEC
Speicherzugriff
Sonderwirkung 1 (z.B. Buszugriff)
Sonderwirkung 2 (z.B. E-A-Zugriff)
Sonderwirkung 3 (z.B. Suchoperation)

Bild 2

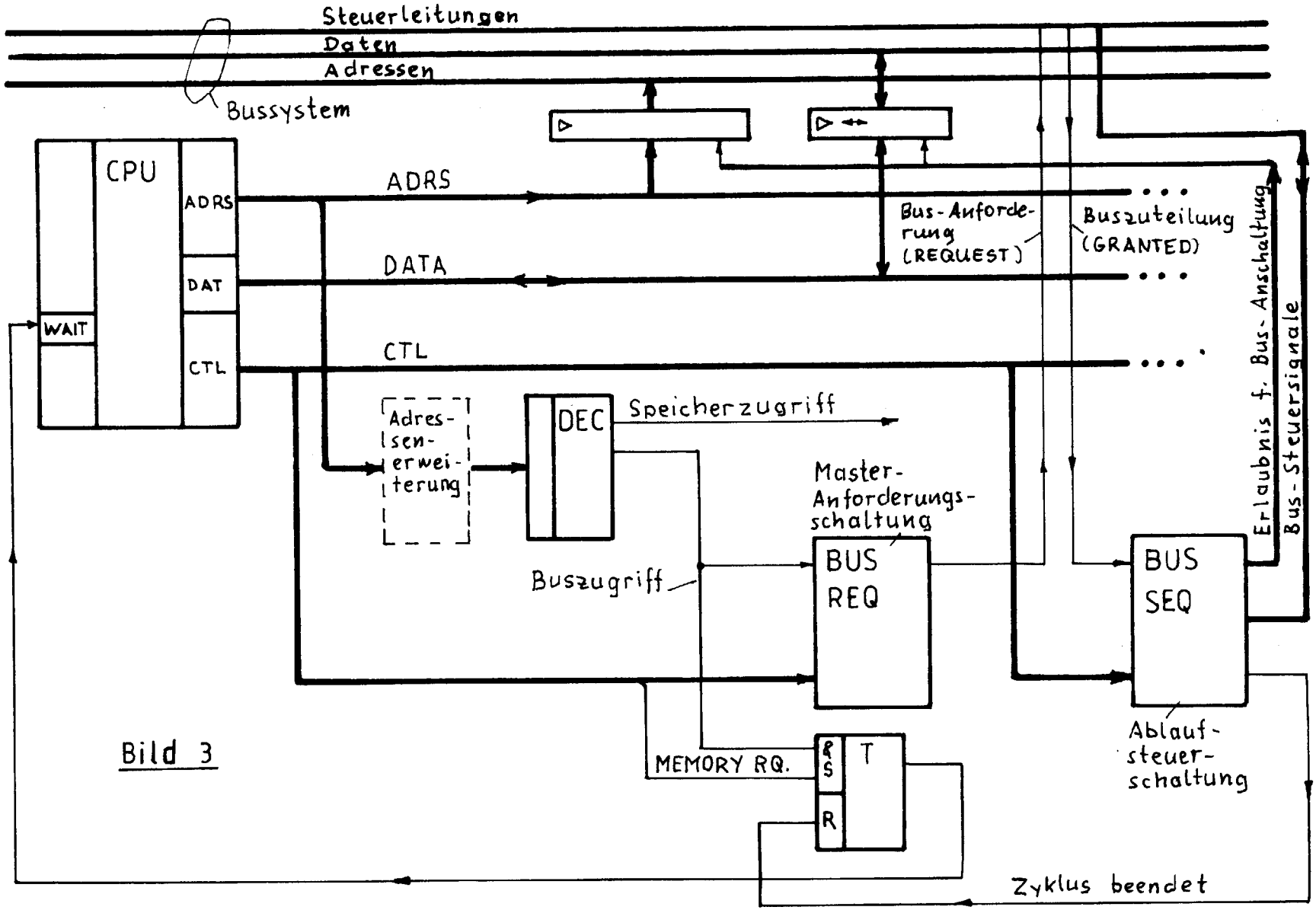

Steuerleitungen
Daten
Adressen
Bussystem
CPU
ADRS
DAT
CTL
WAIT
ADRS
DATA
CTL
Bus-Anforderung (REQUEST)
Buszuteilung (GRANTED)
Erlaubnis f. Bus-Anschaltung
Bus-Steuersignale
Adressenerweiterung
DEC
Speicherzugriff
Master-Anforderungsschaltung
BUS REQ
BUS SEQ
Buszugriff
MEMORY RQ.
S
R
T
Ablaufsteuerschaltung
Zyklus beendet

Bild 3

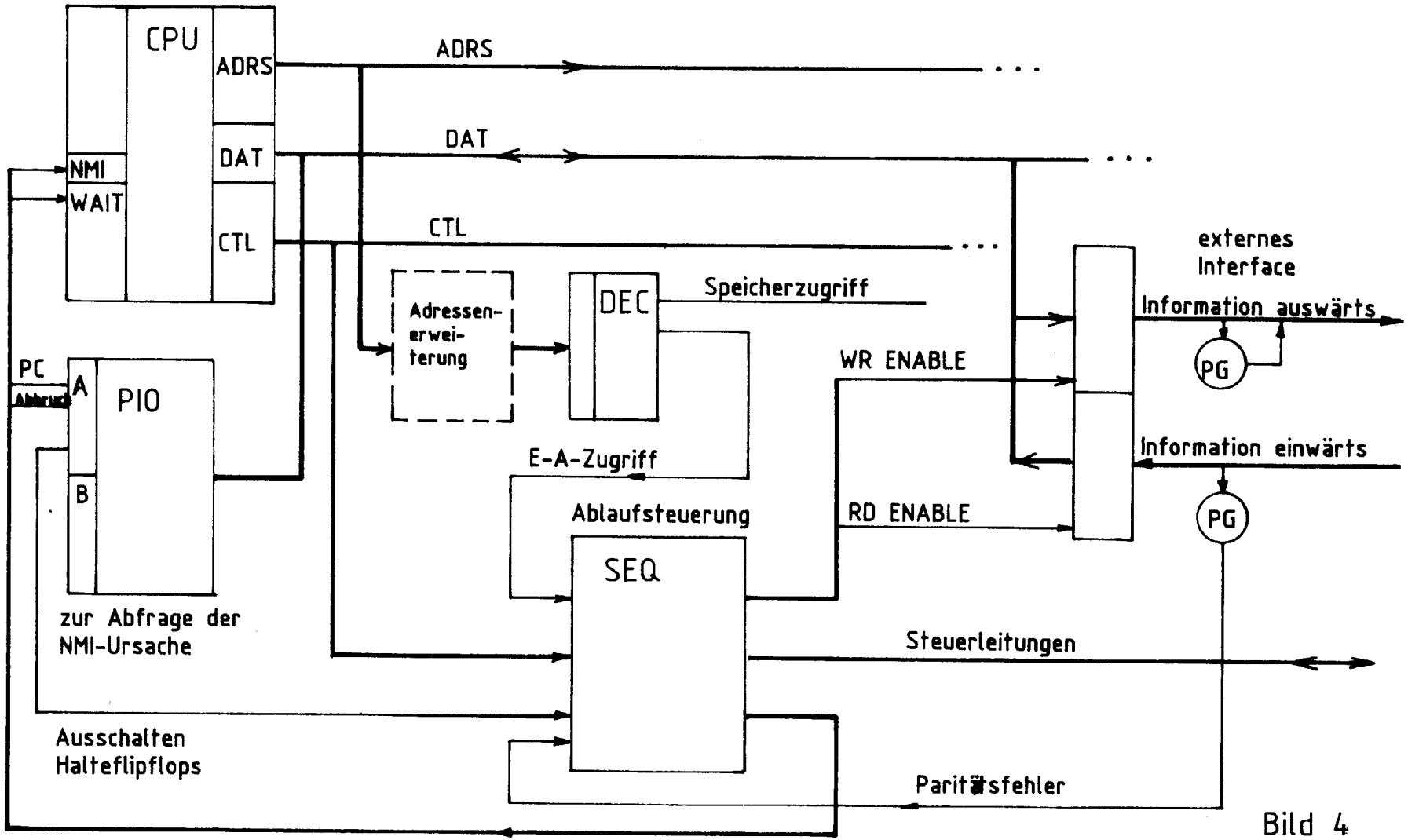

CPU
ADRS
DAT
CTL
NMI
WAIT
ADRS
DAT
CTL
externes
Interface
Information auswärts
Information einwärts
PG
PG
Adressen-
erwei-
terung
DEC
Speicherzugriff
WR ENABLE
RD ENABLE
PC
Abbruch
A
B
PIO
E-A-Zugriff
Ablaufsteuerung
SEQ
zur Abfrage der
NMI-Ursache
Steuerleitungen
Ausschalten
Halteflipflops
Paritätsfehler

Bild 4

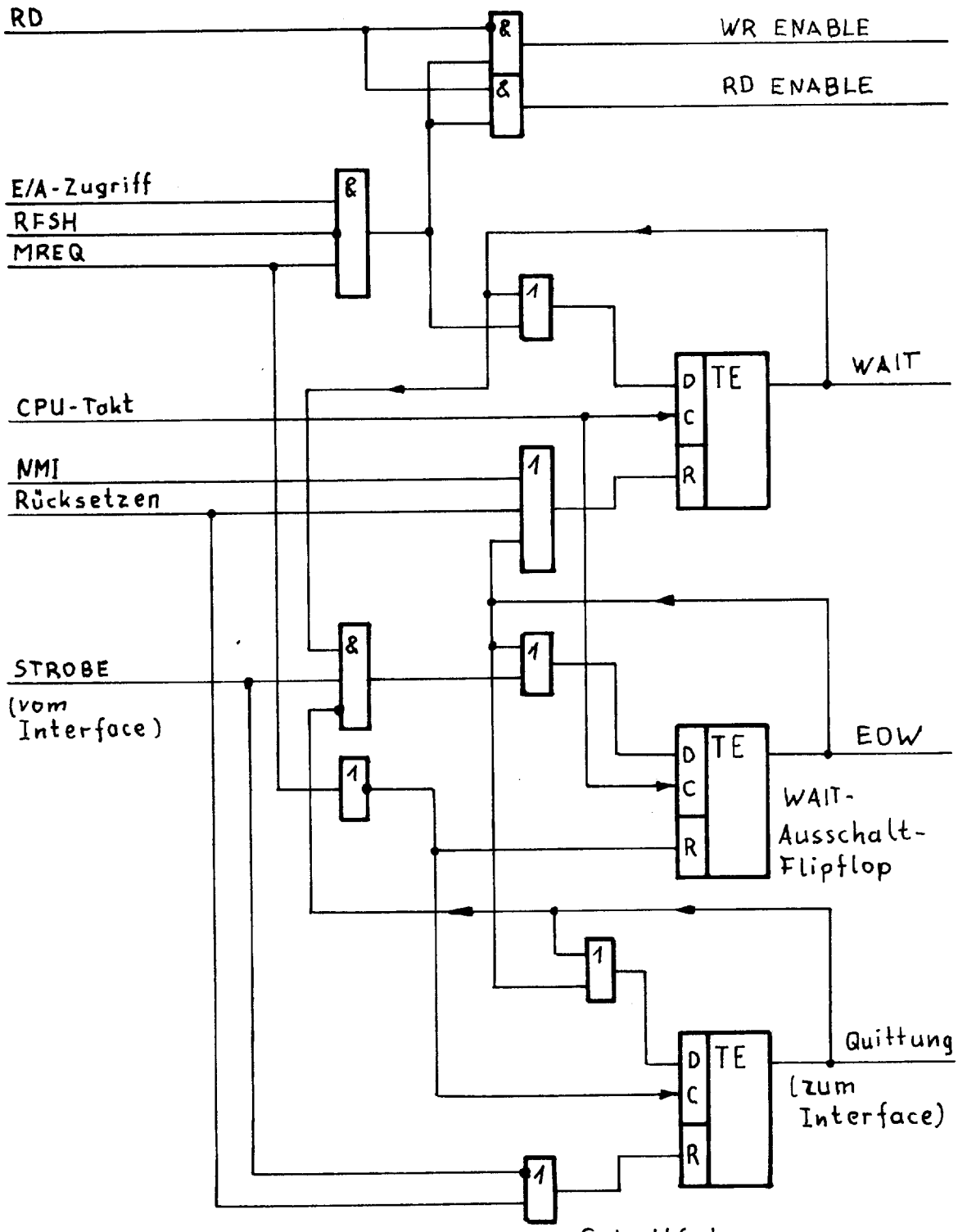
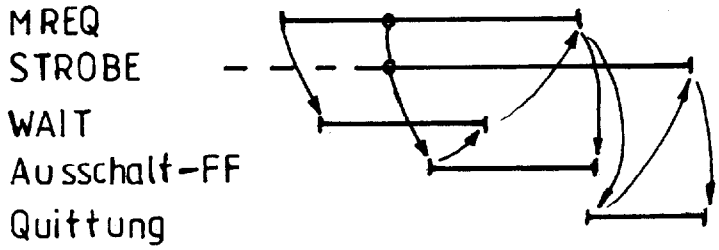


Bild 5

Schaltfolge



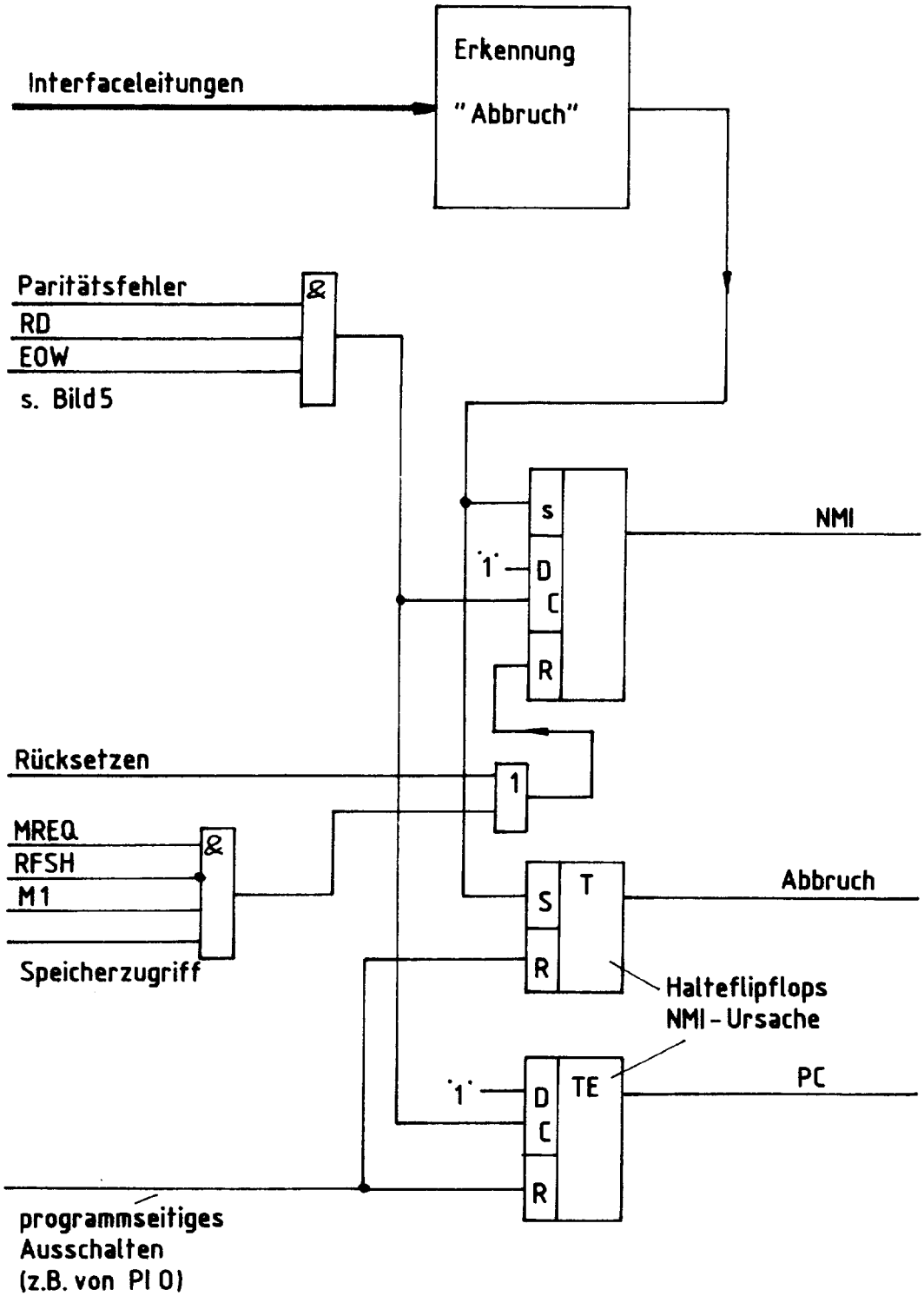

Erkennung
"Abbruch"
Interfaceleitungen
Paritätsfehler
RD
EOW
s. Bild 5
&
S
D
C
R
'1'
NMI
Rücksetzen
1
MREQ
RFSH
M1
Speicherzugriff
&
S
R
T
Abbruch
Halteflipflops
NMI - Ursache
'1'
D
C
R
TE
PC
programmseitiges
Ausschalten
(z.B. von PI O)

Bild 6

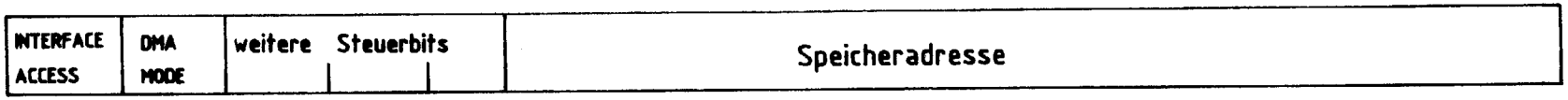

INTERFACE ACCESS
DMA MODE
weitere Steuerbits
Speicheradresse

Bild 7

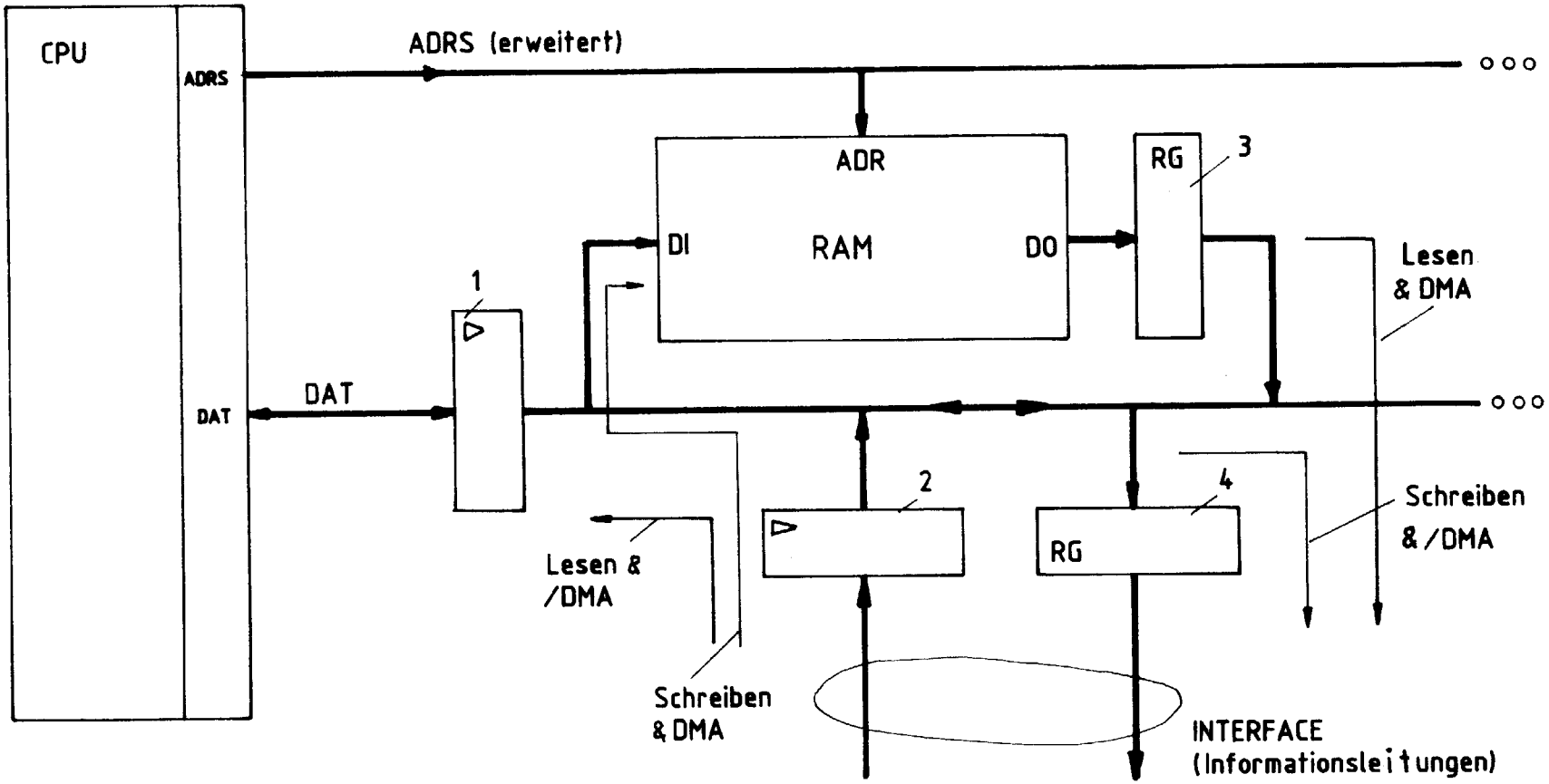
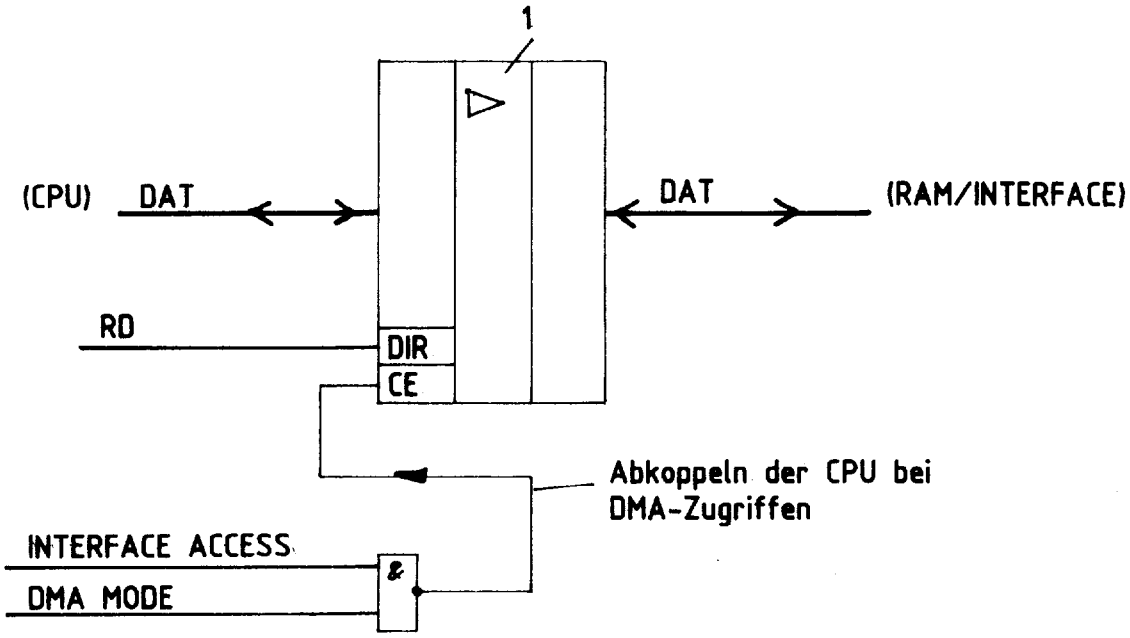

CPU
ADRS
DAT
ADRS (erweitert)
ADR
DI
RAM
DO
RG
3
1
DAT
Lesen
& DMA
2
4
RG
Schreiben
& /DMA
Lesen &
/DMA
Schreiben
& DMA
INTERFACE
(Informationsleitungen)

Bild 8


1
(CPU)
DAT
DAT
(RAM/INTERFACE)
RD
DIR
CE
Abkoppeln der CPU bei
DMA-Zugriffen
INTERFACE ACCESS
DMA MODE
&

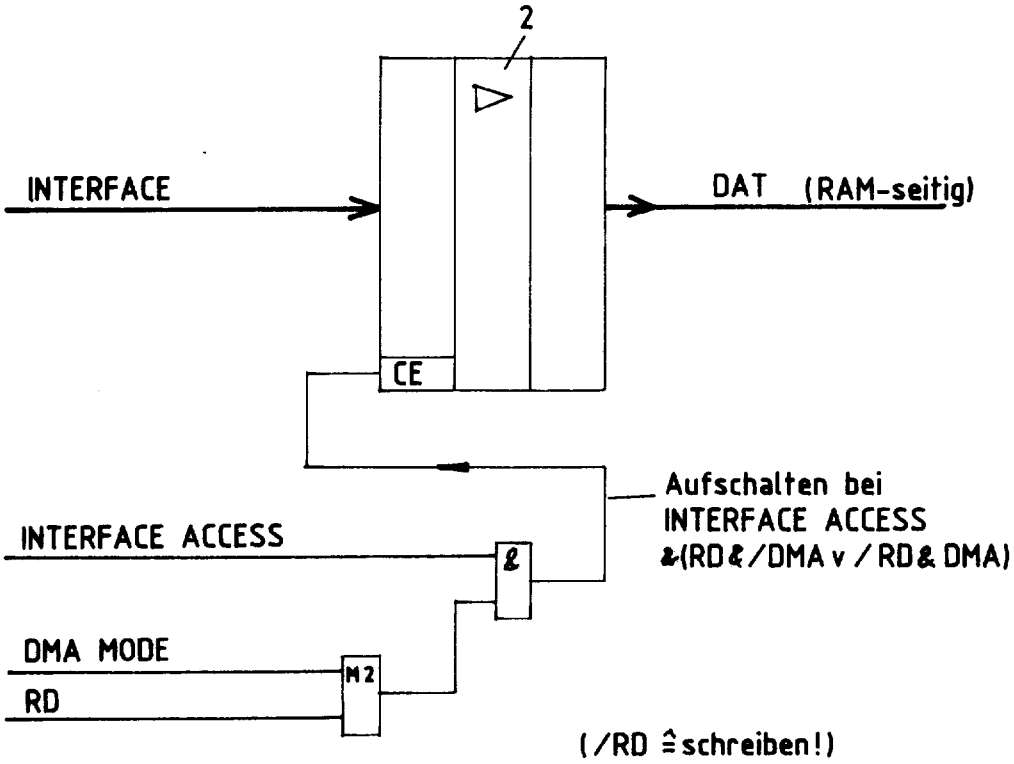

2
INTERFACE
DAT (RAM-seitig)
CE
Aufschalten bei
INTERFACE ACCESS
&(RD&/DMA v / RD& DMA)
INTERFACE ACCESS
&
DMA MODE
RD
M2
(/RD ≙ schreiben!)

Bild 9

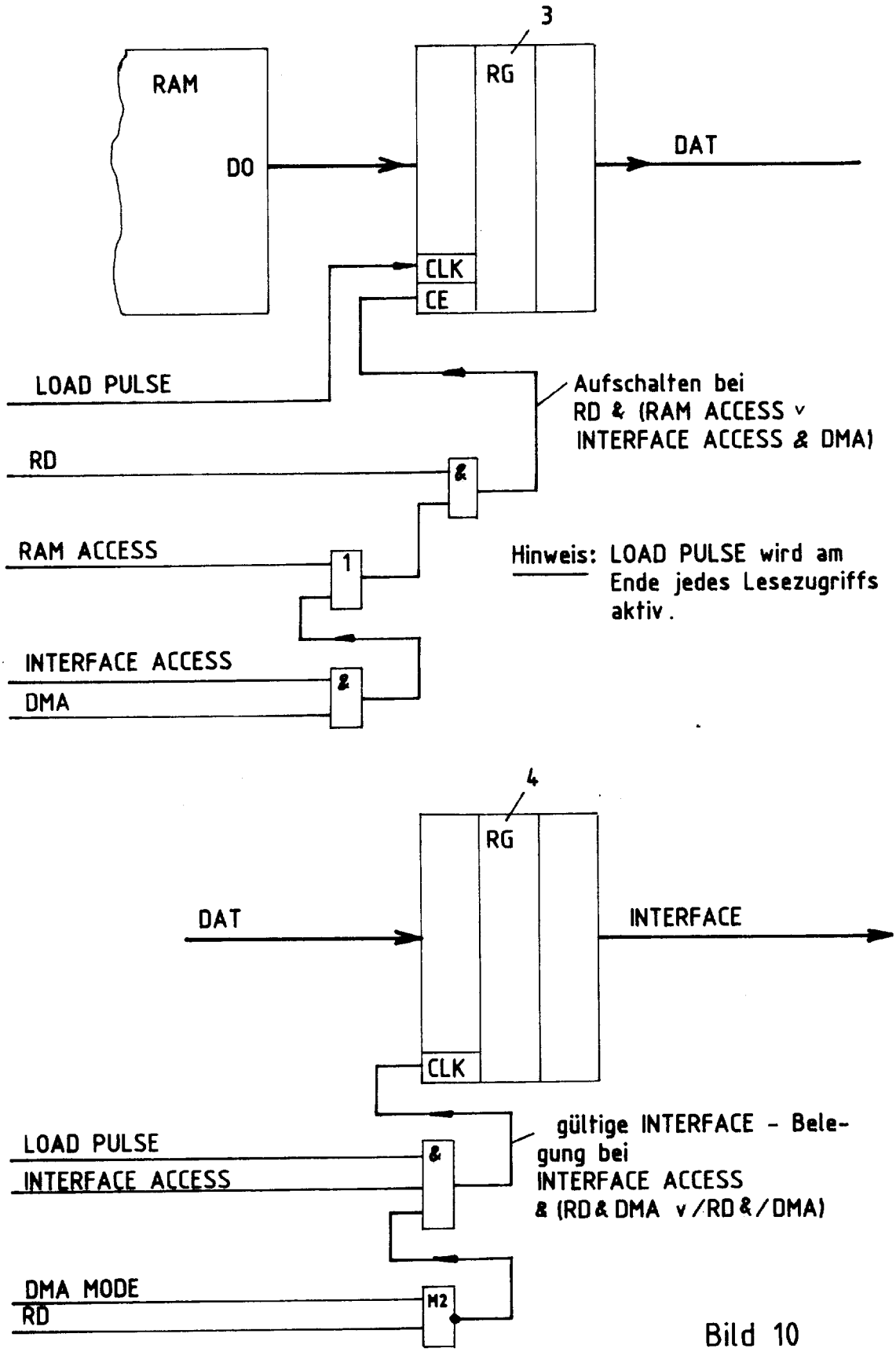
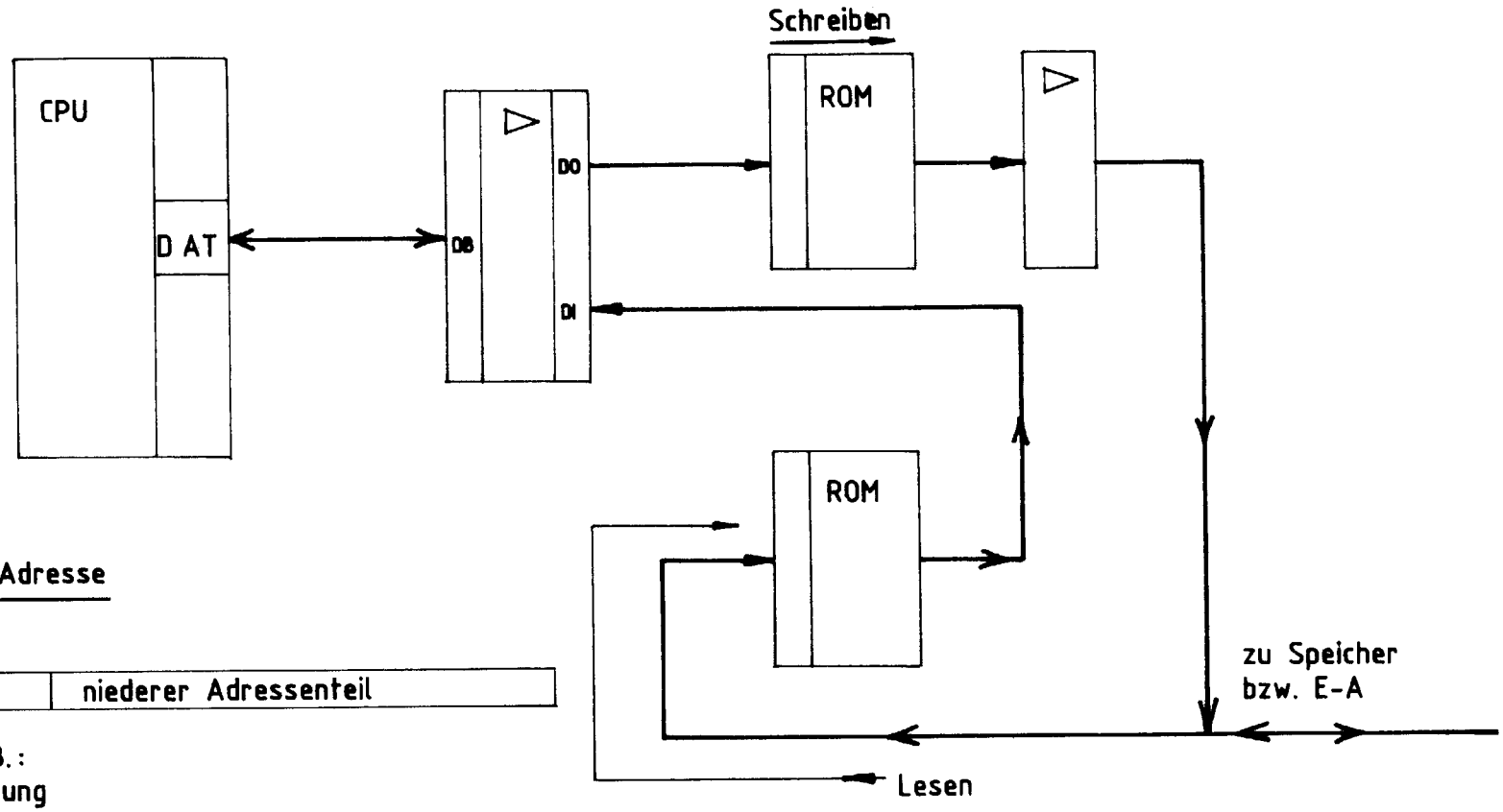

3
RAM
DO
RG
DAT
CLK
CE
LOAD PULSE
Aufschalten bei
RD & (RAM ACCESS v
INTERFACE ACCESS & DMA)
RD
&
RAM ACCESS
1
Hinweis: LOAD PULSE wird am
Ende jedes Lesezugriffs
aktiv.
INTERFACE ACCESS
DMA
&
4
RG
DAT
INTERFACE
CLK
LOAD PULSE
INTERFACE ACCESS
&
gültige INTERFACE - Bele-
gung bei
INTERFACE ACCESS
& (RD & DMA v /RD & /DMA)
DMA MODE
RD
M2

Bild 10



Struktur der Adresse

CONVERSION	niederer Adressenteil

codiert ist z.B.:
- keine Wandlung
-EBCDIC zu ASCII
-ASCII zu EBCDIC

Bild 11

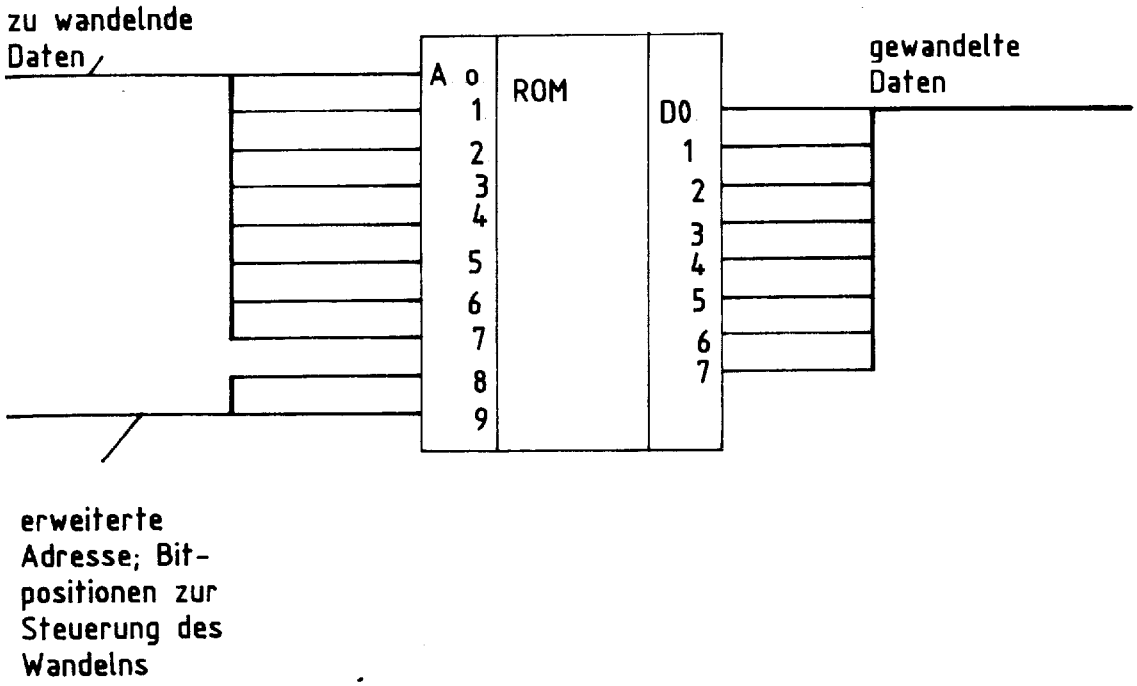

zu wandelnde
Daten
A o
1
2
3
4
5
6
7
8
9
ROM
D0
1
2
3
4
5
6
7
gewandelte
Daten
erweiterte
Adresse; Bit-
positionen zur
Steuerung des
Wandelns

Bild 12

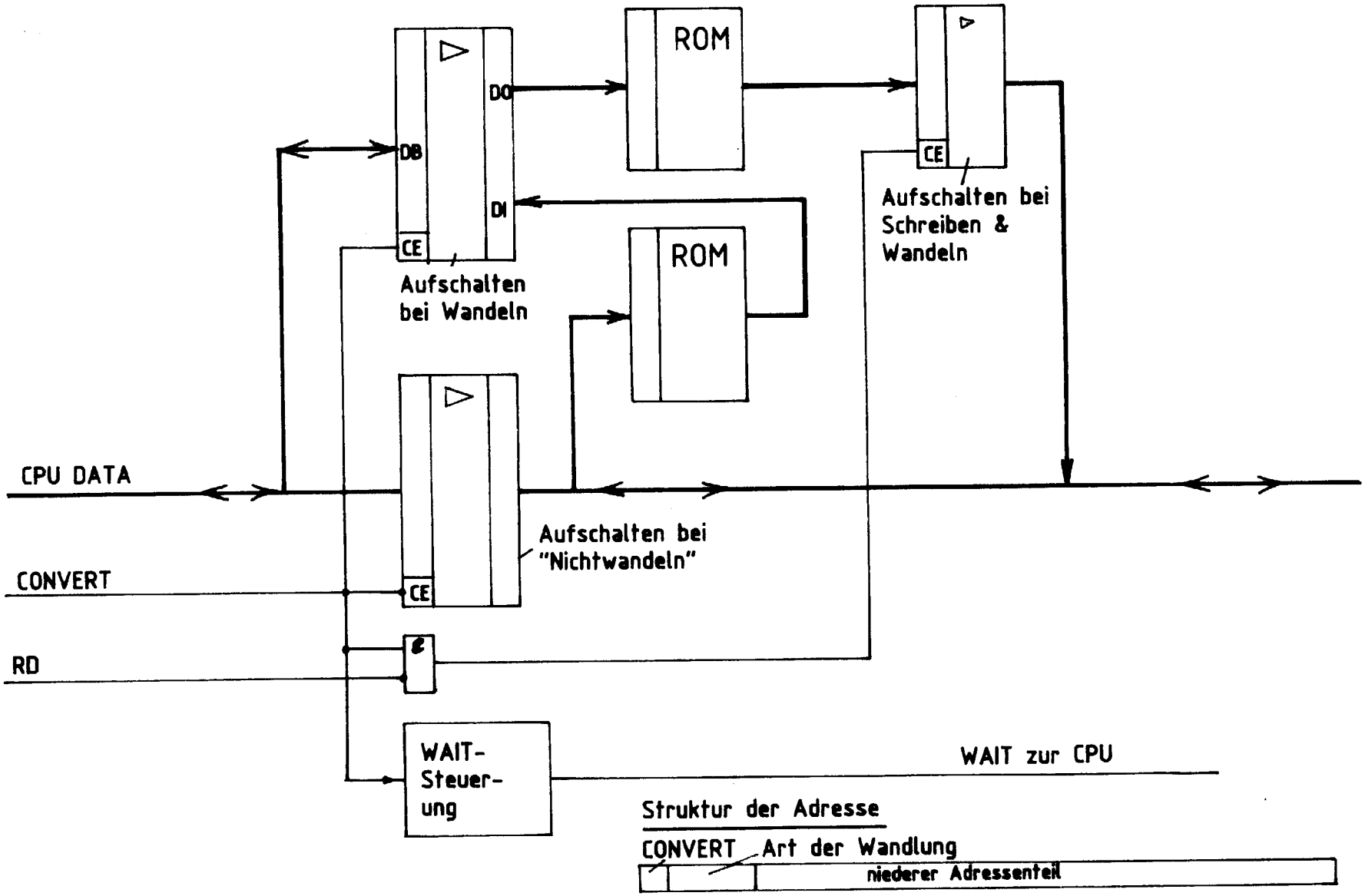
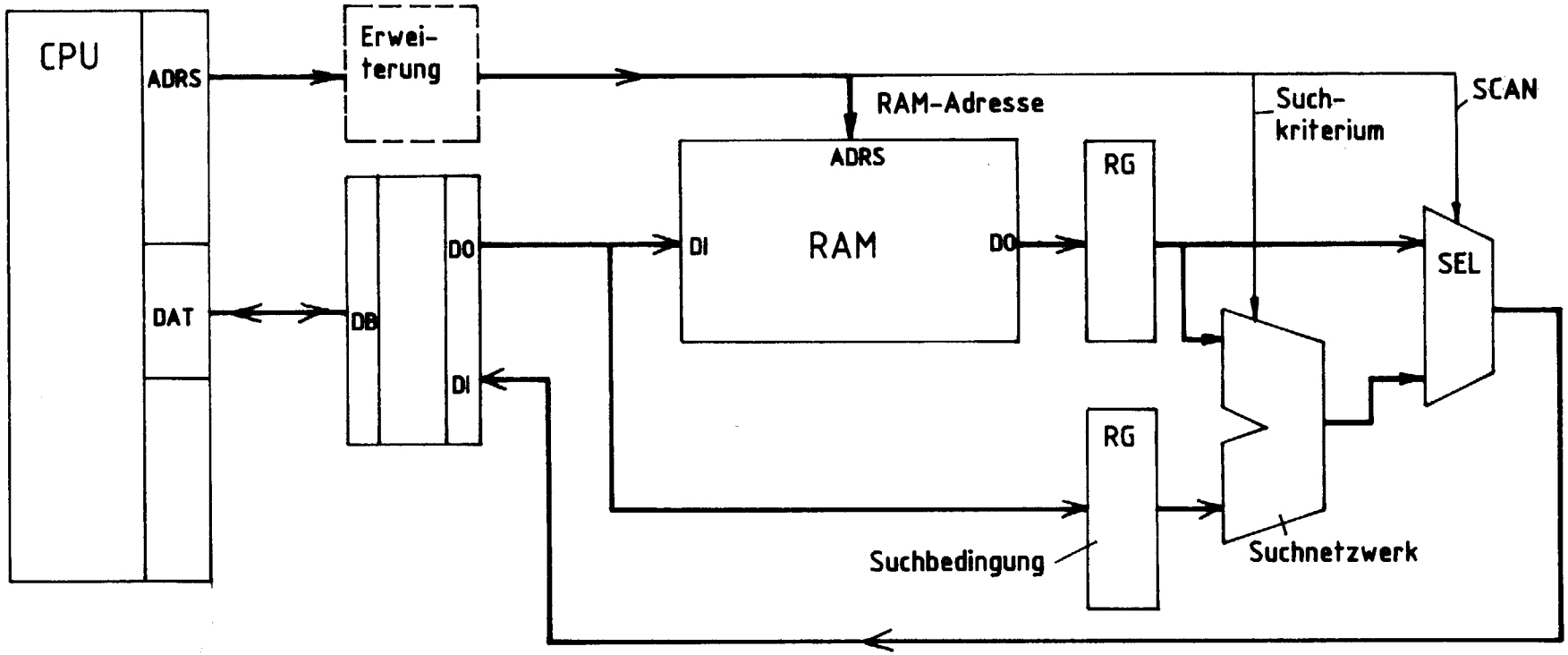

ROM
ROM
DO
DI
DB
CE
CE
CE
Aufschalten bei Wandeln
Aufschalten bei Schreiben & Wandeln
Aufschalten bei "Nichtwandeln"
CPU DATA
CONVERT
RD
WAIT-Steuer-ung
WAIT zur CPU
Struktur der Adresse
CONVERT
Art der Wandlung
niederer Adressenteil

Bild 13



Struktur der Adresse

Bild 14

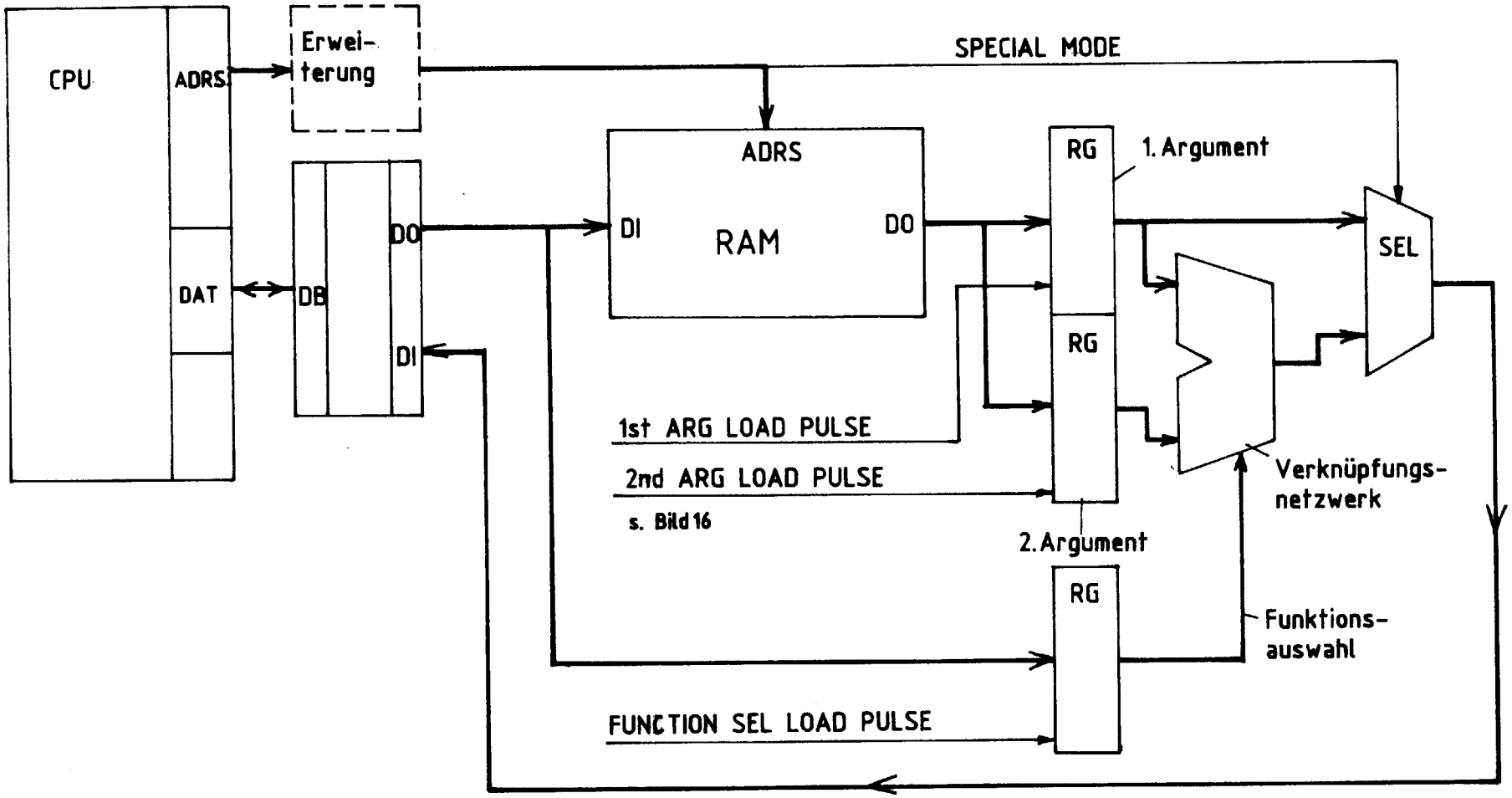

CPU
ADRS
DAT
Erwei-
terung
DB
DO
DI
SPECIAL MODE
ADRS
DI
RAM
DO
RG
1. Argument
RG
SEL
1st ARG LOAD PULSE
2nd ARG LOAD PULSE
s. Bild 16
2. Argument
Verknüpfungs-
netzwerk
RG
Funktions-
auswahl
FUNCTION SEL LOAD PULSE

Bild 15

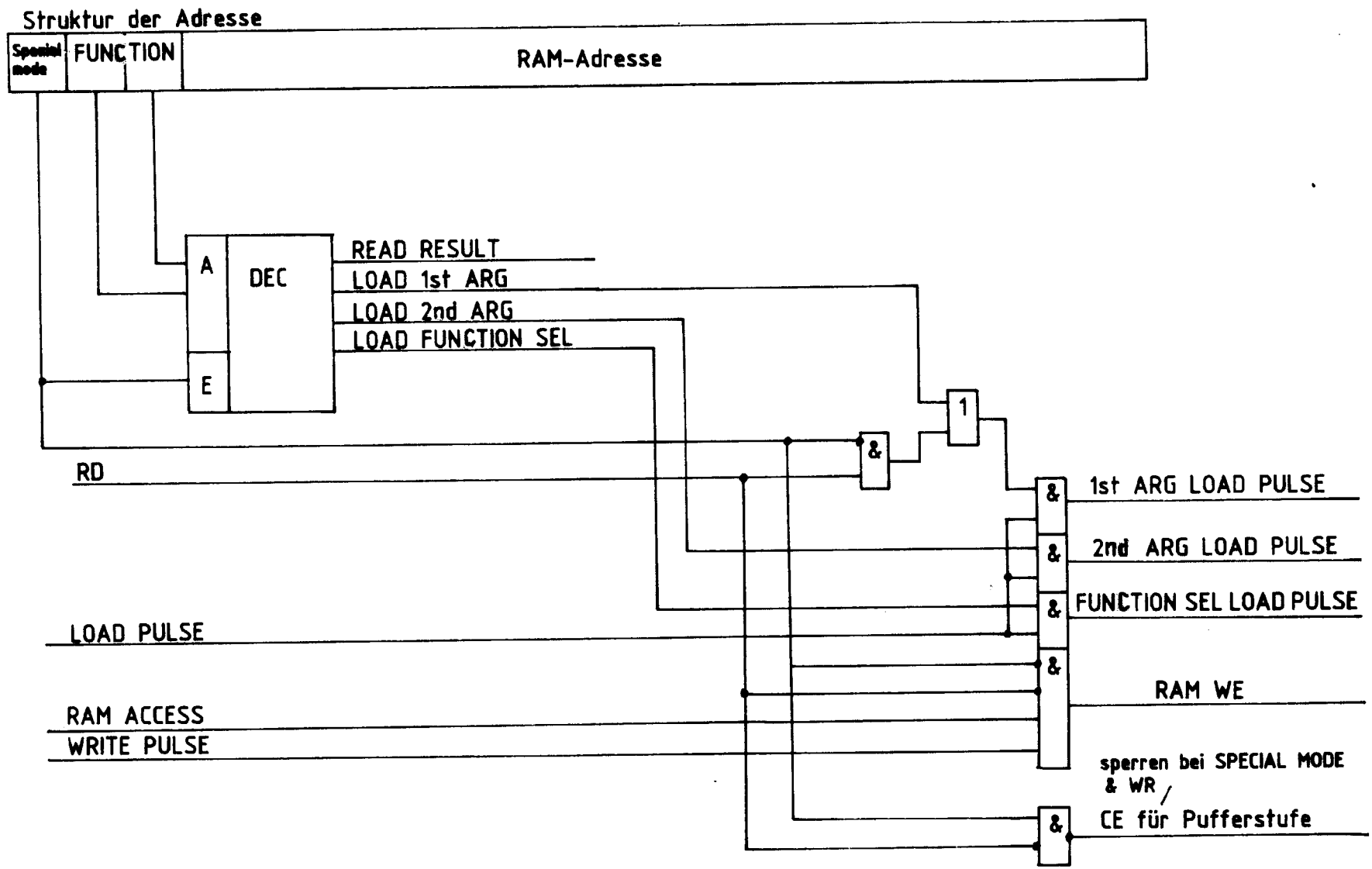
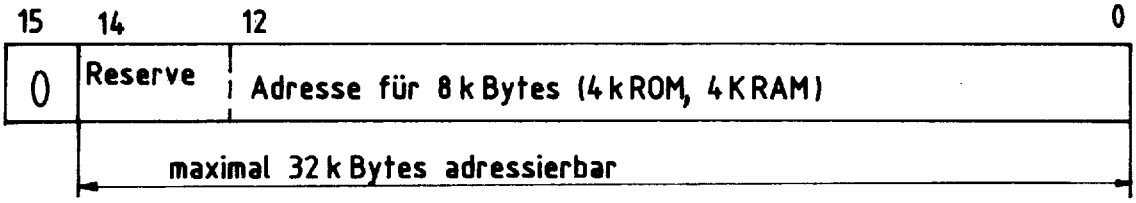
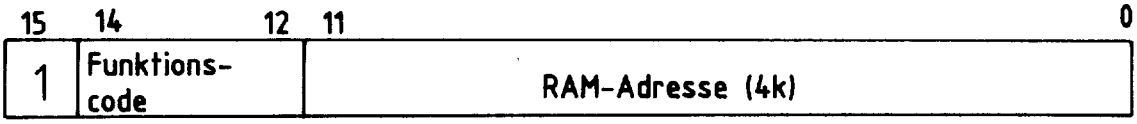

Struktur der Adresse
Special mode
FUNCTION
RAM-Adresse
A
DEC
E
READ RESULT
LOAD 1st ARG
LOAD 2nd ARG
LOAD FUNCTION SEL
1
&
RD
LOAD PULSE
RAM ACCESS
WRITE PULSE
1st ARG LOAD PULSE
2nd ARG LOAD PULSE
FUNCTION SEL LOAD PULSE
RAM WE
sperren bei SPECIAL MODE & WR /
CE für Pufferstufe

Bild 16

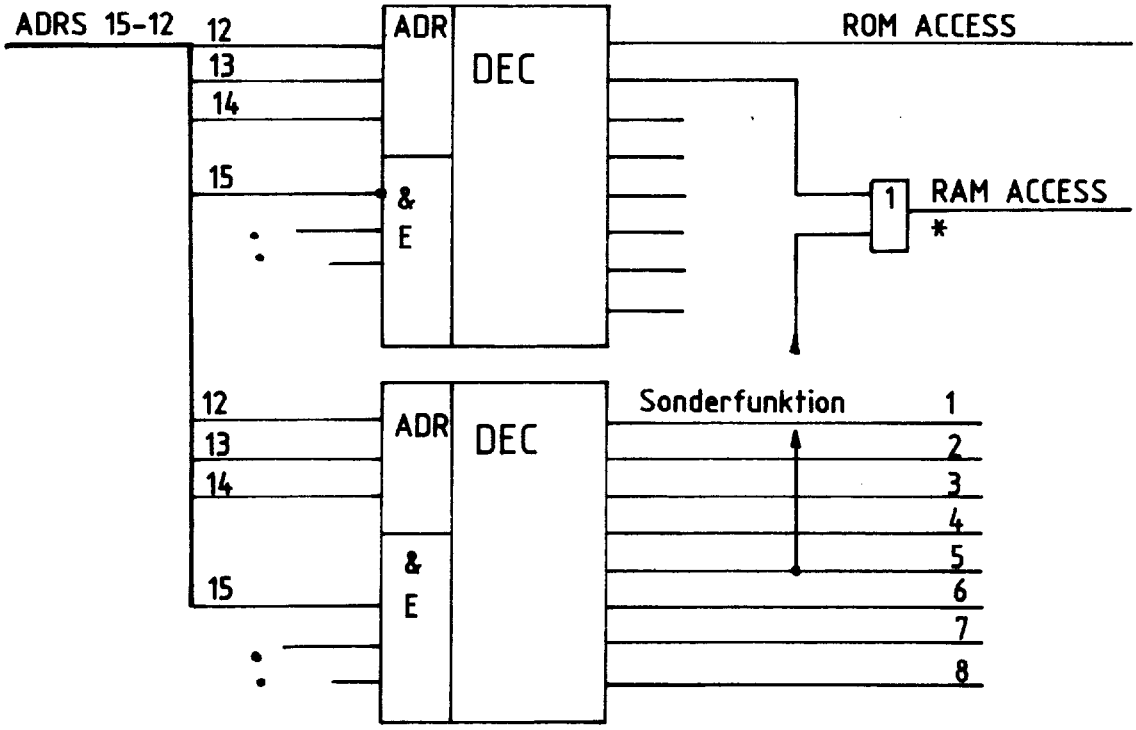
Adresse für Speicherzugriffe



Adresse für Sonderfunktionen



Beispiel für Adressendecodierung



*: Sonderfunktion 5 verlangt RAM-Zugriff

Bild 17

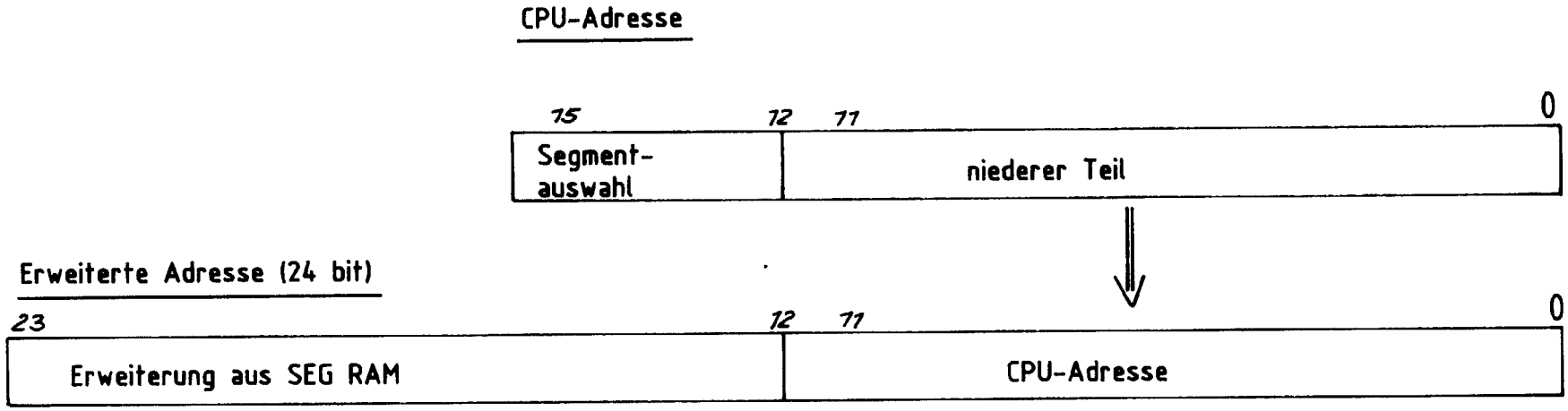

CPU-Adresse
15
12
11
0
Segment-
auswahl
niederer Teil
Erweiterte Adresse (24 bit)
23
12
11
0
Erweiterung aus SEG RAM
CPU-Adresse

Bild 18

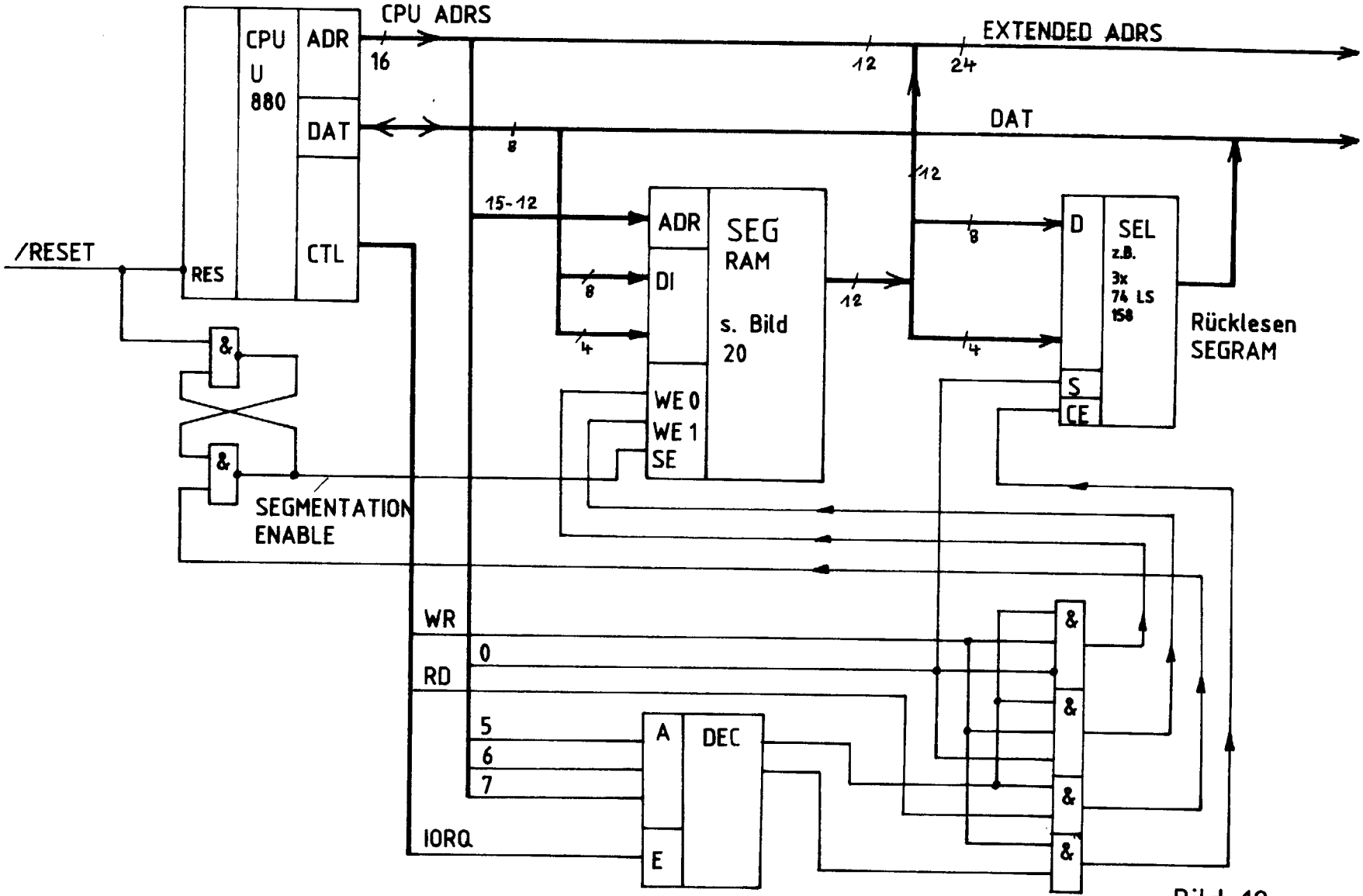

CPU ADRS
EXTENDED ADRS
CPU
U
880
ADR
DAT
CTL
RES
16
12
24
DAT
8
12
15-12
/RESET
ADR
SEG
RAM
DI
s. Bild
20
WE 0
WE 1
SE
8
4
12
D
SEL
z.B.
3x
74 LS
158
S
CE
Rücklesen
SEGRAM
SEGMENTATION
ENABLE
WR
RD
IORQ
0
5
6
7
A
DEC
E

Bild 19

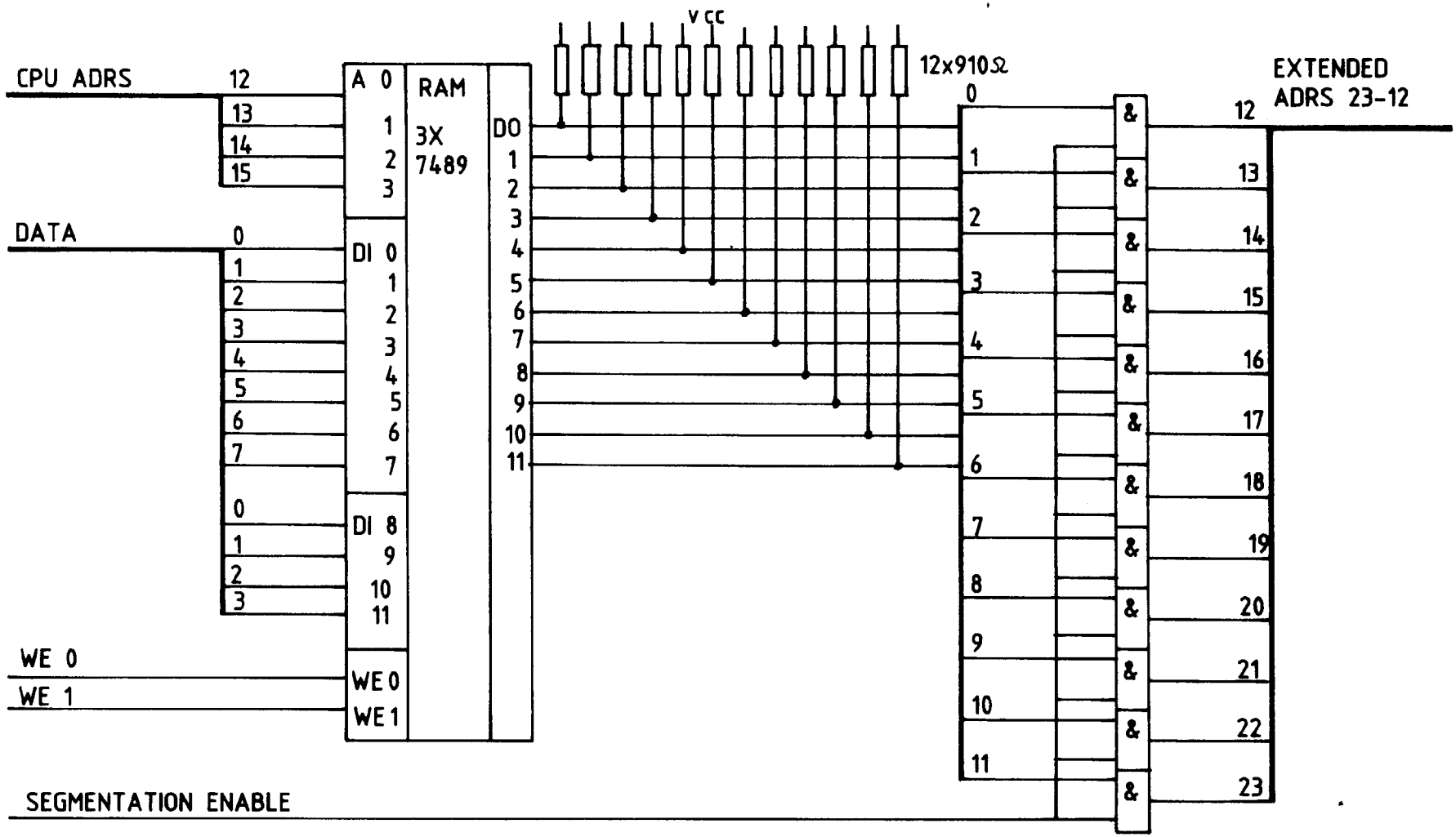

CPU ADRS
12
13
14
15
DATA
0
1
2
3
4
5
6
7
0
1
2
3
WE 0
WE 1
SEGMENTATION ENABLE
A 0
1
2
3
DI 0
1
2
3
4
5
6
7
DI 8
9
10
11
WE 0
WE 1
RAM
3X
7489
DO
1
2
3
4
5
6
7
8
9
10
11
VCC
12x910Ω
0
1
2
3
4
5
6
7
8
9
10
11
&
12
13
14
15
16
17
18
19
20
21
22
23
EXTENDED
ADRS 23-12

Bild 20

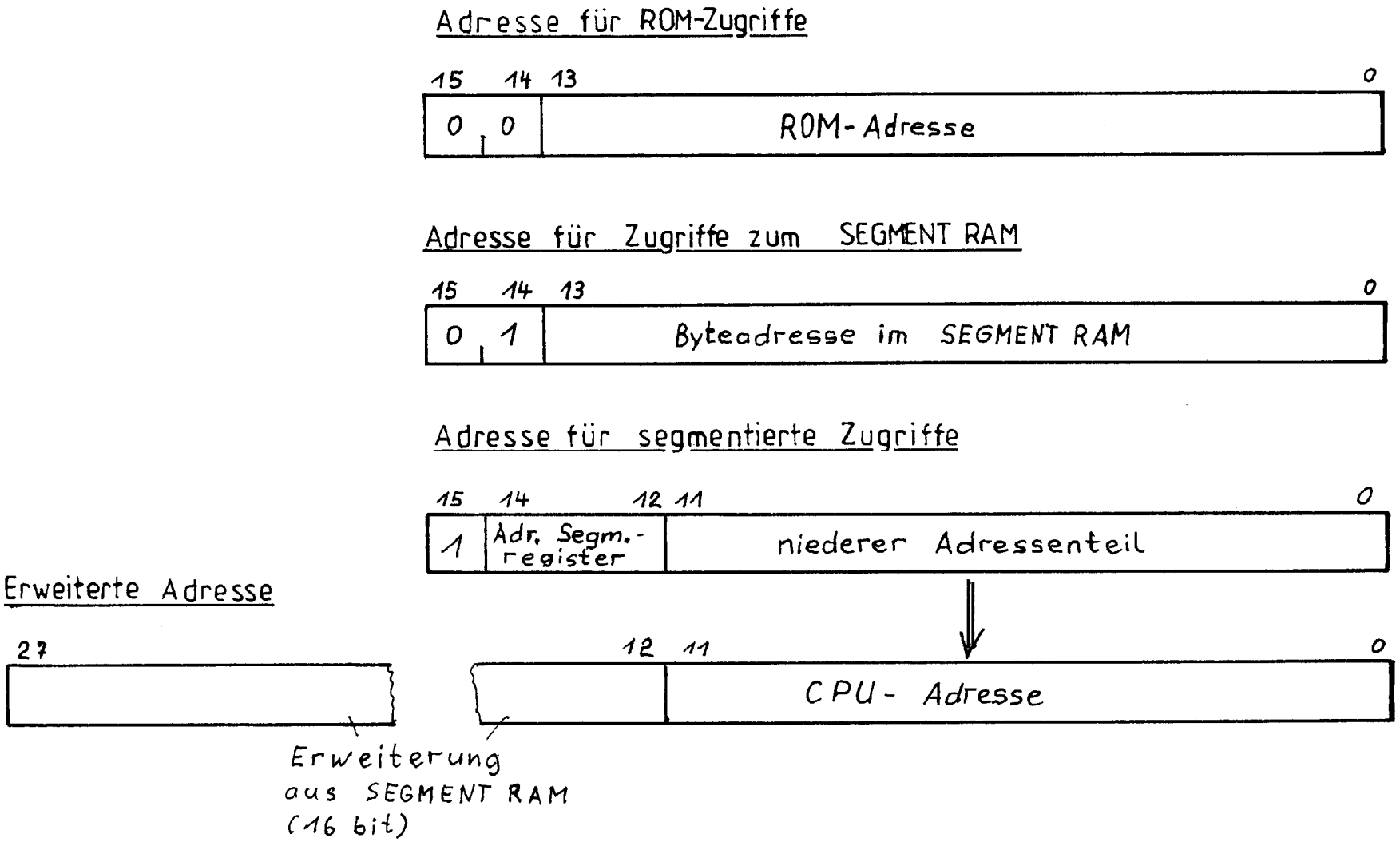

Adresse für ROM-Zugriffe
15 14 13 0
0 0
ROM-Adresse
Adresse für Zugriffe zum SEGMENT RAM
15 14 13 0
0 1
Byteadresse im SEGMENT RAM
Adresse für segmentierte Zugriffe
15 14 12 11 0
1
Adr. Segm.-register
niederer Adressenteil
Erweiterte Adresse
27
12 11 0
CPU- Adresse
Erweiterung aus SEGMENT RAM (16 bit)

Bild 21

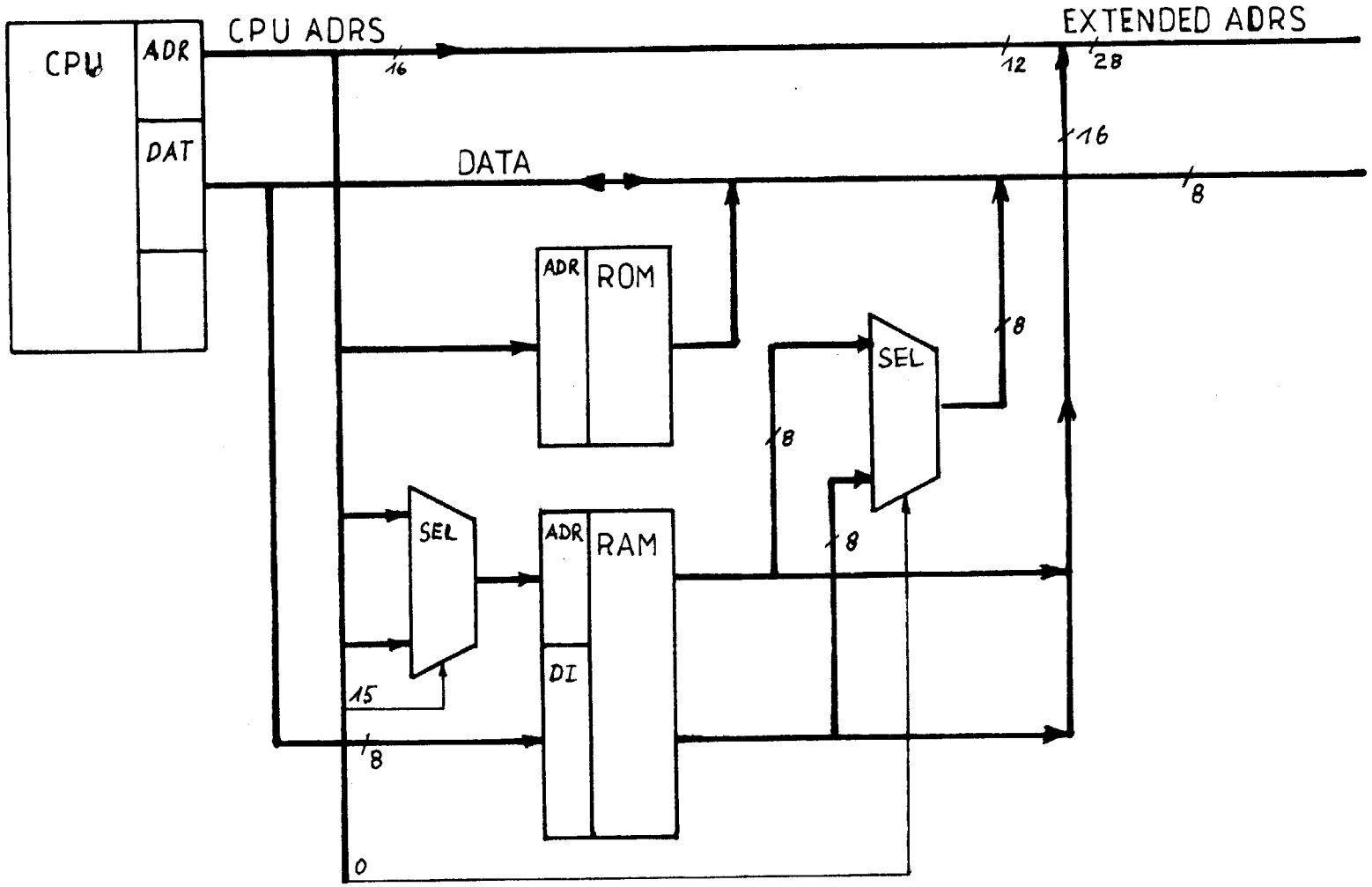

CPU
ADR
DAT
CPU ADRS
16
EXTENDED ADRS
12
28
16
DATA
8
ADR
ROM
SEL
8
8
SEL
ADR
RAM
DI
8
15
8
0

Bild 22

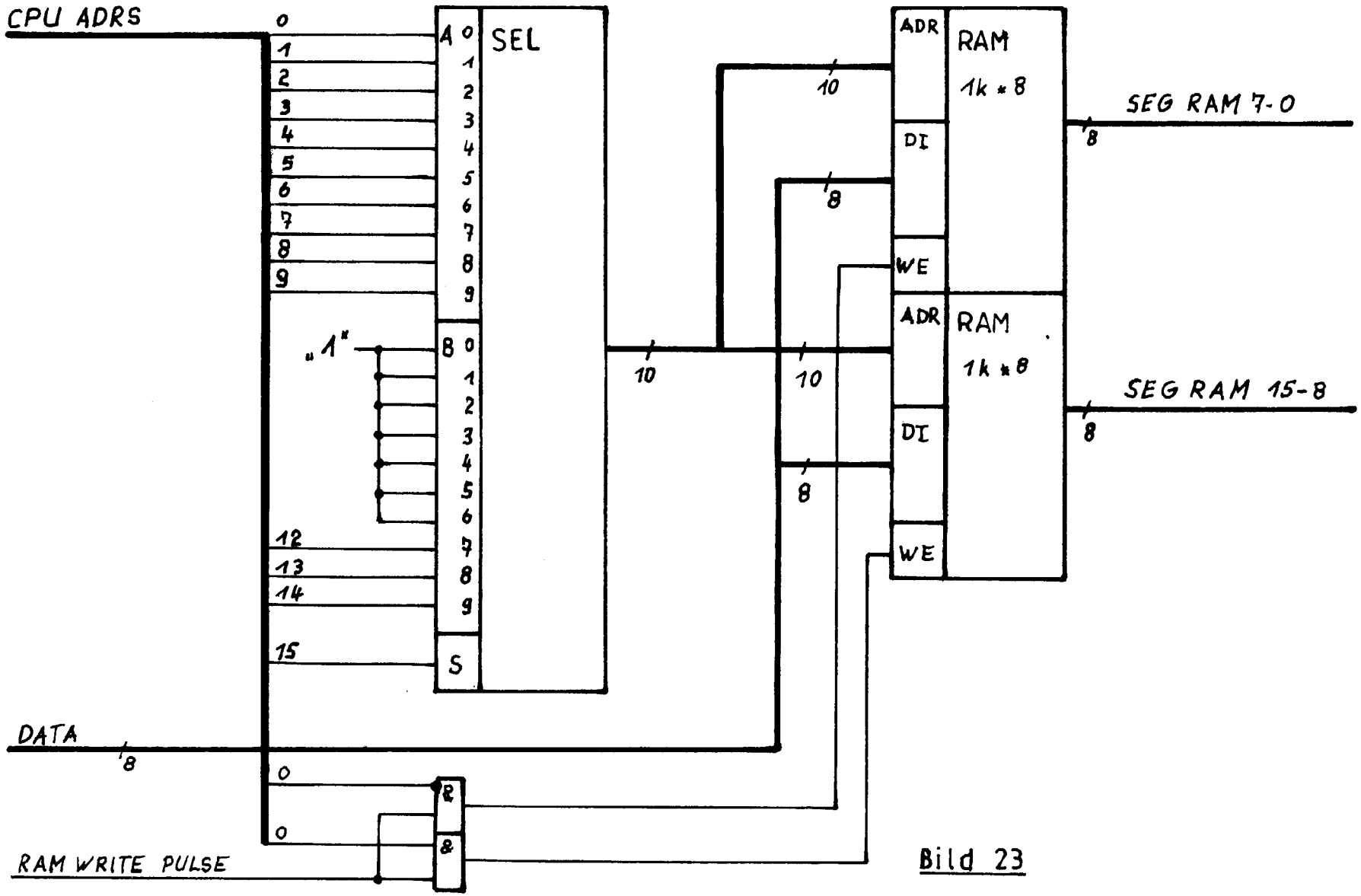

CPU ADRS
0
1
2
3
4
5
6
7
8
9
A
SEL
„1"
B
12
13
14
15
S
10
ADR
RAM
1k * 8
DI
WE
SEG RAM 7-0
8
SEG RAM 15-8
DATA
RAM WRITE PULSE
&

Bild 23

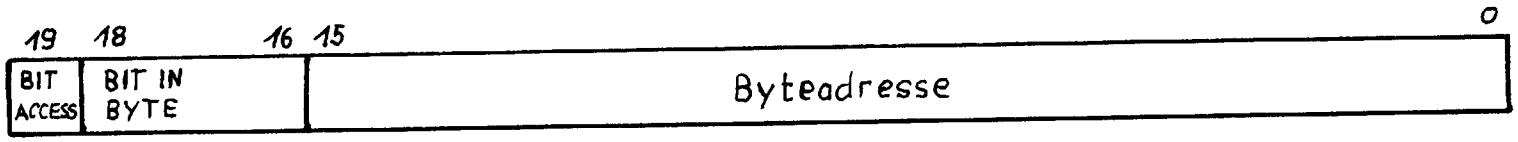


Bild 24

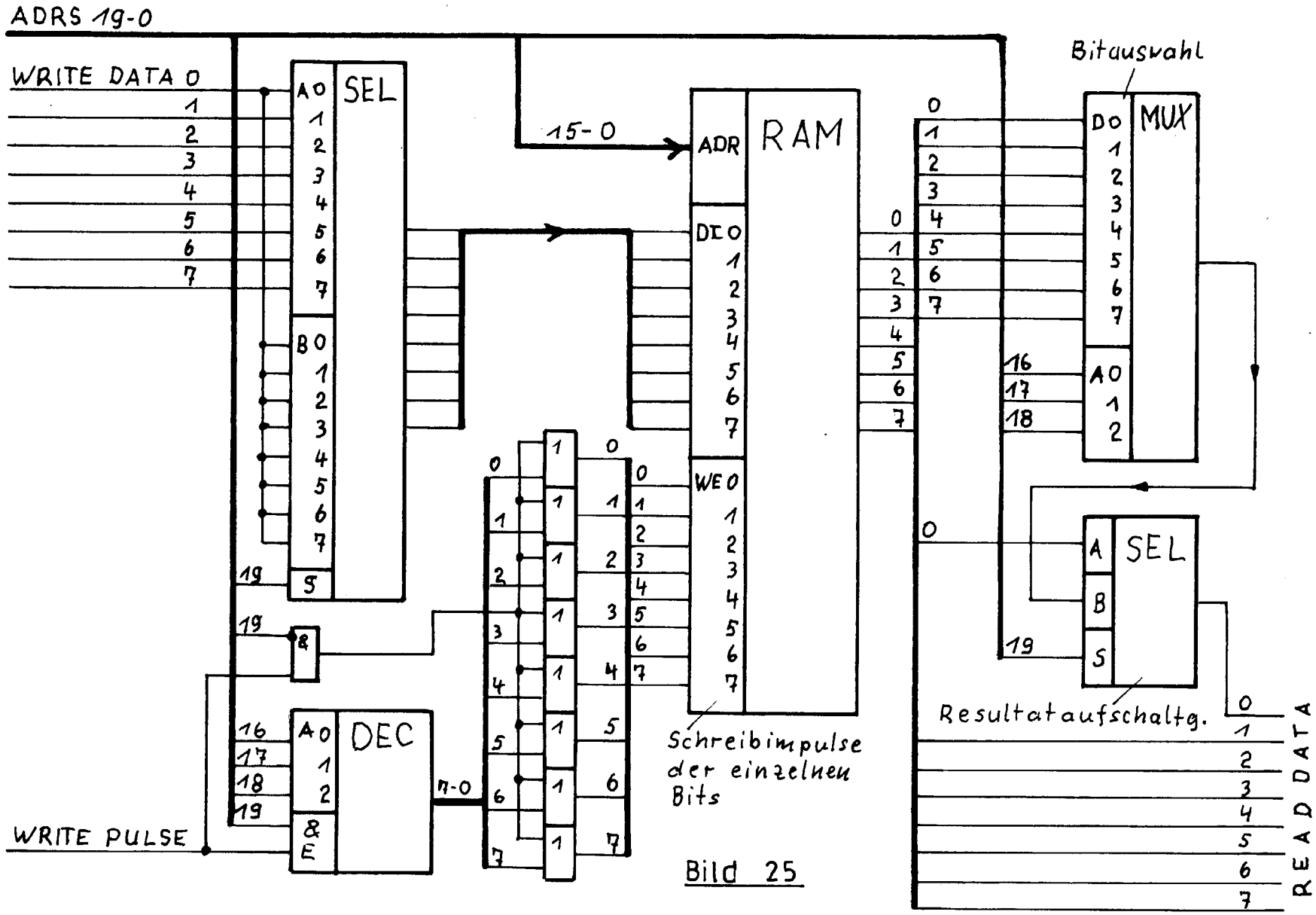
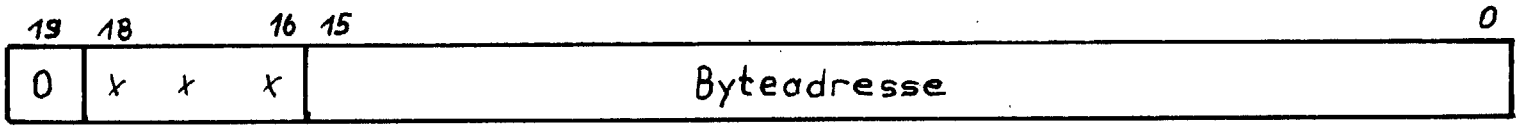

ADRS 19-0
WRITE DATA 0
1
2
3
4
5
6
7
A 0
1
2
3
4
5
6
7
B 0
1
2
3
4
5
6
7
19
S
SEL
15-0
ADR
RAM
DI 0
1
2
3
4
5
6
7
WE 0
1
2
3
4
5
6
7
Schreibimpulse der einzelnen Bits
Bitauswahl
D 0
1
2
3
4
5
6
7
A 0
1
2
MUX
16
17
18
19
A
B
S
SEL
Resultataufschaltg.
READ DATA
0
1
2
3
4
5
6
7
19
&
16
17
18
19
A 0
1
2
&
E
DEC
7-0
WRITE PULSE

Bild 25

a) Bytezugriff



b) Bitzugriff

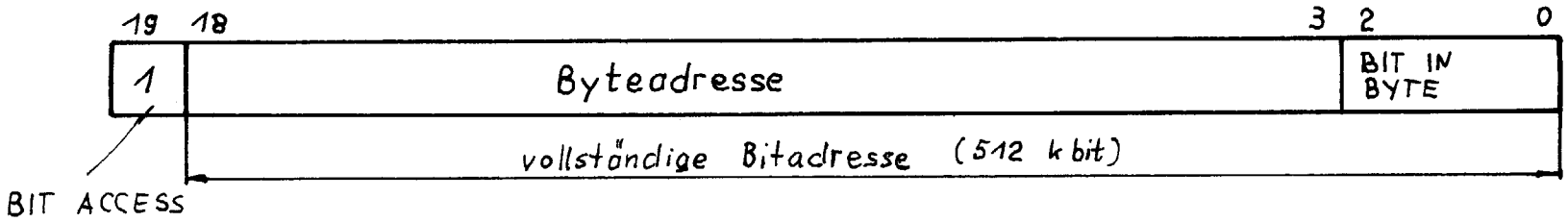


Bild 26

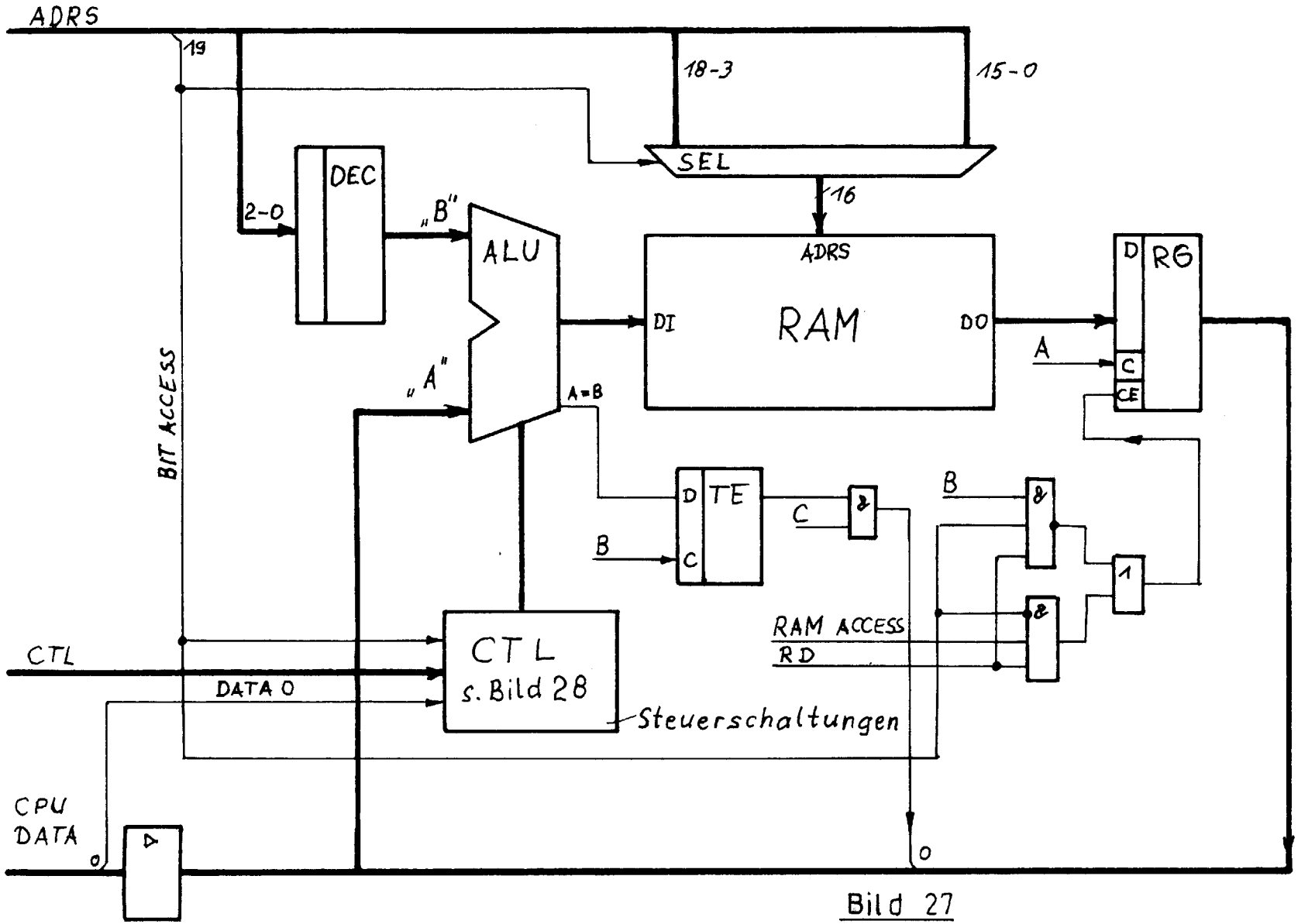

ADRS
19
18-3
15-0
SEL
16
DEC
2-0
„B"
ALU
„A"
A=B
ADRS
RAM
DI
DO
D
RG
A
C
CE
BIT ACCESS
D
TE
C
B
C
&
B
&
&
1
RAM ACCESS
RD
CTL
DATA 0
CTL
s. Bild 28
Steuerschaltungen
CPU
DATA
0
0

Bild 27

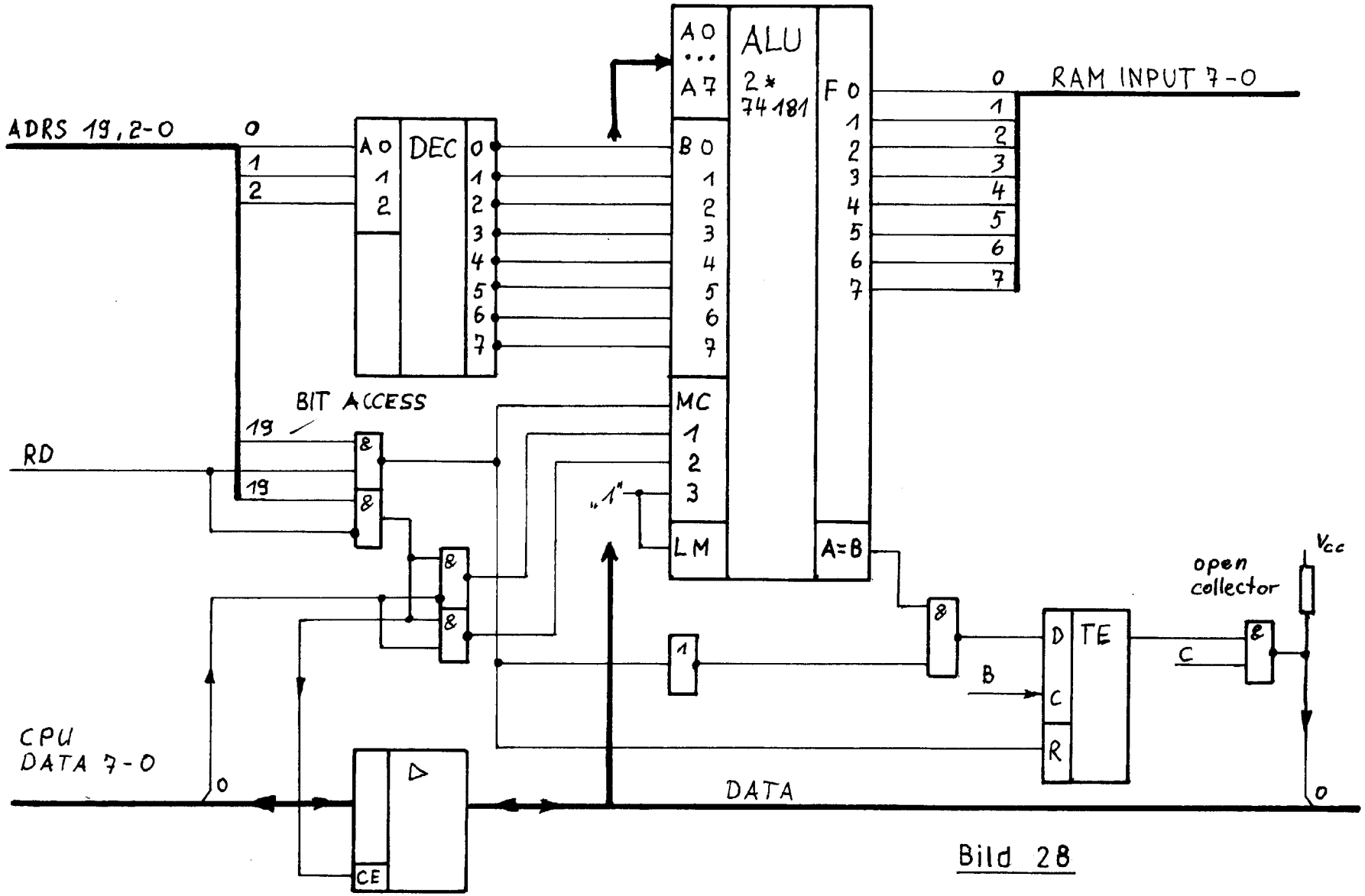

ADRS 19,2-0
RD
CPU
DATA 7-0
BIT ACCESS
DEC
A0
A7
B0
MC
LM
ALU
2*
74181
F0
A=B
RAM INPUT 7-0
"1"
DATA
CE
D
C
R
TE
B
C
open
collector
Vcc

Bild 28

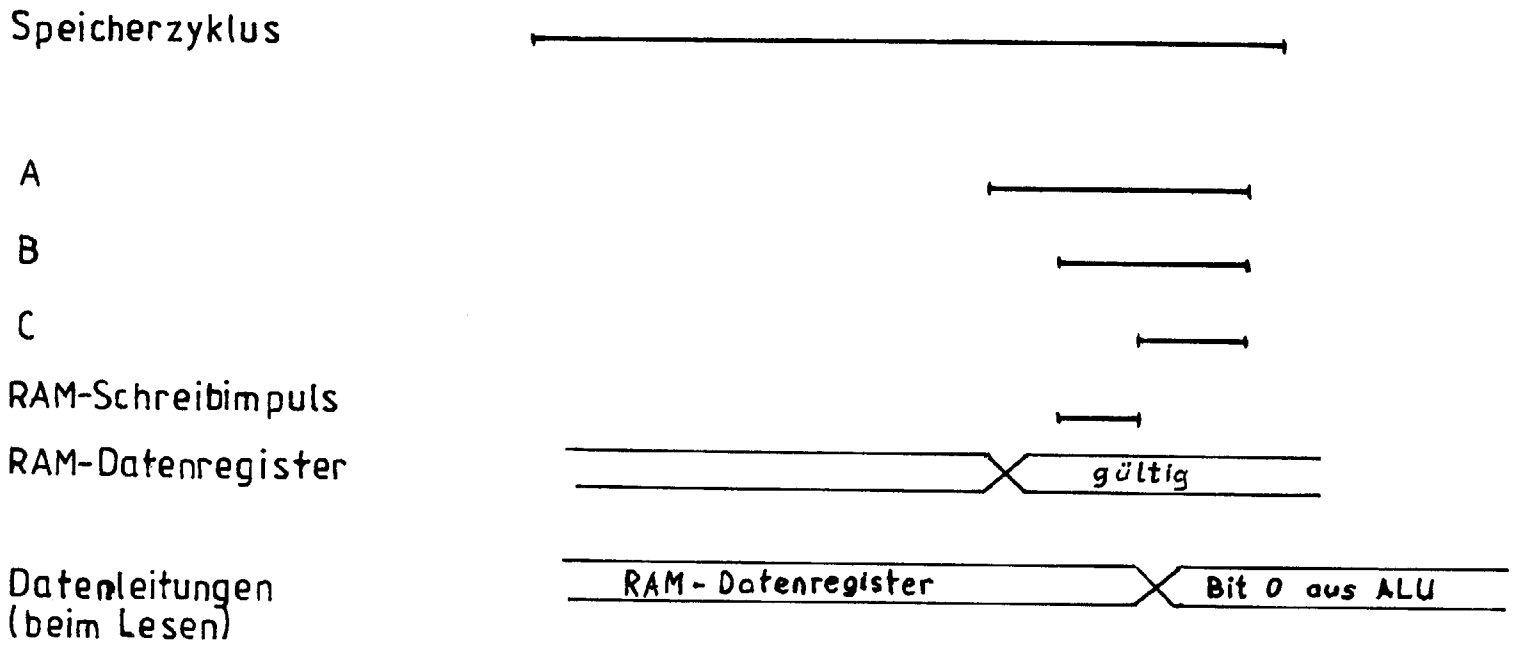

Speicherzyklus
A
B
C
RAM-Schreibimpuls
RAM-Datenregister
gültig
Datenleitungen
(beim Lesen)
RAM-Datenregister
Bit 0 aus ALU

Bild 29

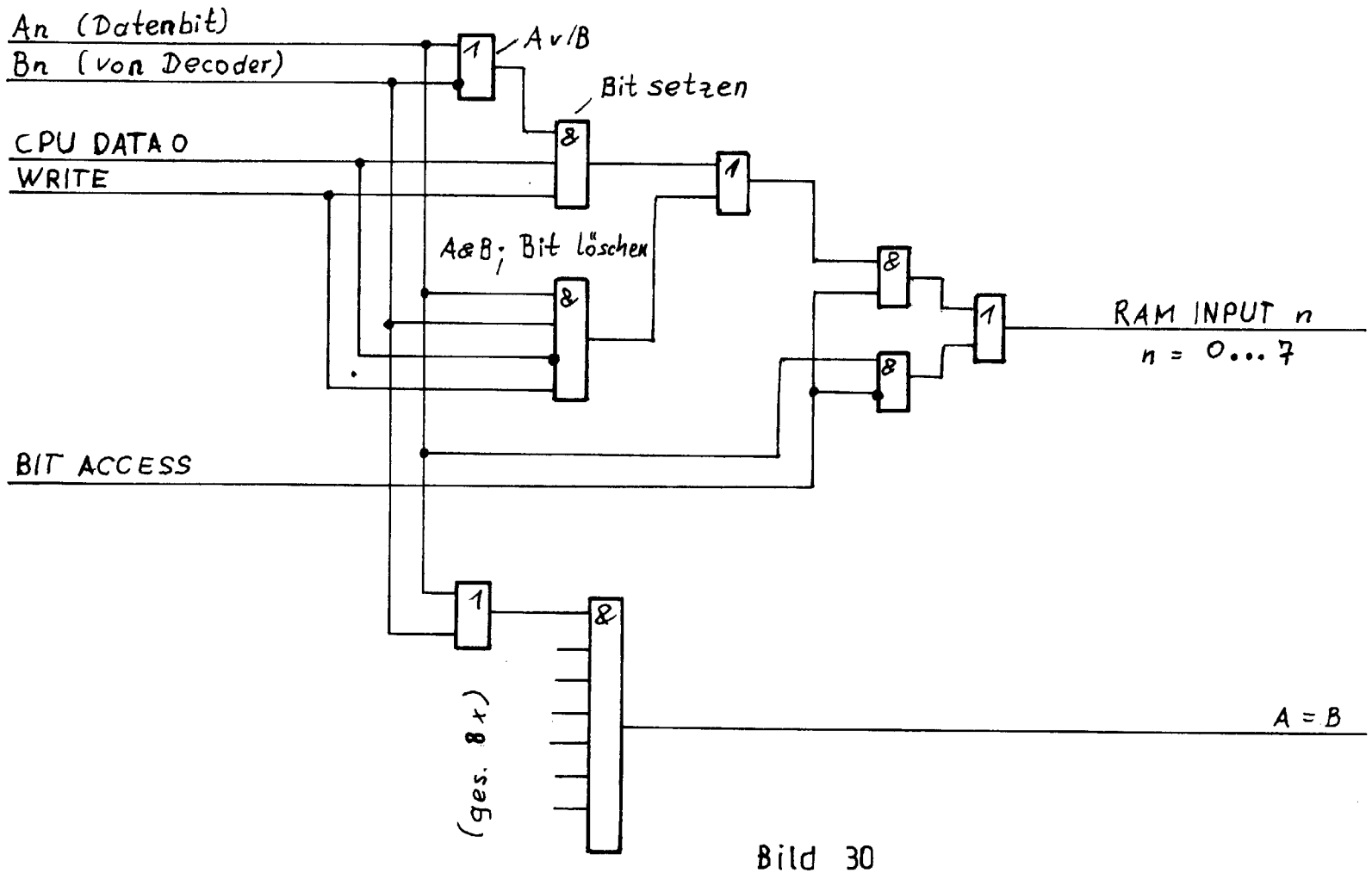

An (Datenbit)
Bn (von Decoder)
CPU DATA 0
WRITE
BIT ACCESS
A v/B
Bit setzen
A&B; Bit löschen
RAM INPUT n
n = 0...7
(ges. 8x)
A = B

Bild 30

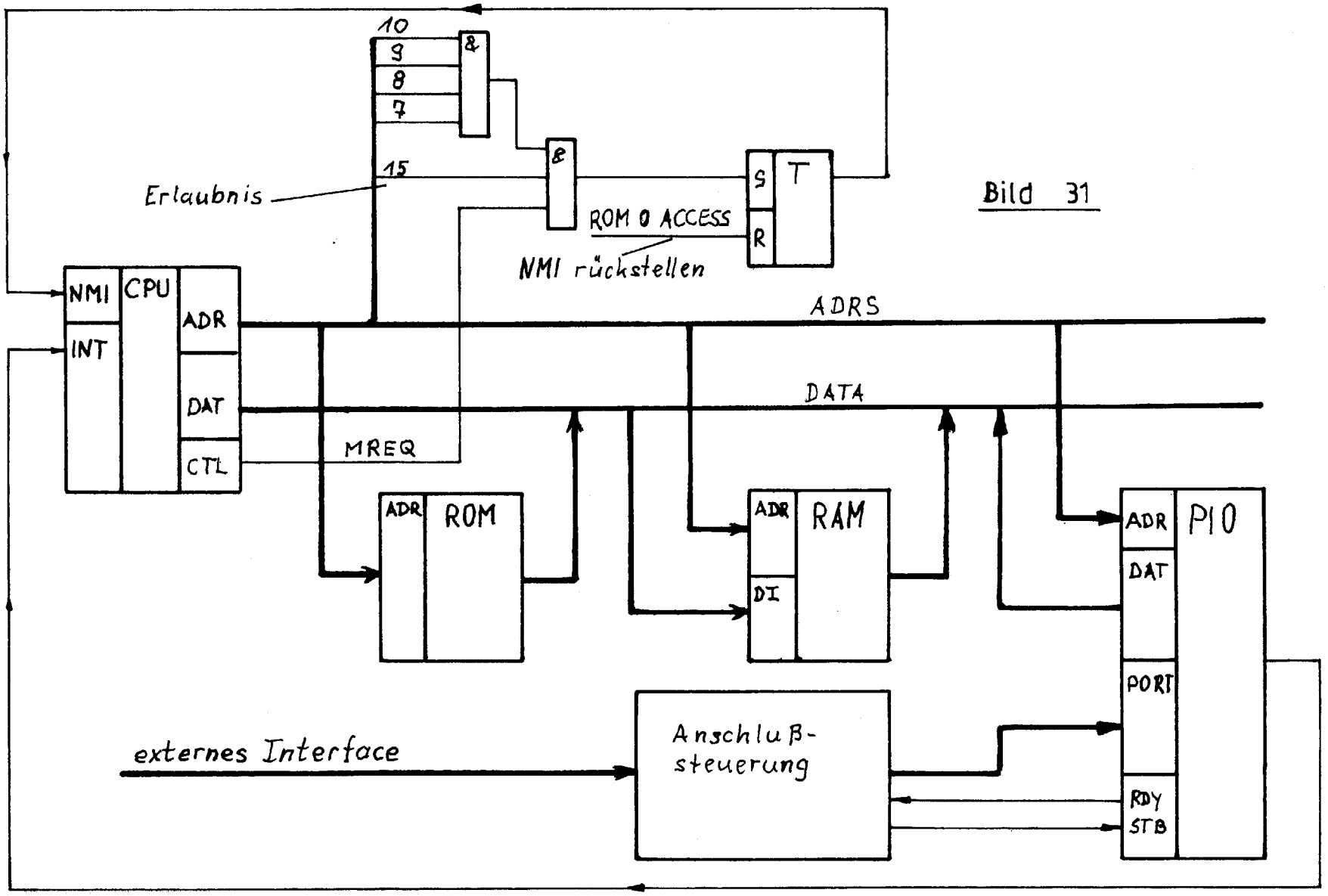

10
9
8
7
&
15
Erlaubnis
&
ROM 0 ACCESS
NMI rückstellen
S
R
T
NMI
INT
CPU
ADR
DAT
CTL
ADRS
DATA
MREQ
ADR
ROM
ADR
DI
RAM
ADR
DAT
PORT
RDY
STB
PIO
externes Interface
Anschluß-
steuerung

Bild 31

```
 4        EXAF                          } Registertausch
 4        EXX
10        IN                            ;Abholen Datenbyte
 7        LD (HL),A                     ;Speichern in RAM
 6        INC HL                        ;Adresse erhöhen
10        LD BC,-1920                   }
11        ADD HL,BC                     } Test auf Bereichs-
 4        LD A,L                        } überschreitung
 4        OR H
12        JRNZ                          ;Übersprung, wenn keine
          spezifische Reaktion           Bereichsüberschreitung
10        LD BC,1920                    ;Kompensation für die
11        ADD HL,BC                      Testoperation (alter Inhalt
 4        EXX                            nach HL)
 4        EXAF                          }
 4        EI                            } Registertausch und Rückkehr
14        RETI                          }
119 Zyklen
```

Bild 32

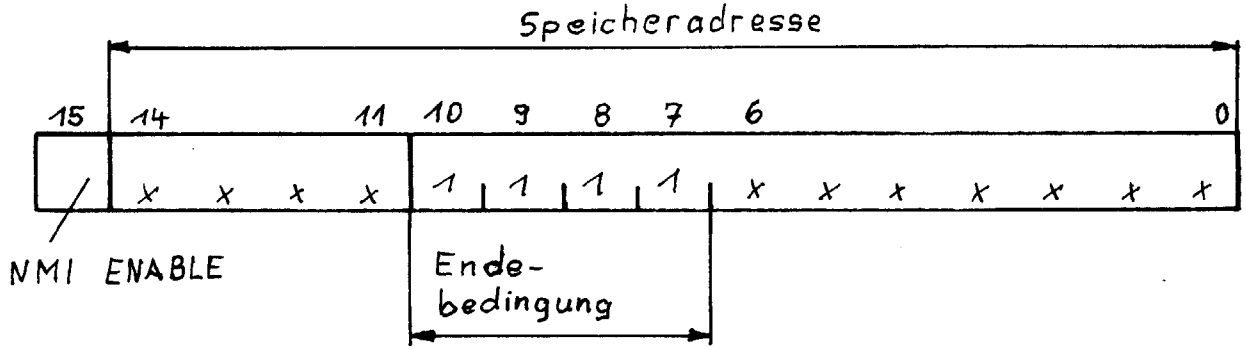

Speicheradresse
15
14
11
10
9
8
7
6
0
x x x x 1 1 1 1 x x x x x x x
NMI ENABLE
Ende-
bedingung

Bild 33

```
 4          EXAF
 4          EXX
10          IN              ;Abholen Datenbyte
 7          LD (HL),A       ;Speichern in RAM
 6          INC HL          ;Adresse erhöhen
 4          EXX
 4          EXAF
 4          EI
14          RETI
57 Zyklen
```

Bild 34

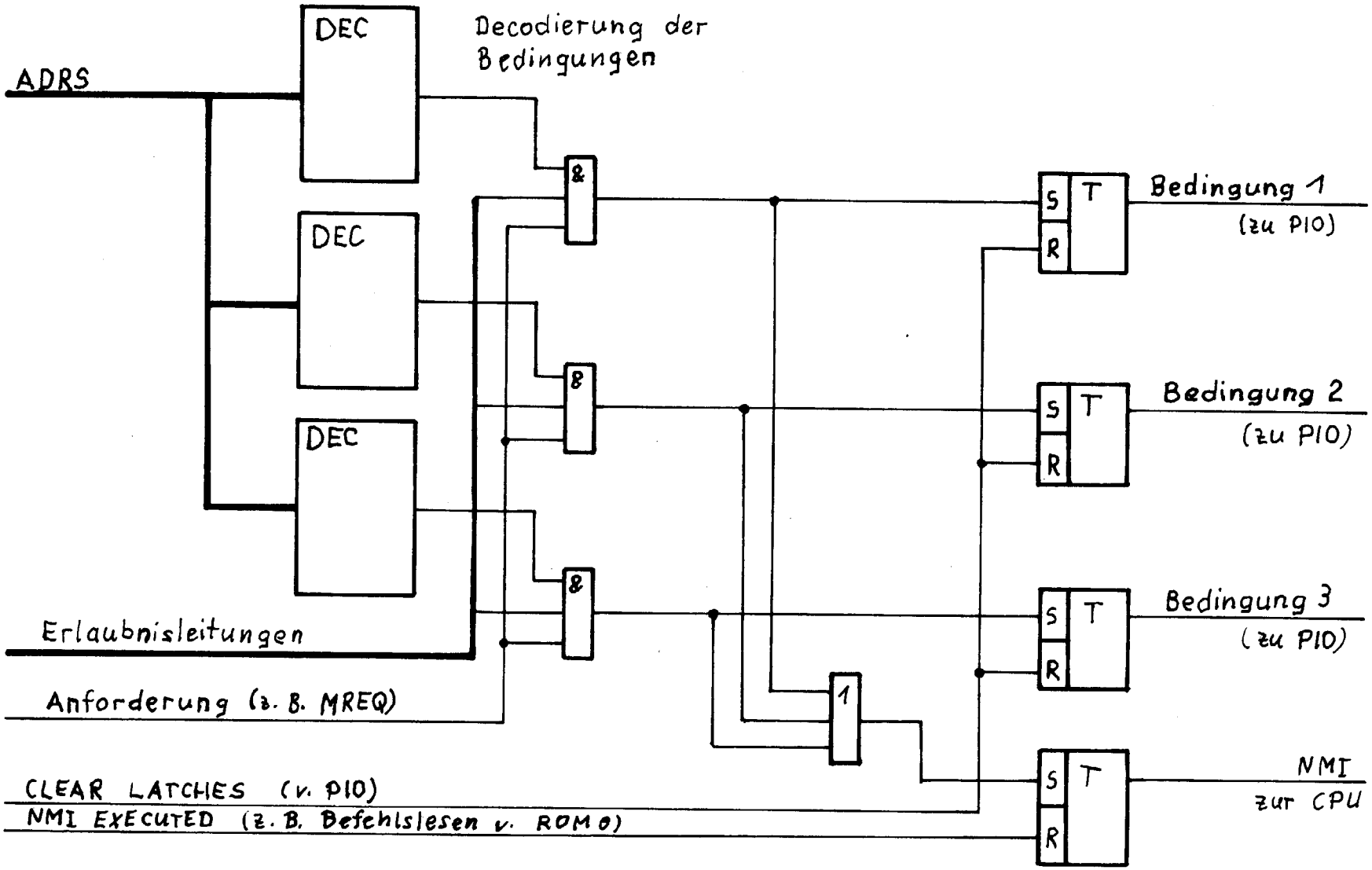

ADRS
DEC
DEC
DEC
Decodierung der Bedingungen
&
&
&
Erlaubnisleitungen
Anforderung (z. B. MREQ)
1
S
R
T
Bedingung 1
(zu PIO)
Bedingung 2
(zu PIO)
Bedingung 3
(zu PIO)
NMI
zur CPU
CLEAR LATCHES (v. PIO)
NMI EXECUTED (z. B. Befehlslesen v. ROM 0)

Bild 35

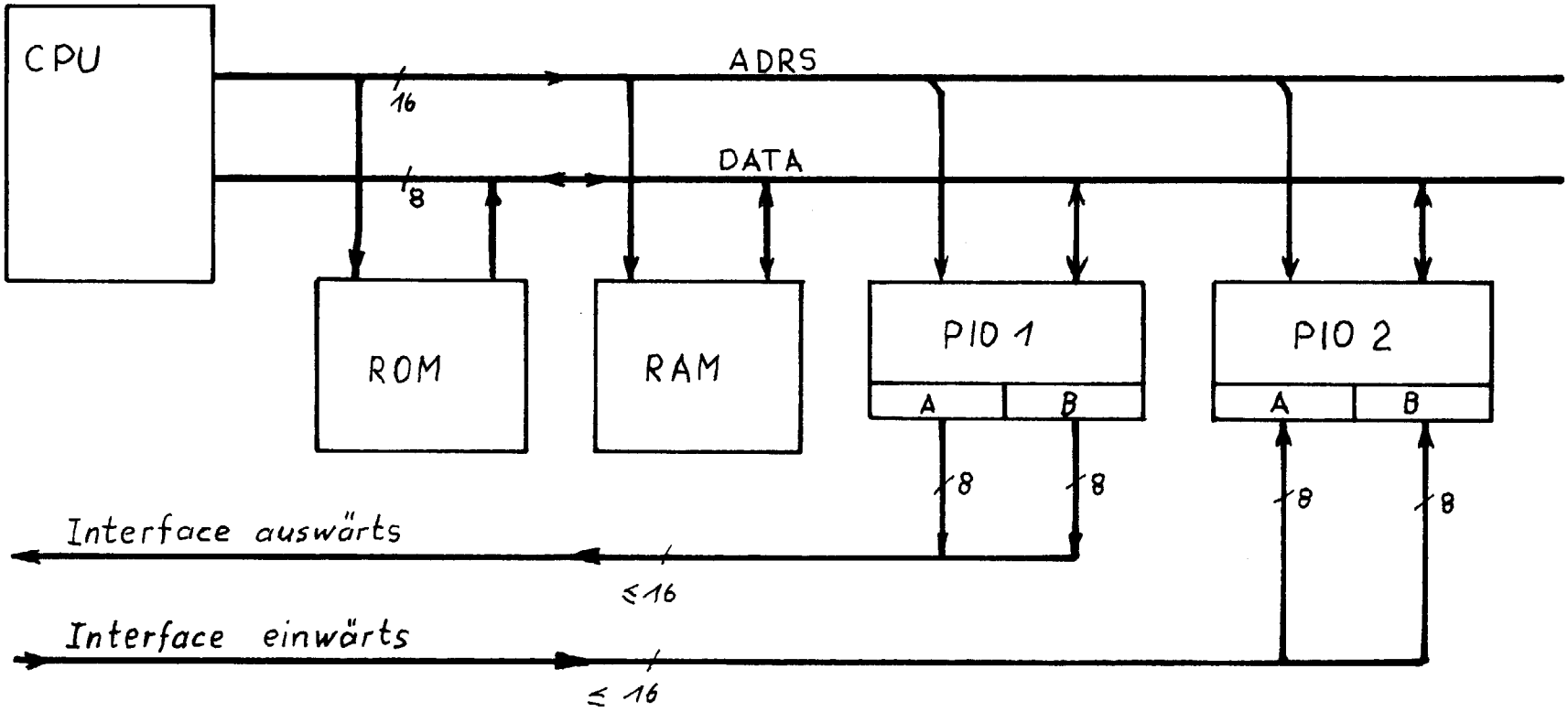

CPU
ADRS
16
DATA
8
ROM
RAM
PIO 1
A
B
PIO 2
A
B
8
8
8
8
Interface auswärts
≤16
Interface einwärts
≤ 16

Bild 36

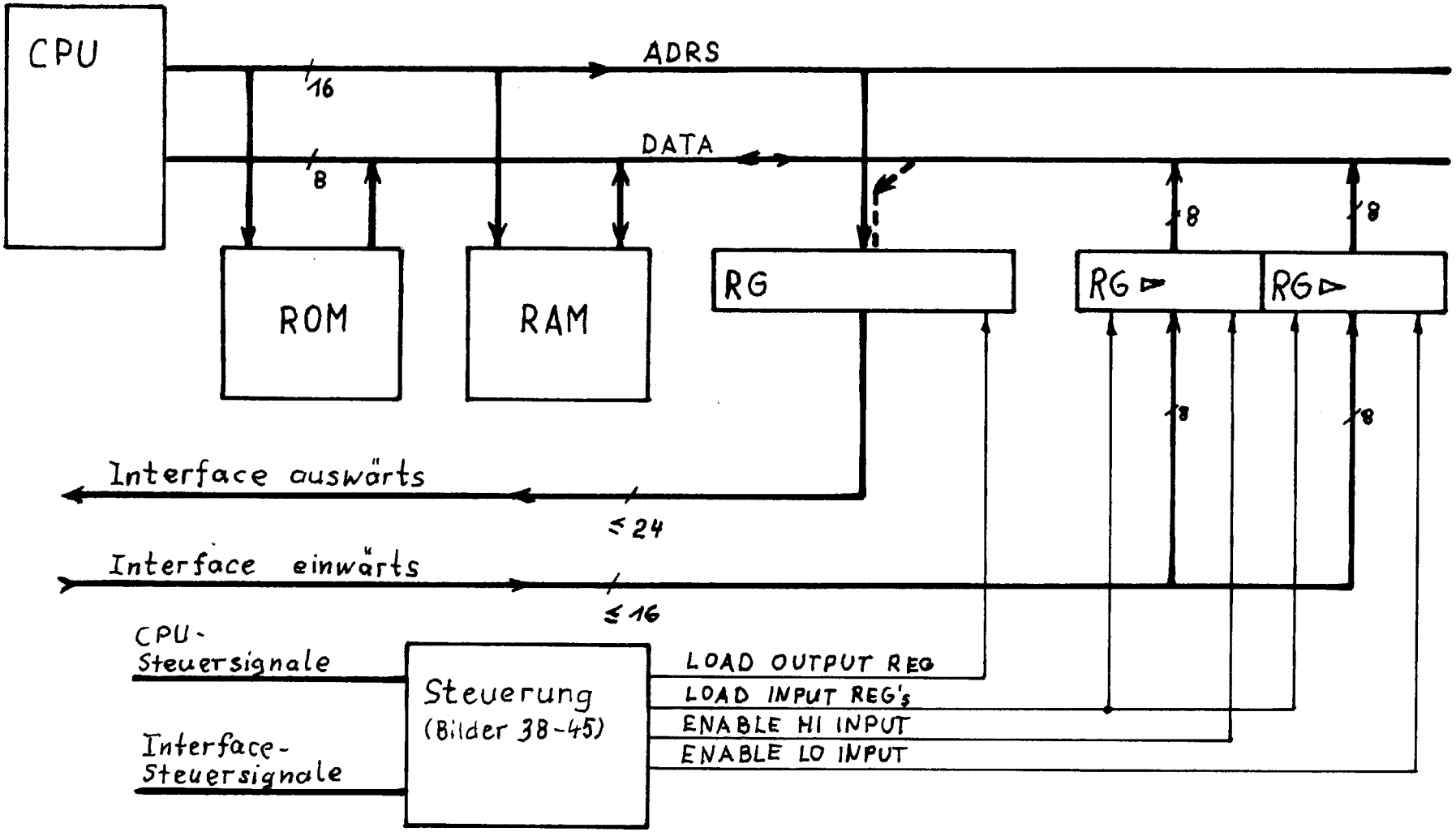

CPU
ADRS
16
DATA
8
ROM
RAM
RG
RG
RG
8
8
8
8
Interface auswärts
≤ 24
Interface einwärts
≤ 16
CPU-
Steuersignale
Interface-
Steuersignale
Steuerung
(Bilder 38-45)
LOAD OUTPUT REG
LOAD INPUT REG's
ENABLE HI INPUT
ENABLE LO INPUT

Bild 37

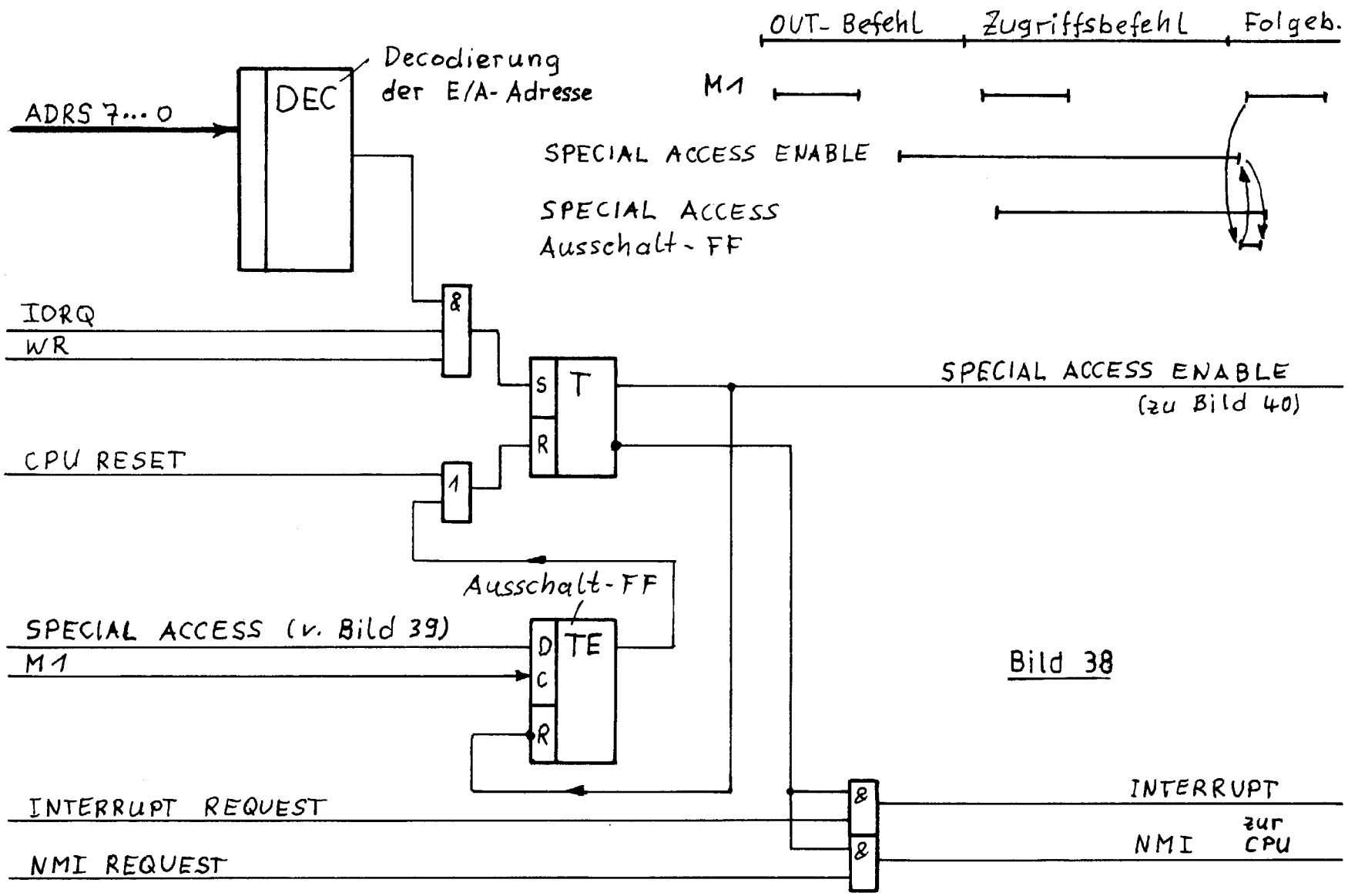
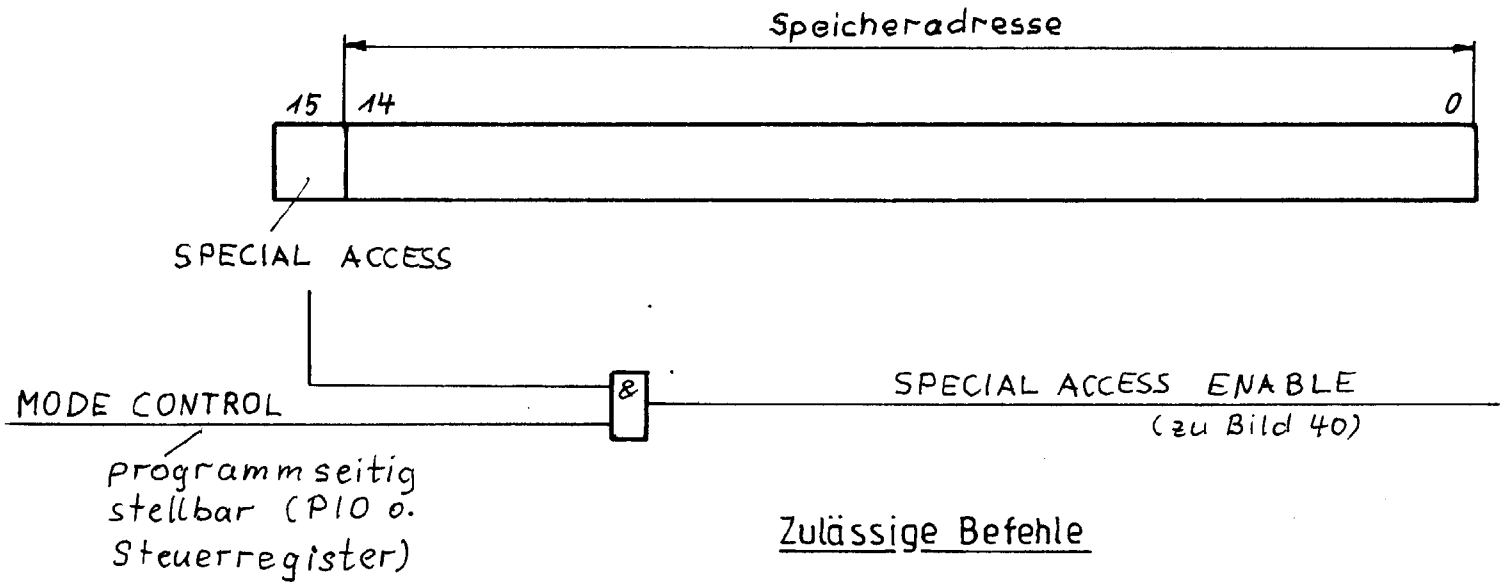

OUT- Befehl
Zugriffsbefehl
Folgeb.
M1
SPECIAL ACCESS ENABLE
SPECIAL ACCESS
Ausschalt-FF
ADRS 7...0
DEC
Decodierung
der E/A-Adresse
IORQ
WR
&
S
T
R
SPECIAL ACCESS ENABLE
(zu Bild 40)
CPU RESET
1
Ausschalt-FF
SPECIAL ACCESS (v. Bild 39)
M1
D
TE
C
R
INTERRUPT REQUEST
NMI REQUEST
&
&
INTERRUPT
NMI
zur CPU

Bild 38

Adressenstruktur



Zulässige Befehle

Ausgabe	Eingabe
LD (BC), A	LD BC, mn
LD (DE), A	LD DE, mn
LD (HL), A	LD HL, mn

Bild 39

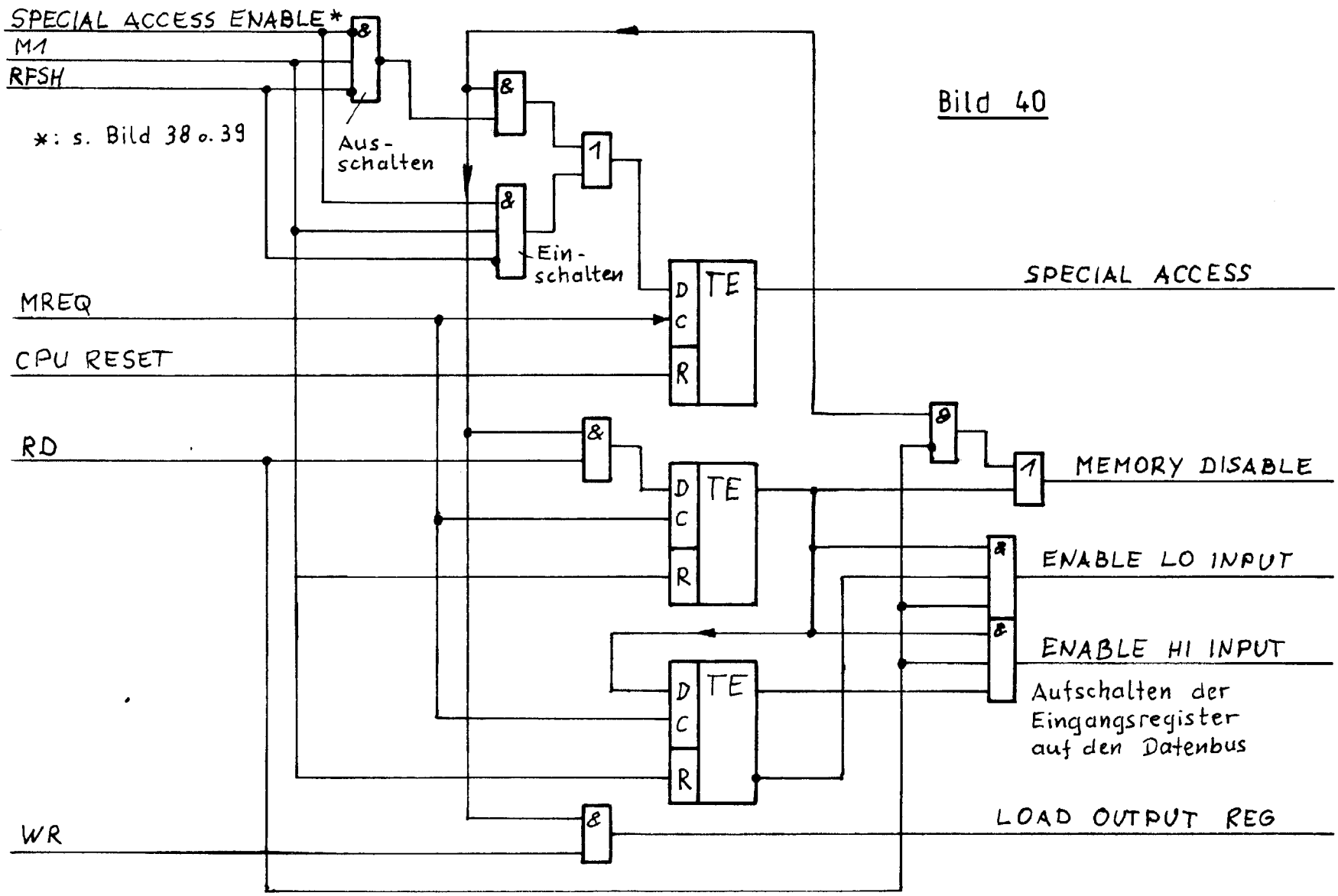

SPECIAL ACCESS ENABLE*
M1
RFSH
*: s. Bild 38 o. 39
Aus-
schalten
Ein-
schalten
MREQ
CPU RESET
RD
WR
D
C
R
TE
SPECIAL ACCESS
MEMORY DISABLE
ENABLE LO INPUT
ENABLE HI INPUT
Aufschalten der
Eingangsregister
auf den Datenbus
LOAD OUTPUT REG

Bild 40

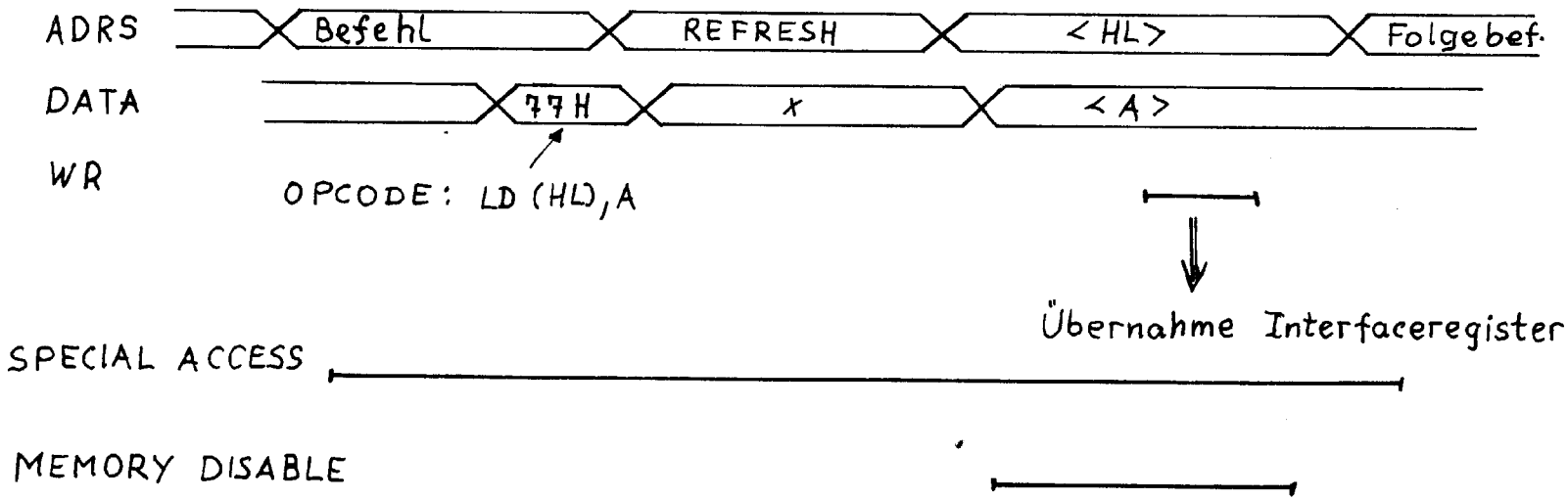

ADRS
Befehl
REFRESH
<HL>
Folgebef.
DATA
77H
x
<A>
WR
OPCODE: LD (HL),A
Übernahme Interfaceregister
SPECIAL ACCESS
MEMORY DISABLE

Bild 41

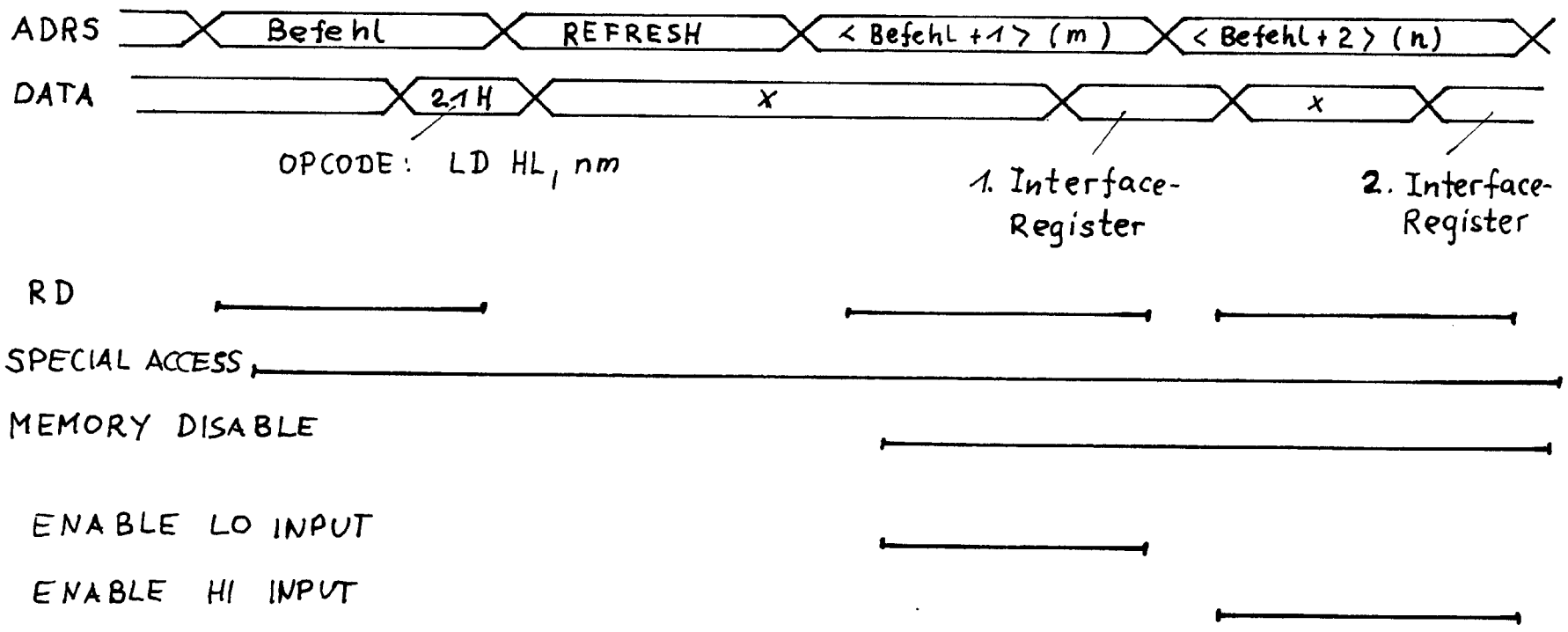

ADRS
Befehl
REFRESH
< Befehl +1 > (m)
< Befehl + 2 > (n)
DATA
21 H
x
x
OPCODE: LD HL, nm
1. Interface-Register
2. Interface-Register
RD
SPECIAL ACCESS
MEMORY DISABLE
ENABLE LO INPUT
ENABLE HI INPUT

Bild 42

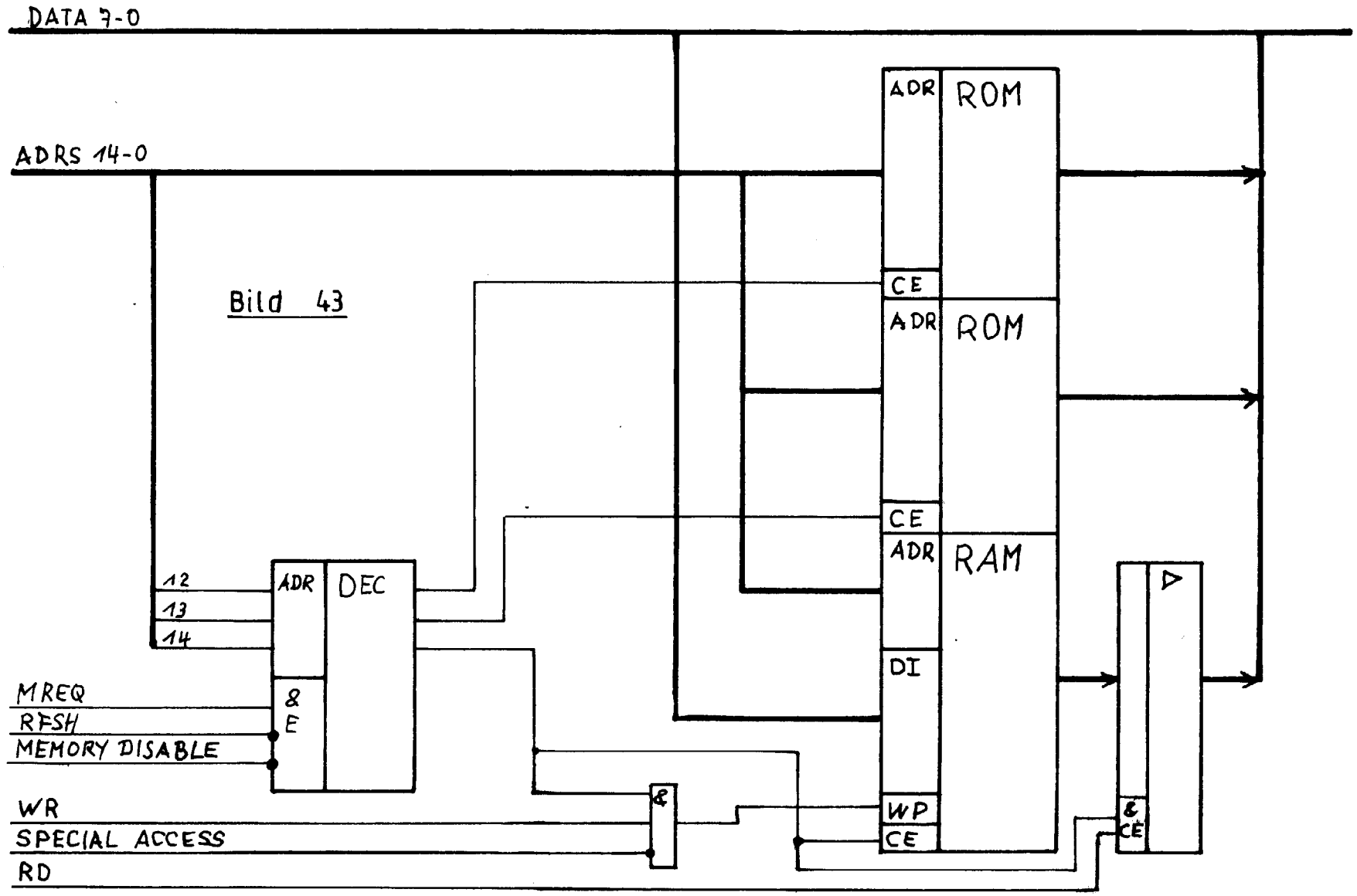

DATA 7-0
ADRS 14-0
ADR
ROM
CE
ADR
ROM
CE
ADR
RAM
DI
WP
CE
12
13
14
ADR
DEC
&
E
MREQ
RFSH
MEMORY DISABLE
WR
SPECIAL ACCESS
RD
&
&
CE

Bild 43

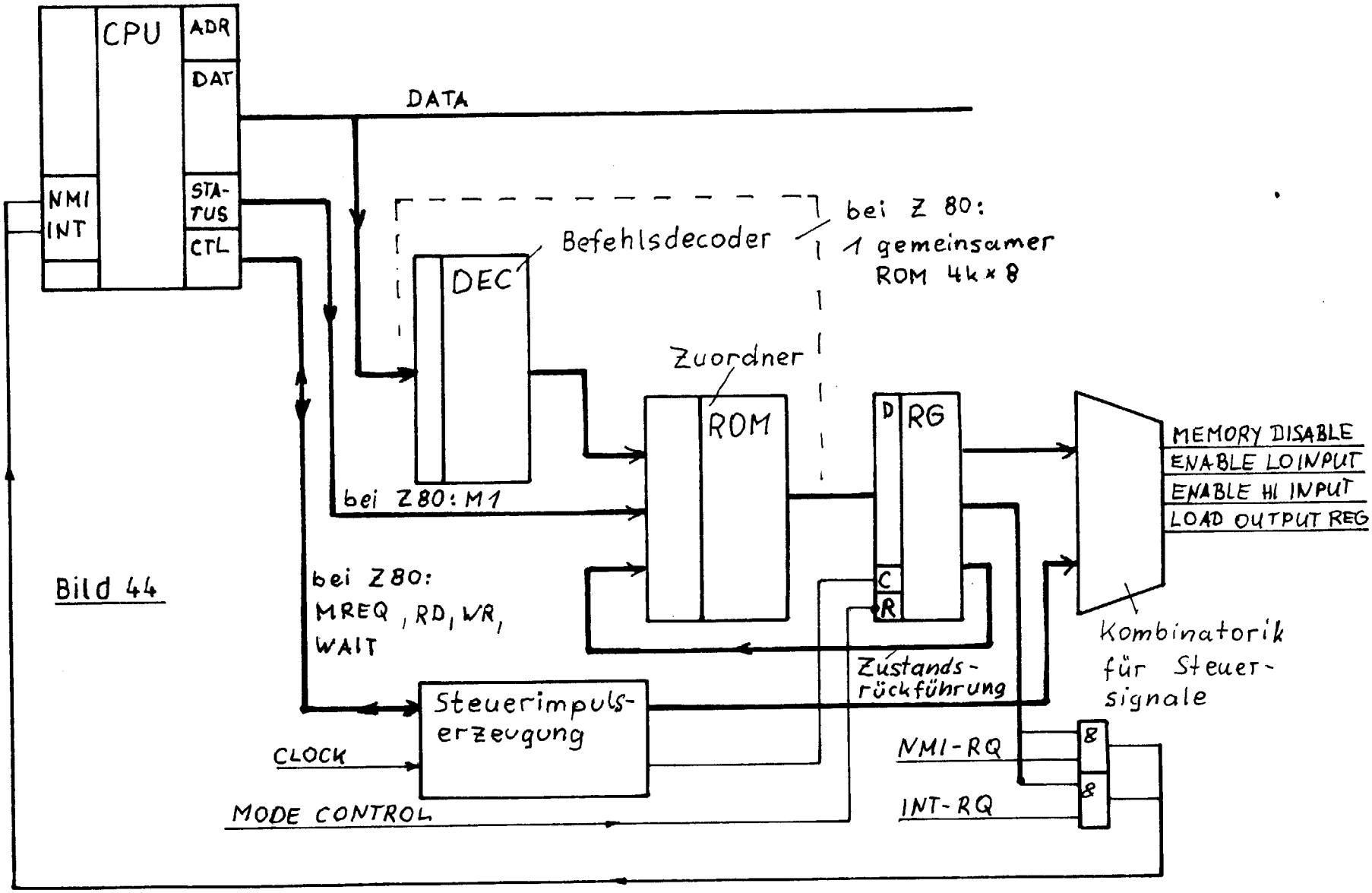

CPU
ADR
DAT
STA-
TUS
CTL
NMI
INT
DATA
Befehlsdecoder
DEC
bei Z 80:
1 gemeinsamer
ROM 4k × 8
Zuordner
ROM
D
RG
C
R
MEMORY DISABLE
ENABLE LO INPUT
ENABLE HI INPUT
LOAD OUTPUT REG
bei Z80: M1
bei Z80:
MREQ, RD, WR,
WAIT
Kombinatorik
für Steuer-
signale
Zustands-
rückführung
Steuerimpuls-
erzeugung
CLOCK
MODE CONTROL
NMI-RQ
INT-RQ
8
8

Bild 44

Erläuterungen

ZB: Zwei-Byte-Opcode (CDH, DDH, EDH, FDH)

SW: Opcode, der für Sonderwirkungen vorgesehen ist

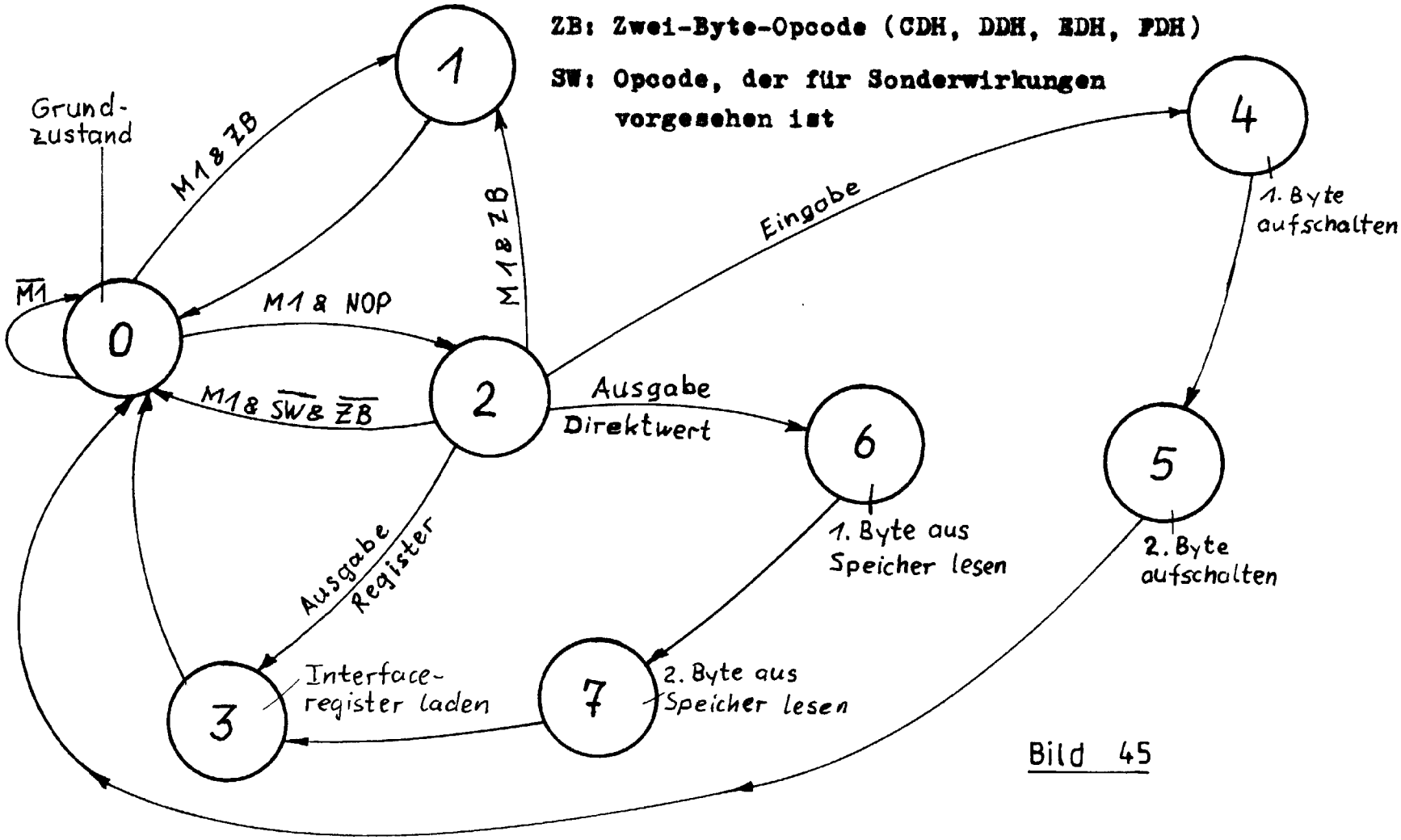


Bild 45

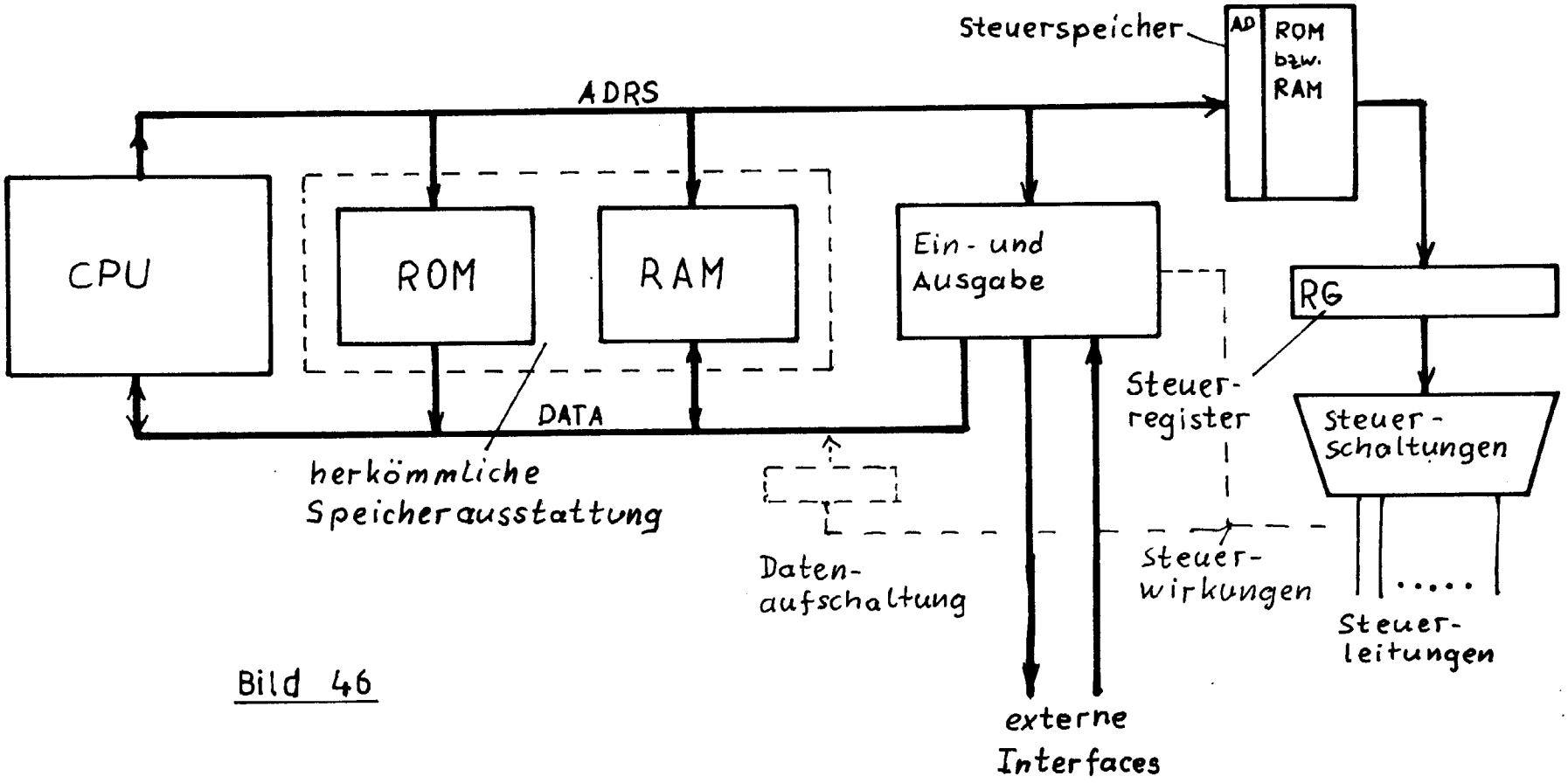

Steuerspeicher
AD
ROM
bzw.
RAM
ADRS
CPU
ROM
RAM
Ein- und
Ausgabe
RG
Steuer-
register
Steuer-
schaltungen
DATA
herkömmliche
Speicherausstattung
Daten-
aufschaltung
Steuer-
wirkungen
Steuer-
leitungen
externe
Interfaces

Bild 46

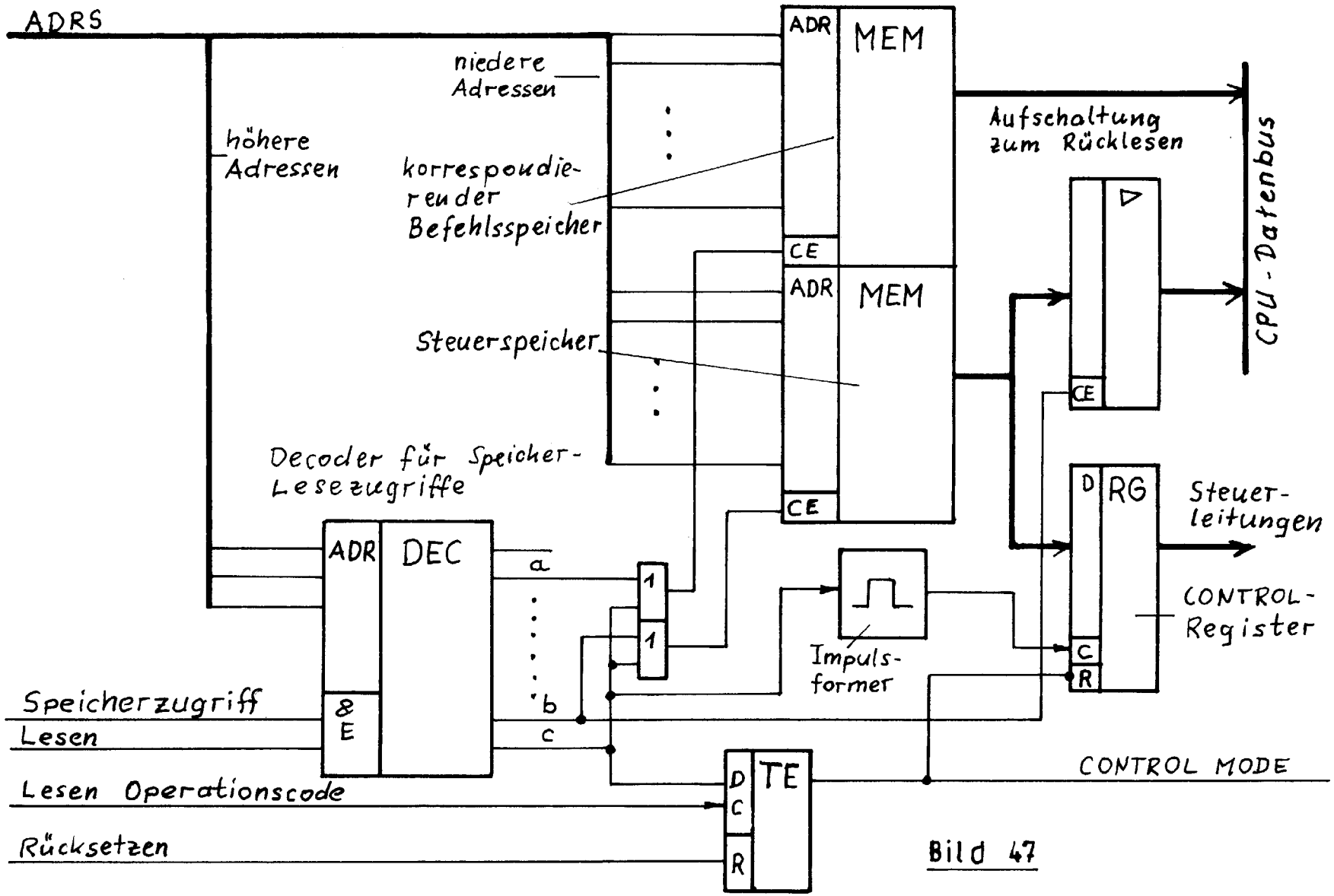

ADRS
niedere Adressen
höhere Adressen
korrespondierender Befehlsspeicher
Steuerspeicher
Decoder für Speicher-Lesezugriffe
ADR
MEM
CE
ADR
MEM
CE
Aufschaltung zum Rücklesen
CPU-Datenbus
CE
D
RG
C
R
Steuerleitungen
CONTROL-Register
ADR
DEC
&
E
a
b
c
1
1
Impulsformer
Speicherzugriff
Lesen
Lesen Operationscode
Rücksetzen
D
C
R
TE
CONTROL MODE

Bild 47

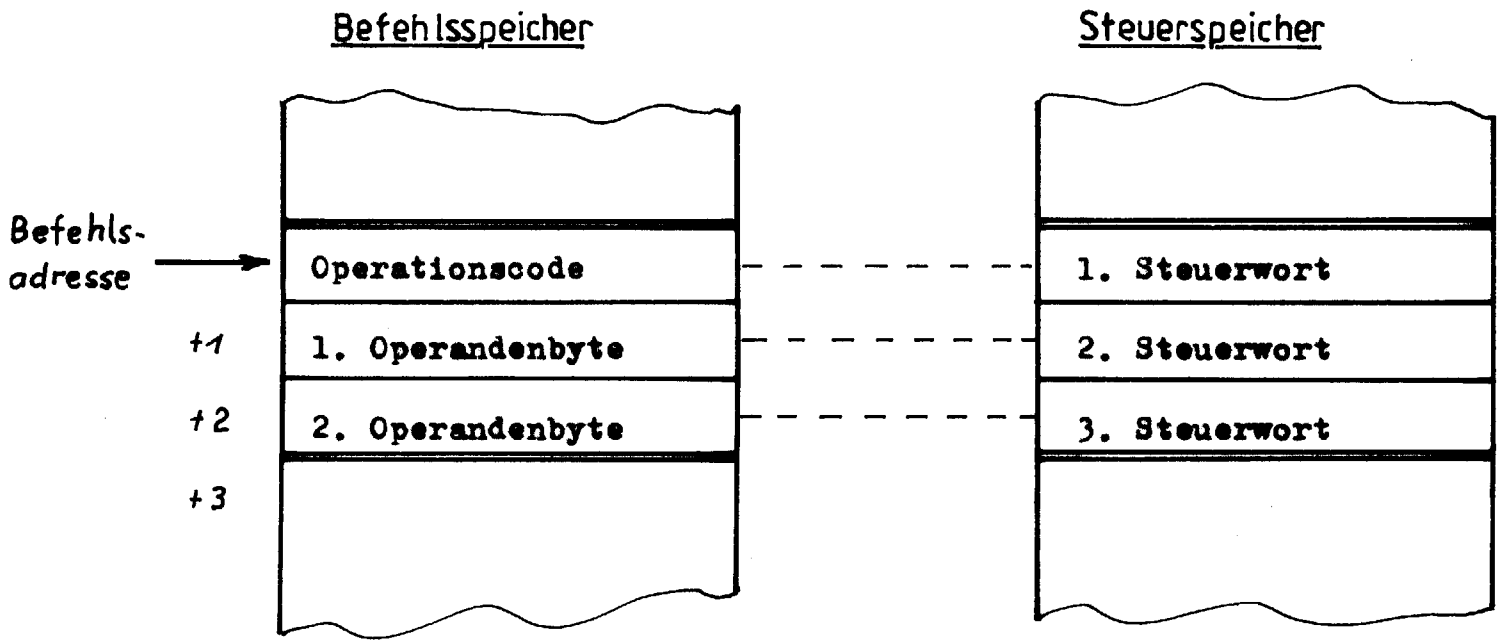

Befehlsspeicher
Steuerspeicher
Befehls-
adresse
Operationscode
1. Operandenbyte
2. Operandenbyte
+1
+2
+3
1. Steuerwort
2. Steuerwort
3. Steuerwort

Bild 48

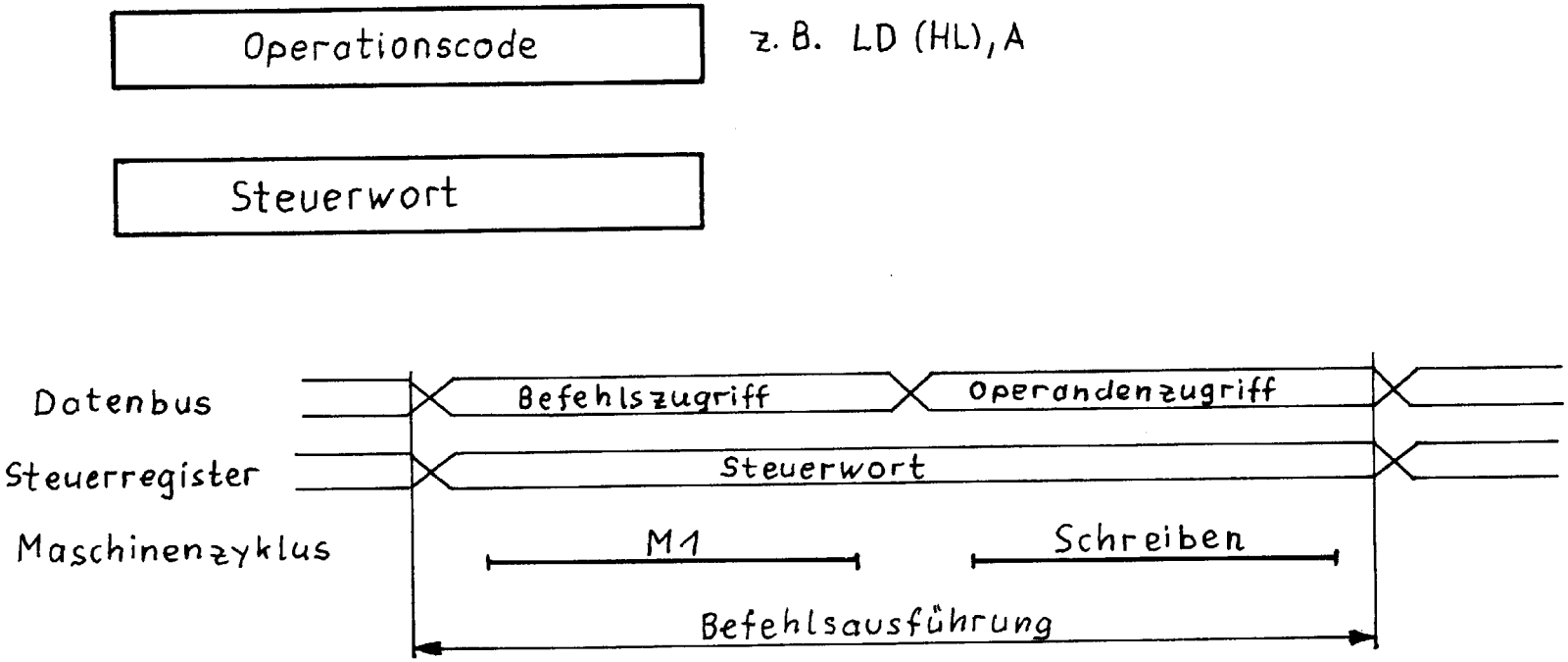

Operationscode
z. B. LD (HL), A
Steuerwort
Datenbus
Steuerregister
Maschinenzyklus
Befehlszugriff
Operandenzugriff
Steuerwort
M1
Schreiben
Befehlsausführung

Bild 49

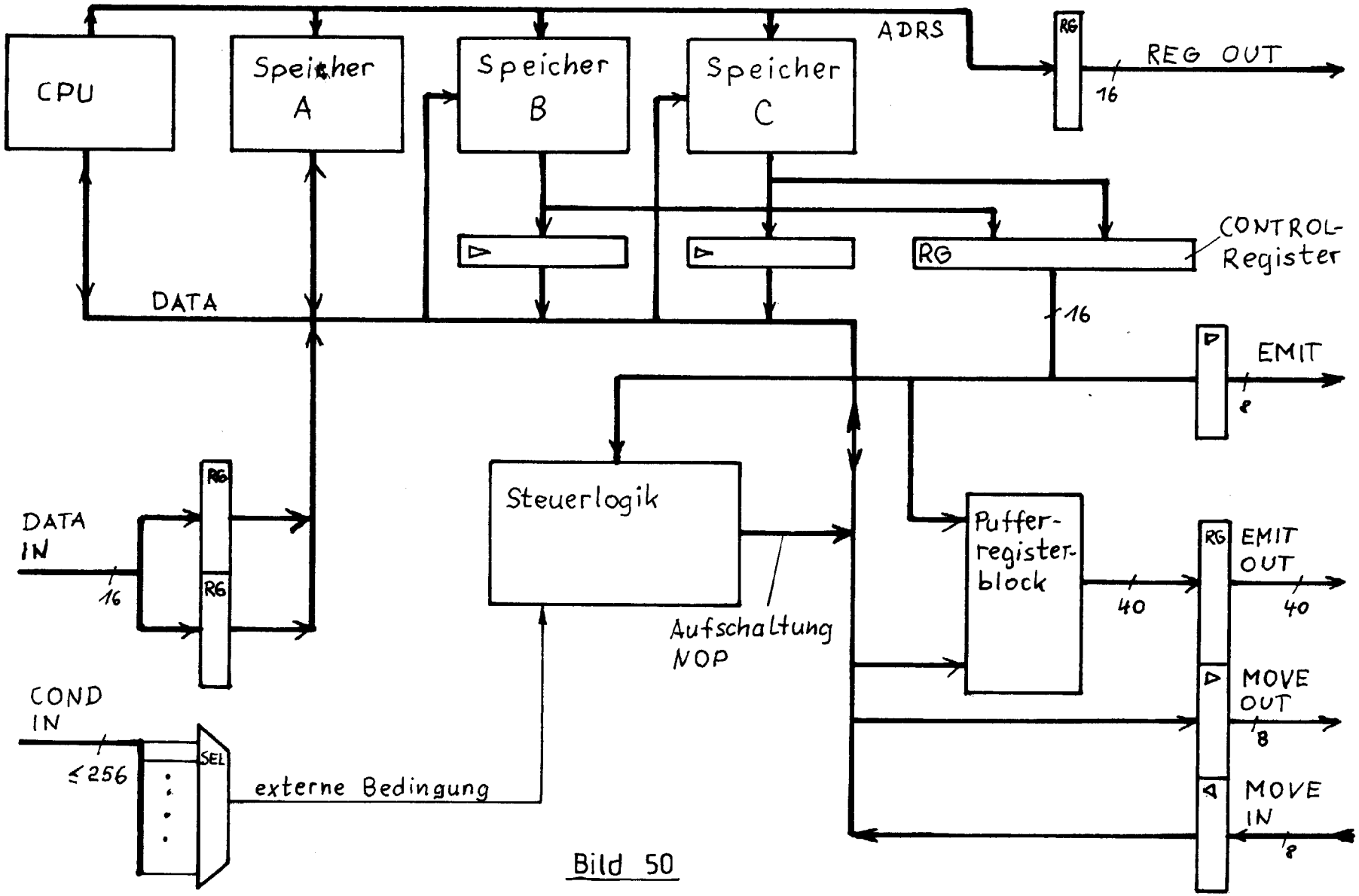
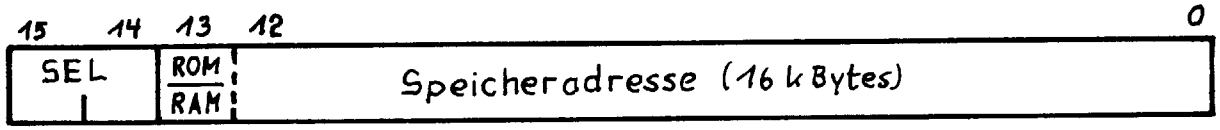

CPU
Speicher A
Speicher B
Speicher C
ADRS
RG
REG OUT
16
CONTROL-Register
RG
16
DATA
EMIT
8
Steuerlogik
Puffer-register-block
40
RG
EMIT OUT
40
MOVE OUT
8
MOVE IN
8
DATA IN
16
RG
RG
COND IN
≤256
SEL
externe Bedingung
Aufschaltung NOP

Bild 50



0 0 - Direktzugriff Speicher A

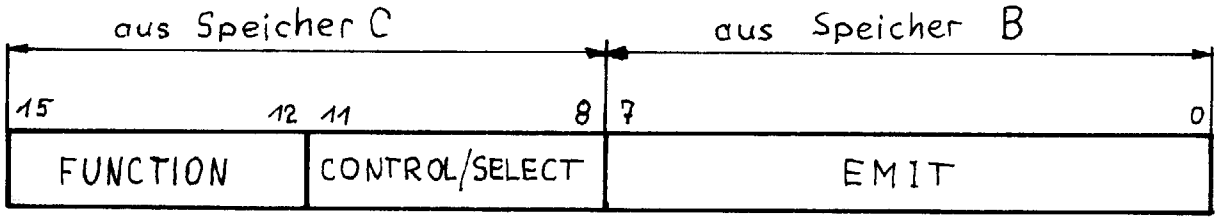
0 1 - CONTROL MODE - Zugriff (A + B + C parallel)

1 0 - Direktzugriff Speicher B

1 1 - Direktzugriff Speicher C

Adressenbit 13 = 0: ROM- Zugriff
= 1: RAM- Zugriff

Bild 51



Funktionen

- NO OP
- EMIT OUTPUT
- REGISTER OUTPUT
- REGISTER INPUT
- MOVE MEMORY TO INTERFACE
- MOVE INTERFACE TO MEMORY
- JUMP } ACCORDING TO EXTERNAL CONDITION
- SKIP }
- 8 Reserve- Funktionen

Bild 52

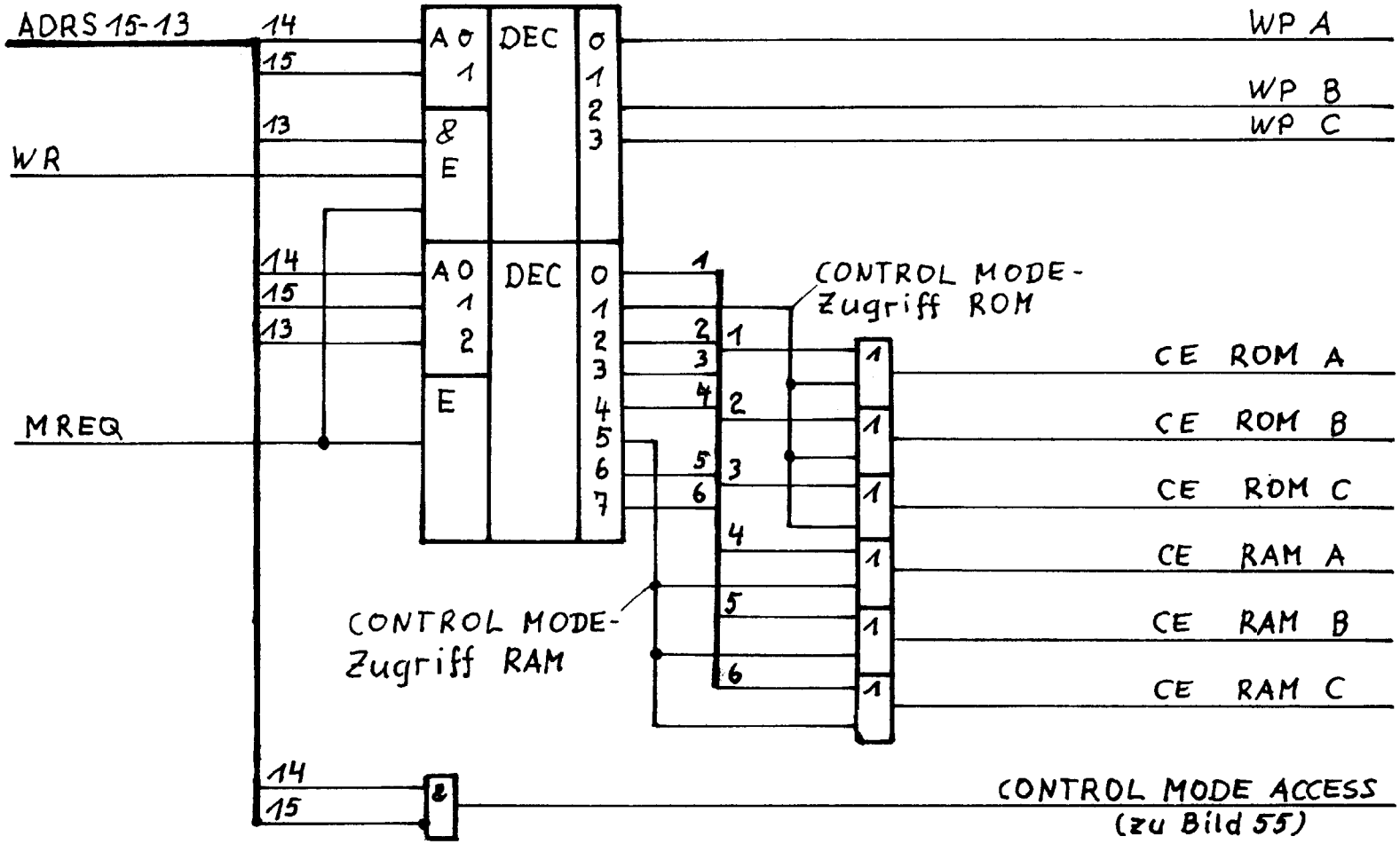

ADRS 15-13
WR
MREQ
14
15
13
A 0
1
&
E
DEC
0
1
2
3
A 0
1
2
E
DEC
0
1
2
3
4
5
6
7
WP A
WP B
WP C
CONTROL MODE-
Zugriff ROM
CONTROL MODE-
Zugriff RAM
CE ROM A
CE ROM B
CE ROM C
CE RAM A
CE RAM B
CE RAM C
CONTROL MODE ACCESS
(zu Bild 55)

Bild 53

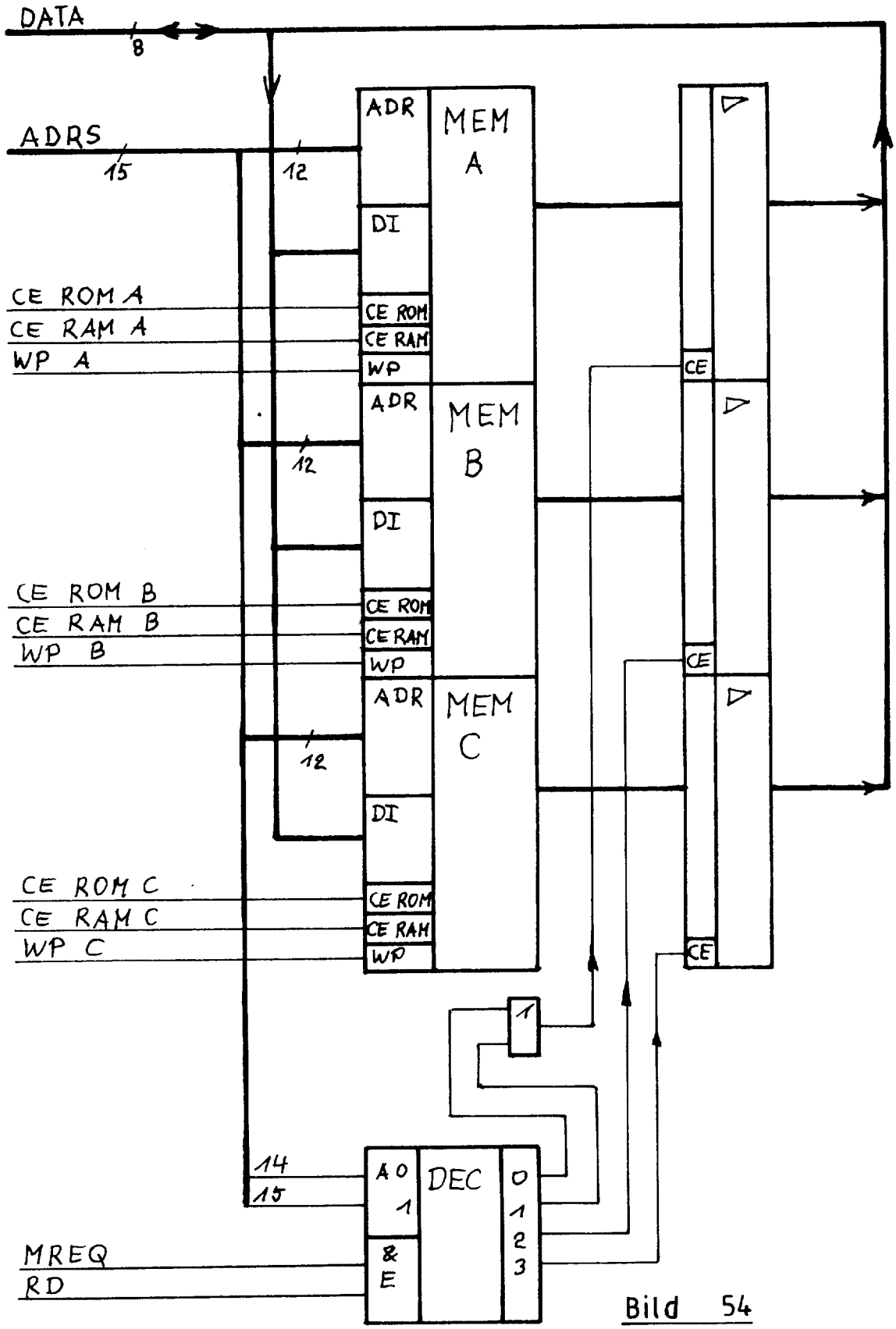

DATA
8
ADRS
15
12
ADR
MEM
A
DI
CE ROM A
CE RAM A
WP A
CE ROM
CE RAM
WP
CE
ADR
MEM
B
DI
CE ROM B
CE RAM B
WP B
CE ROM
CE RAM
WP
CE
ADR
MEM
C
DI
CE ROM C
CE RAM C
WP C
CE ROM
CE RAM
WP
CE
1
14
15
A 0
1
DEC
0
1
2
3
MREQ
RD
&
E

Bild 54

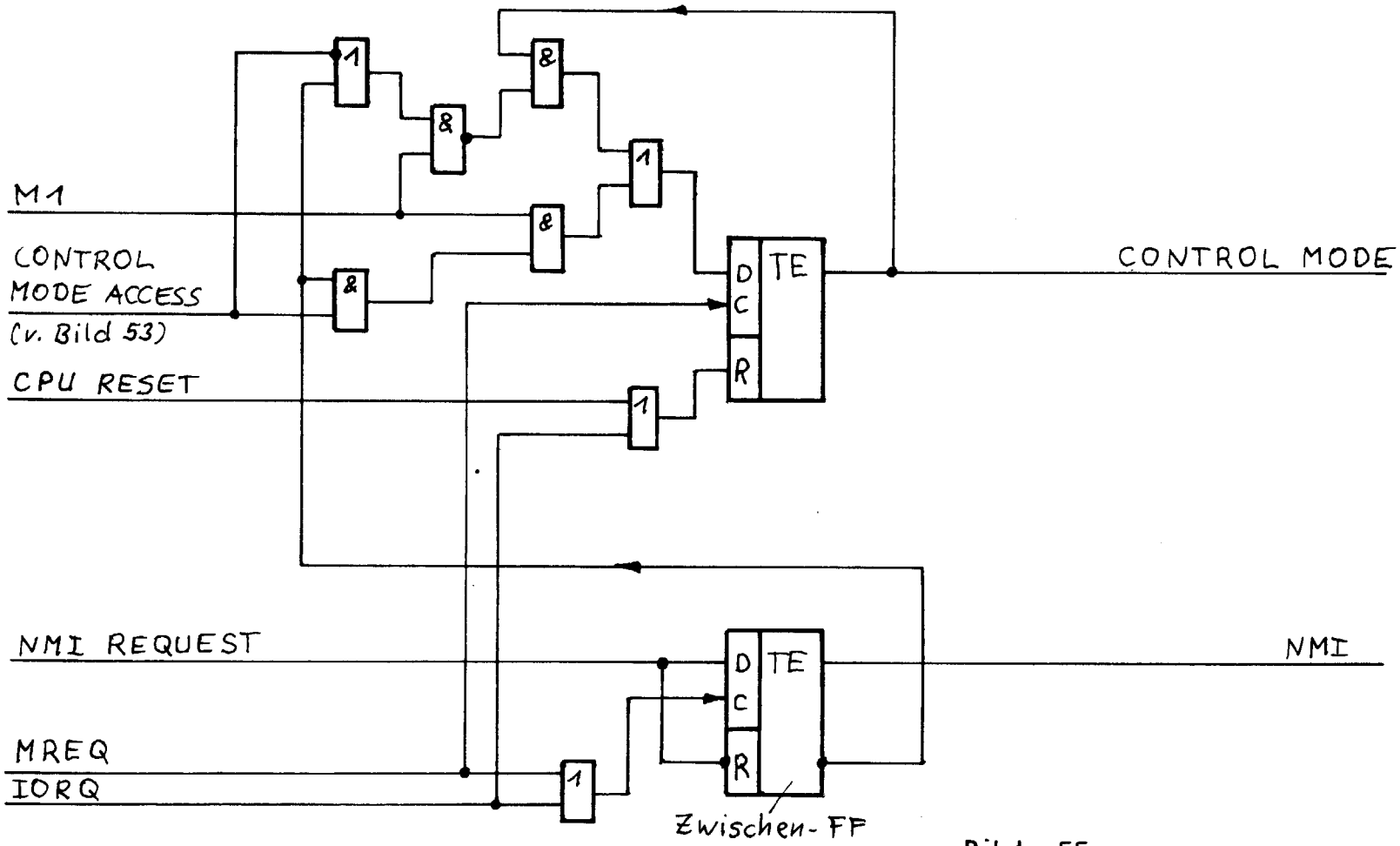

M1
CONTROL
MODE ACCESS
(v. Bild 53)
CPU RESET
NMI REQUEST
MREQ
IORQ
D
C
R
TE
CONTROL MODE
NMI
Zwischen-FF

Bild 55

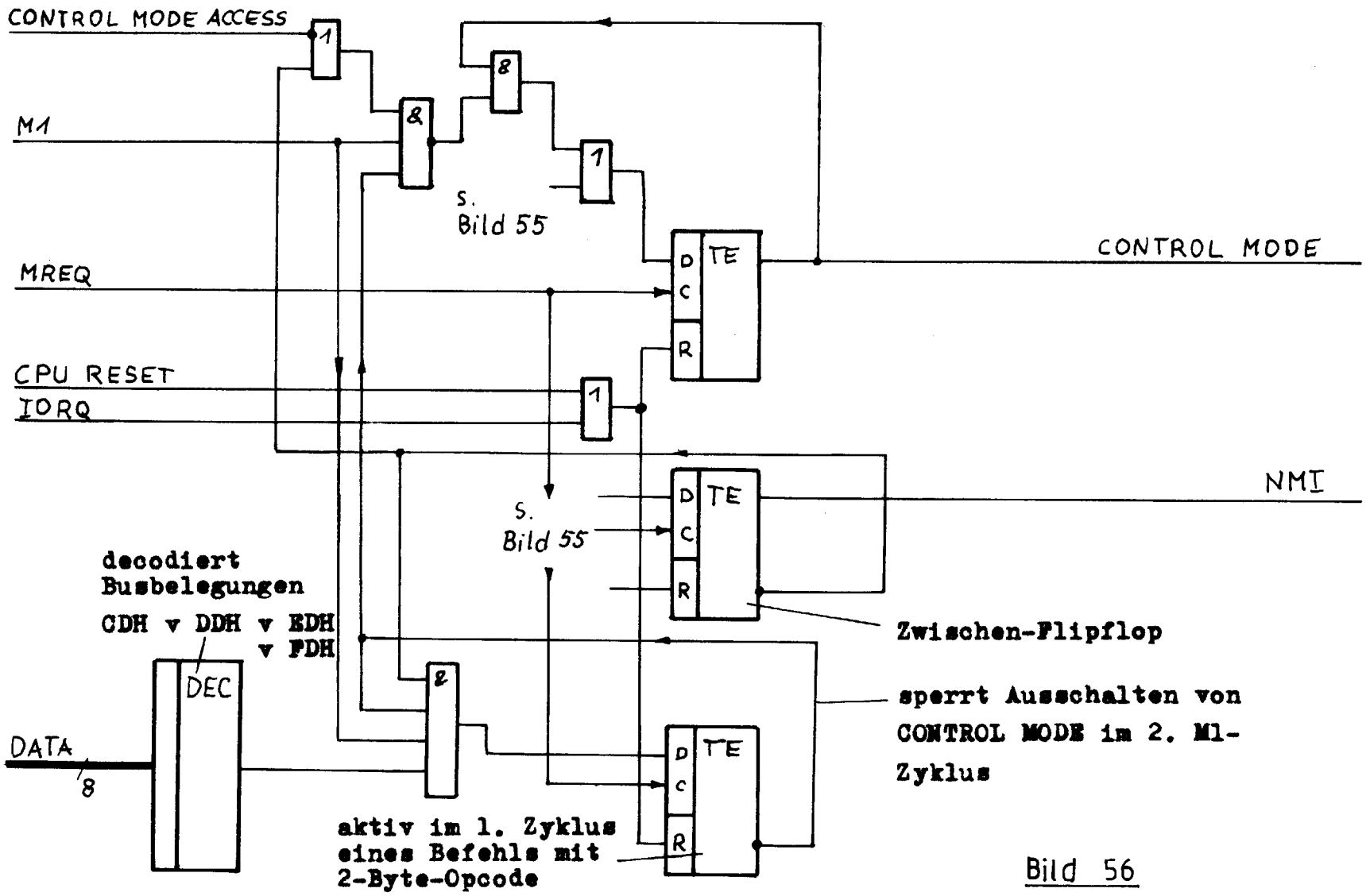

CONTROL MODE ACCESS
M1
MREQ
CPU RESET
IORQ
DATA
8
DEC
decodiert
Busbelegungen
CDH v DDH v EDH
v FDH
S.
Bild 55
S.
Bild 55
D
C
R
TE
CONTROL MODE
NMI
Zwischen-Flipflop
sperrt Ausschalten von
CONTROL MODE im 2. M1-
Zyklus
aktiv im 1. Zyklus
eines Befehls mit
2-Byte-Opcode

Bild 56

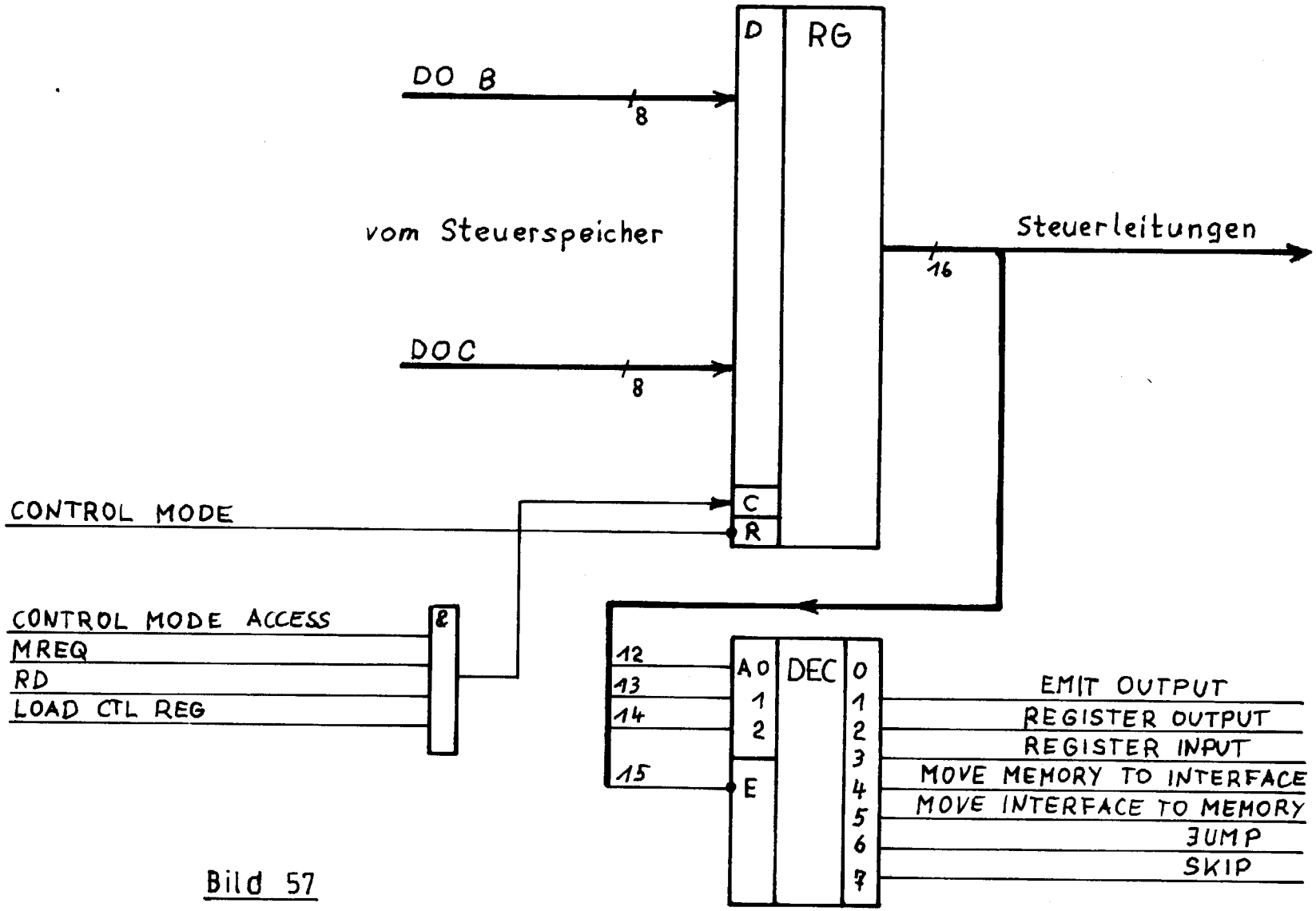
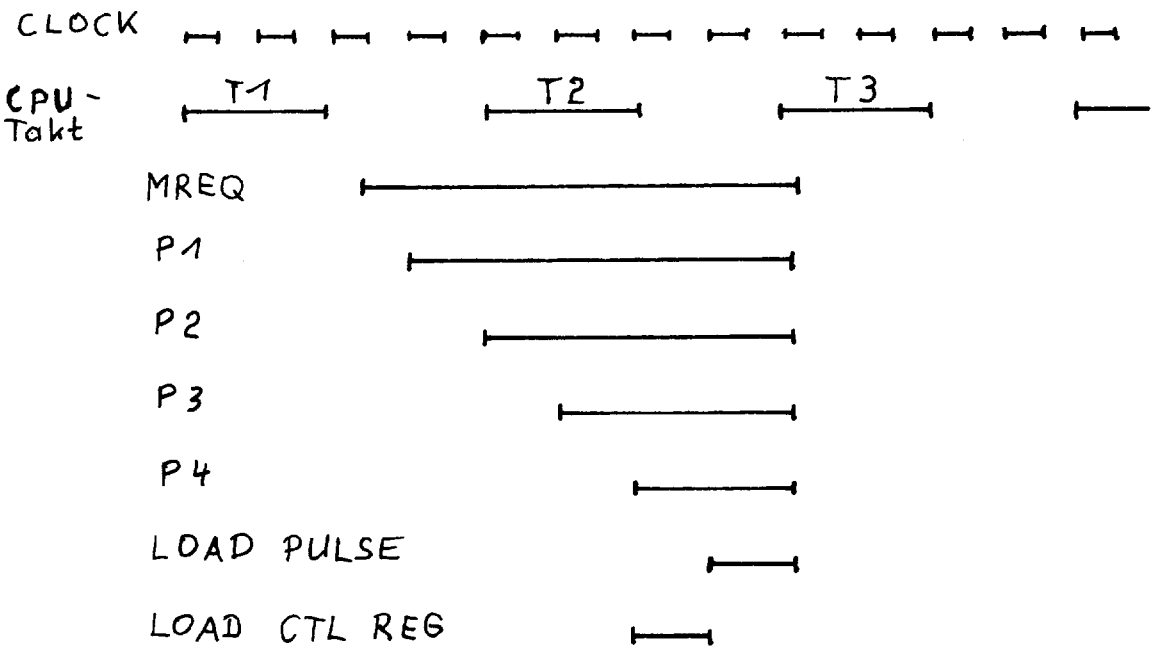
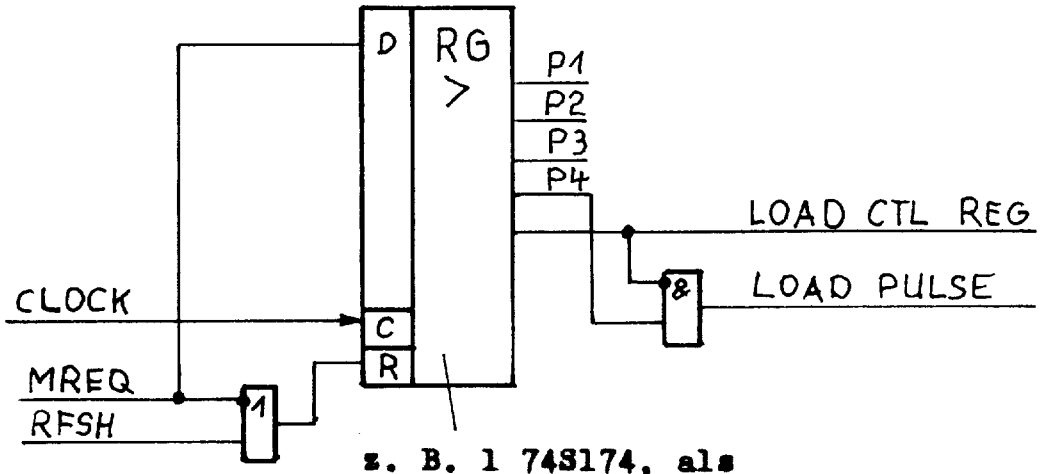

D
RG
DO B
8
vom Steuerspeicher
Steuerleitungen
16
DOC
8
C
R
CONTROL MODE
CONTROL MODE ACCESS
MREQ
RD
LOAD CTL REG
&
12
13
14
15
A 0
1
2
E
DEC
0
1
2
3
4
5
6
7
EMIT OUTPUT
REGISTER OUTPUT
REGISTER INPUT
MOVE MEMORY TO INTERFACE
MOVE INTERFACE TO MEMORY
JUMP
SKIP

Bild 57





z. B. 1 74S174, als Schieberegister verschaltet

Bild 58

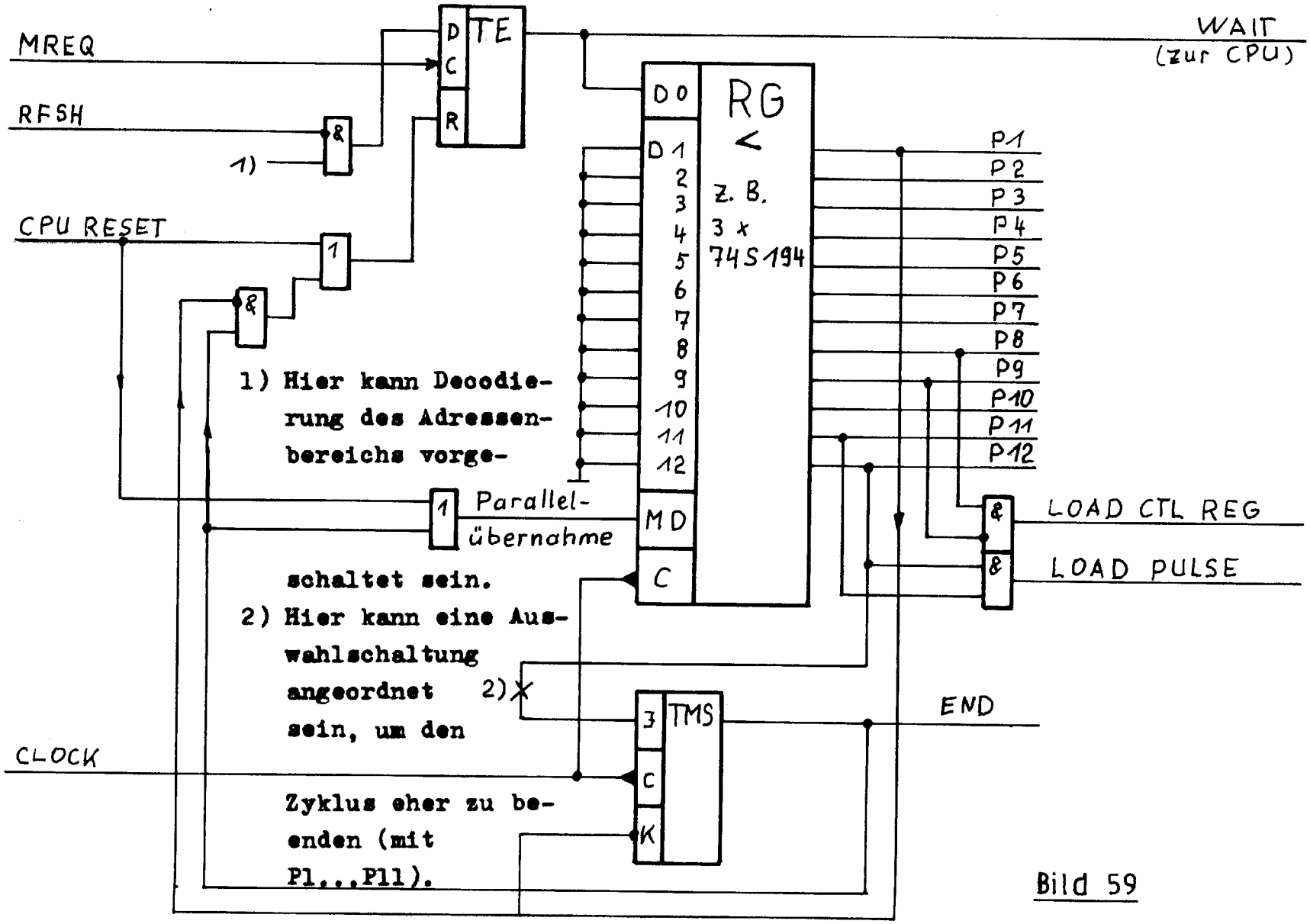

MREQ
RFSH
CPU RESET
CLOCK
TE
D
C
R
&
1
1)
WAIT
(zur CPU)
RG
D0
D1
2
3
4
5
6
7
8
9
10
11
12
MD
C
z. B.
3 x
74S194
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
LOAD CTL REG
LOAD PULSE
Parallel-
übernahme
1) Hier kann Decodierung des Adressenbereichs vorgeschaltet sein.
2) Hier kann eine Auswahlschaltung angeordnet sein, um den Zyklus eher zu beenden (mit P1...P11).
2)
3
TMS
K
END

Bild 59

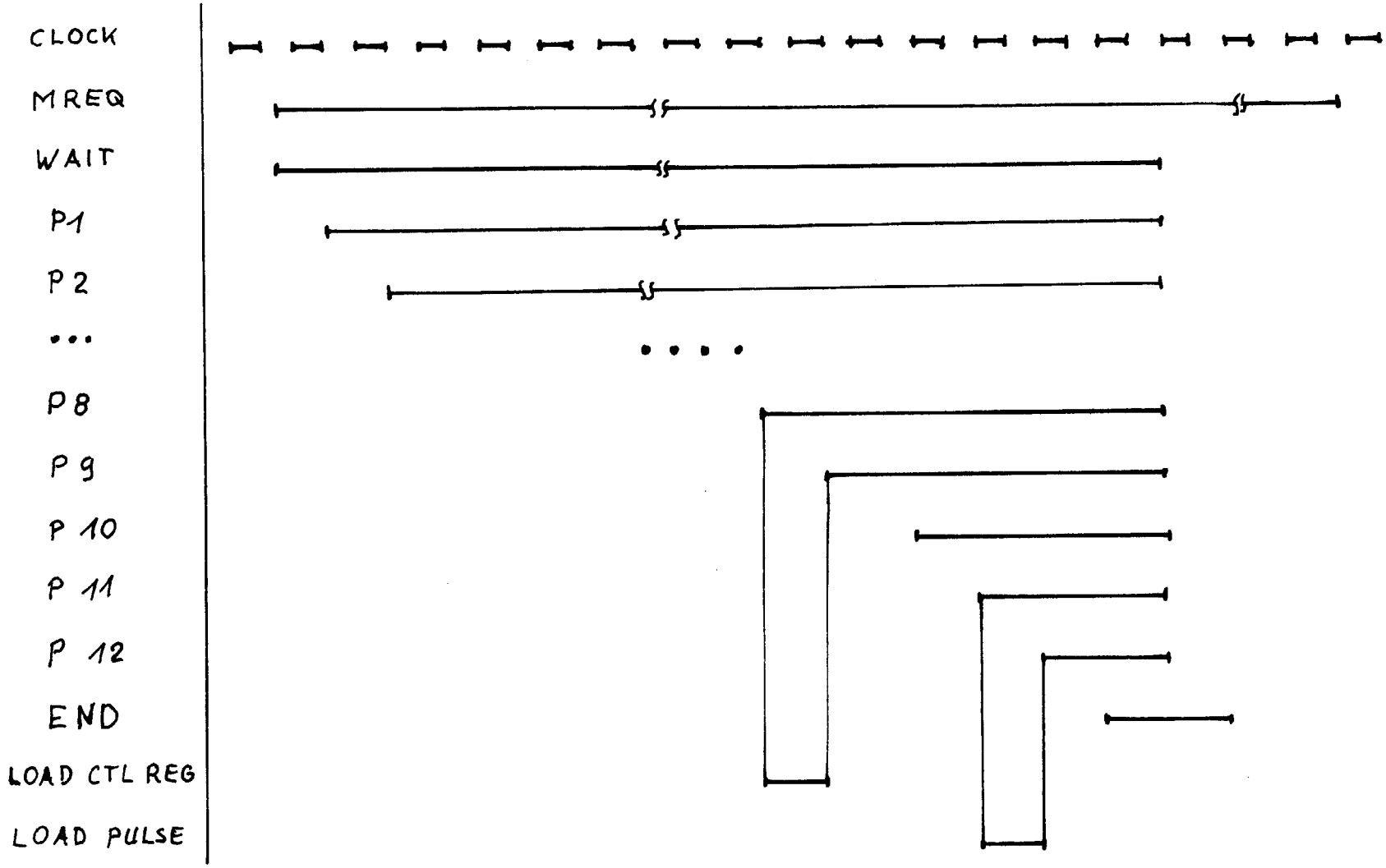

CLOCK
MREQ
WAIT
P1
P 2
...
P 8
P 9
P 10
P 11
P 12
END
LOAD CTL REG
LOAD PULSE

Bild 60

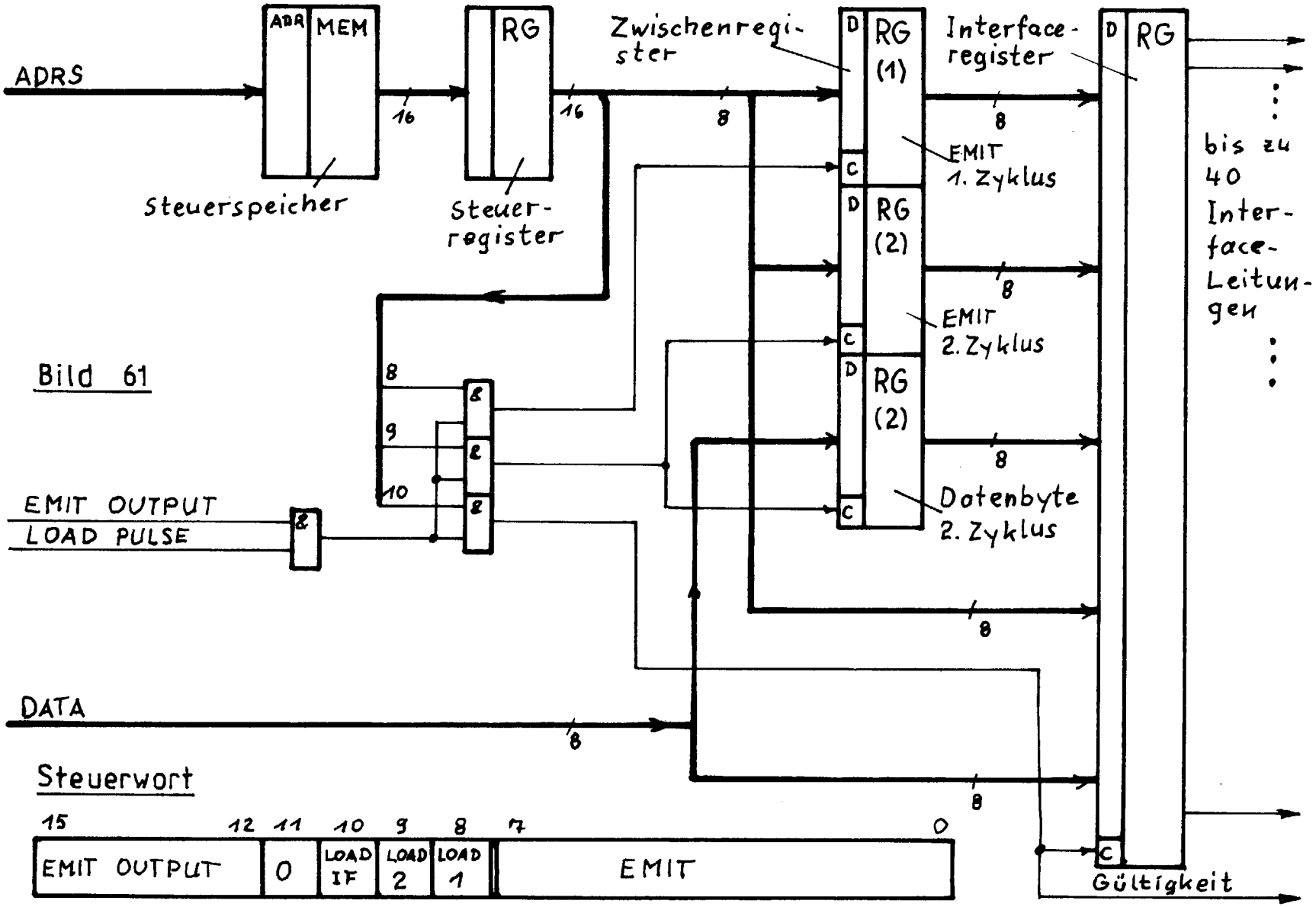

ADRS
ADR
MEM
Steuerspeicher
16
RG
Steuer-
register
16
Zwischenregi-
ster
8
D
RG
(1)
C
EMIT
1. Zyklus
D
RG
(2)
C
EMIT
2. Zyklus
D
RG
(2)
C
Datenbyte
2. Zyklus
Interface-
register
8
D
RG
bis zu
40
Inter-
face-
Leitun-
gen
C
Gültigkeit
8
9
10
&
EMIT OUTPUT
LOAD PULSE
DATA
Steuerwort
15
12
11
10
9
8
7
0
EMIT OUTPUT
0
LOAD IF
LOAD 2
LOAD 1
EMIT

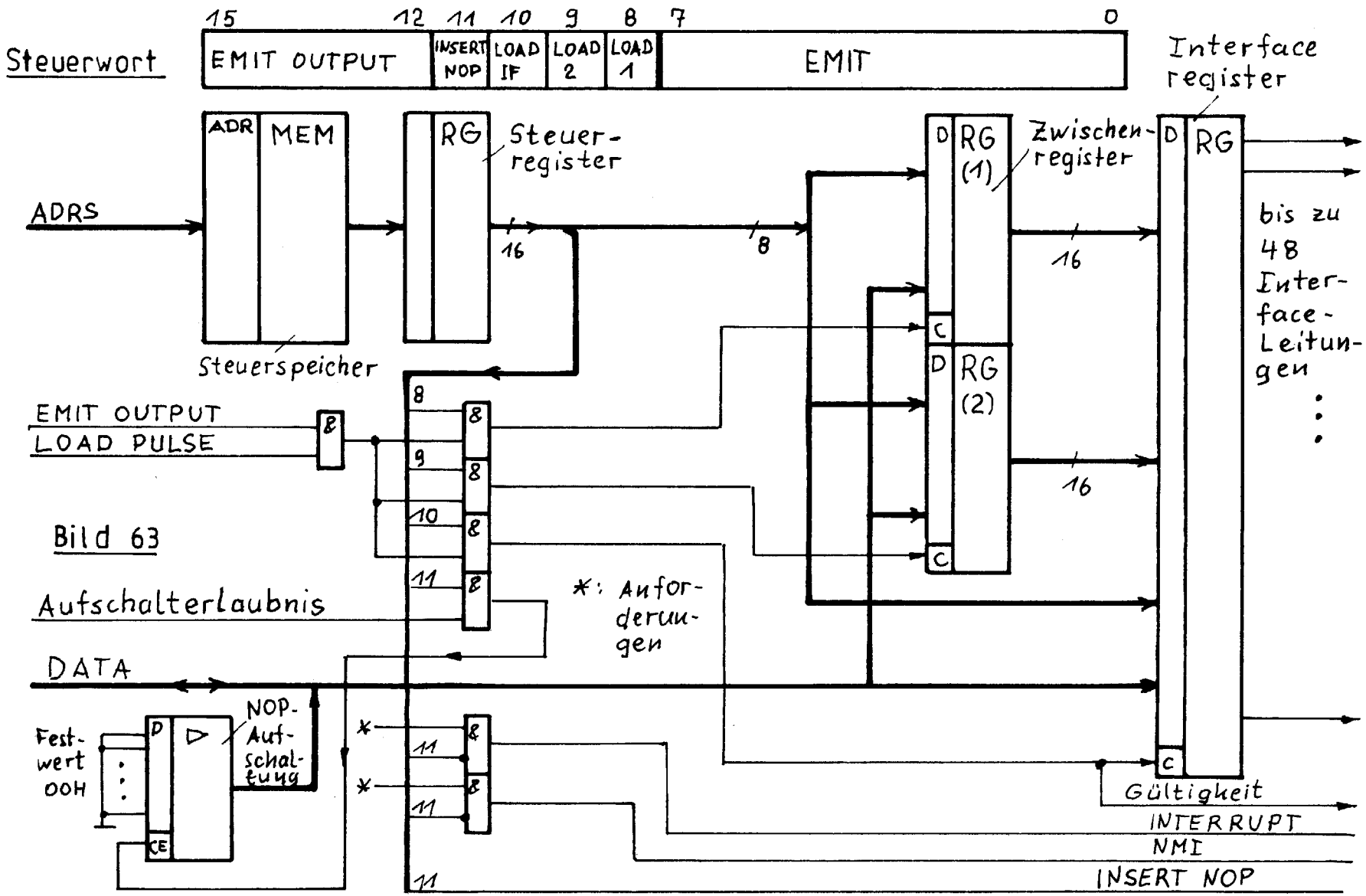
Bild 61

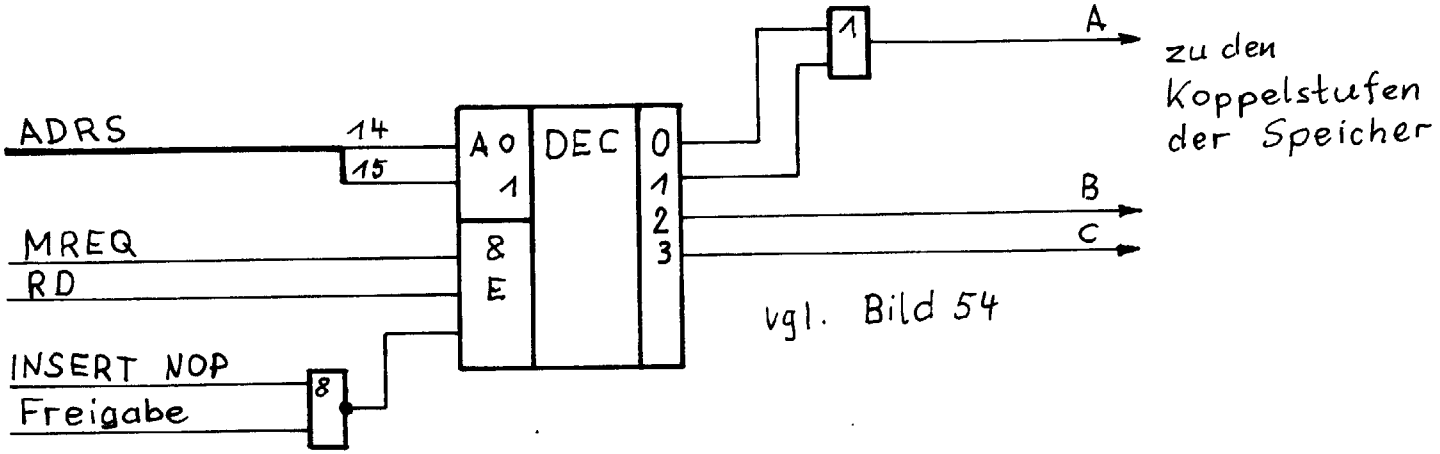
Zyklus	Speicher A	Speicher B					Speicher C
1	LD HL, mn Opcode	EMIT OUTPUT	0	0	0	1	EMIT 1
2	n	EMIT OUTPUT	0	0	1	0	EMIT 2
3	m	EMIT OUTPUT	0	1	0	0	EMIT 3

Interfacebelegung

39				0
m	EMIT 3	n	EMIT 2	EMIT 1
3. Zyklus		2. Zyklus		1. Zyklus

Bild 62


Steuerwort
15
12
11
10
9
8
7
0
EMIT OUTPUT
INSERT NOP
LOAD IF
LOAD 2
LOAD 1
EMIT
Interface register
ADR
MEM
RG
Steuer-register
Steuerspeicher
ADRS
16
8
D
RG (1)
Zwischen-register
C
RG (2)
bis zu 48 Inter-face-Leitungen
EMIT OUTPUT
LOAD PULSE
&
Bild 63
Aufschalterlaubnis
*: Anfor-derun-gen
DATA
Fest-wert 00H
NOP-Auf-schal-tung
CE
Gültigkeit
INTERRUPT
NMI
INSERT NOP



Zeitbedingungen

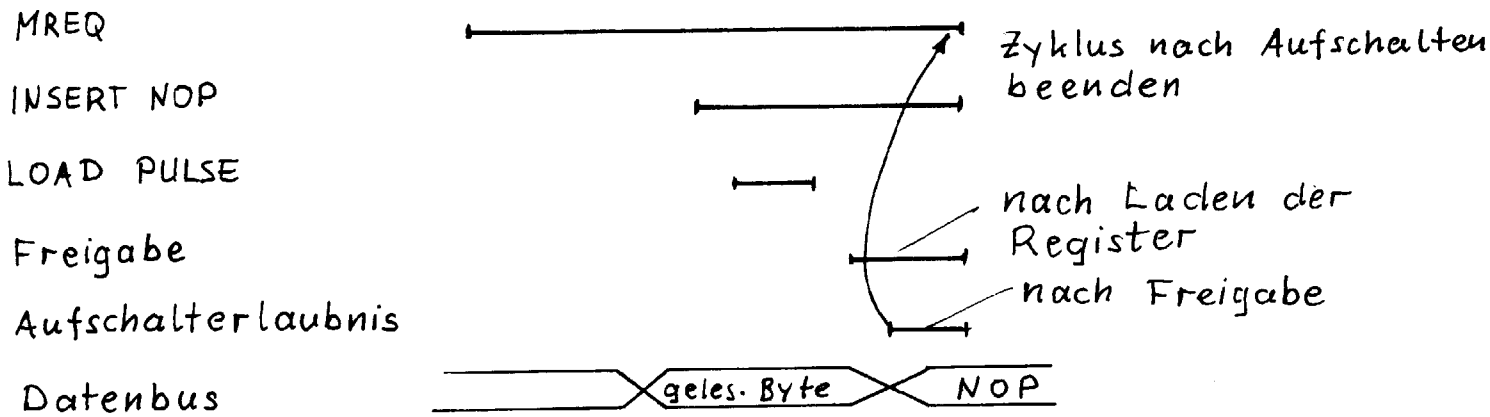


Bild 64

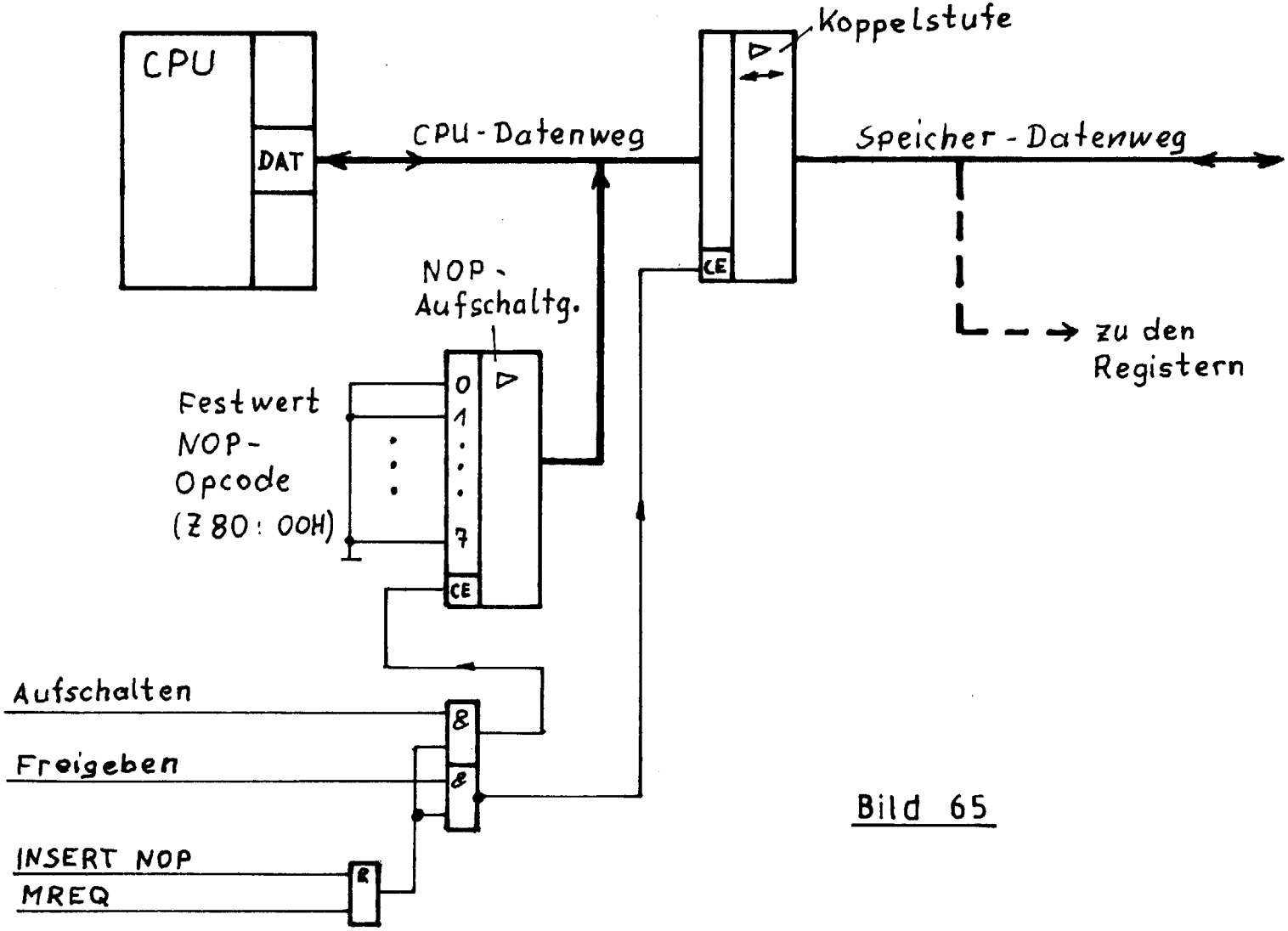
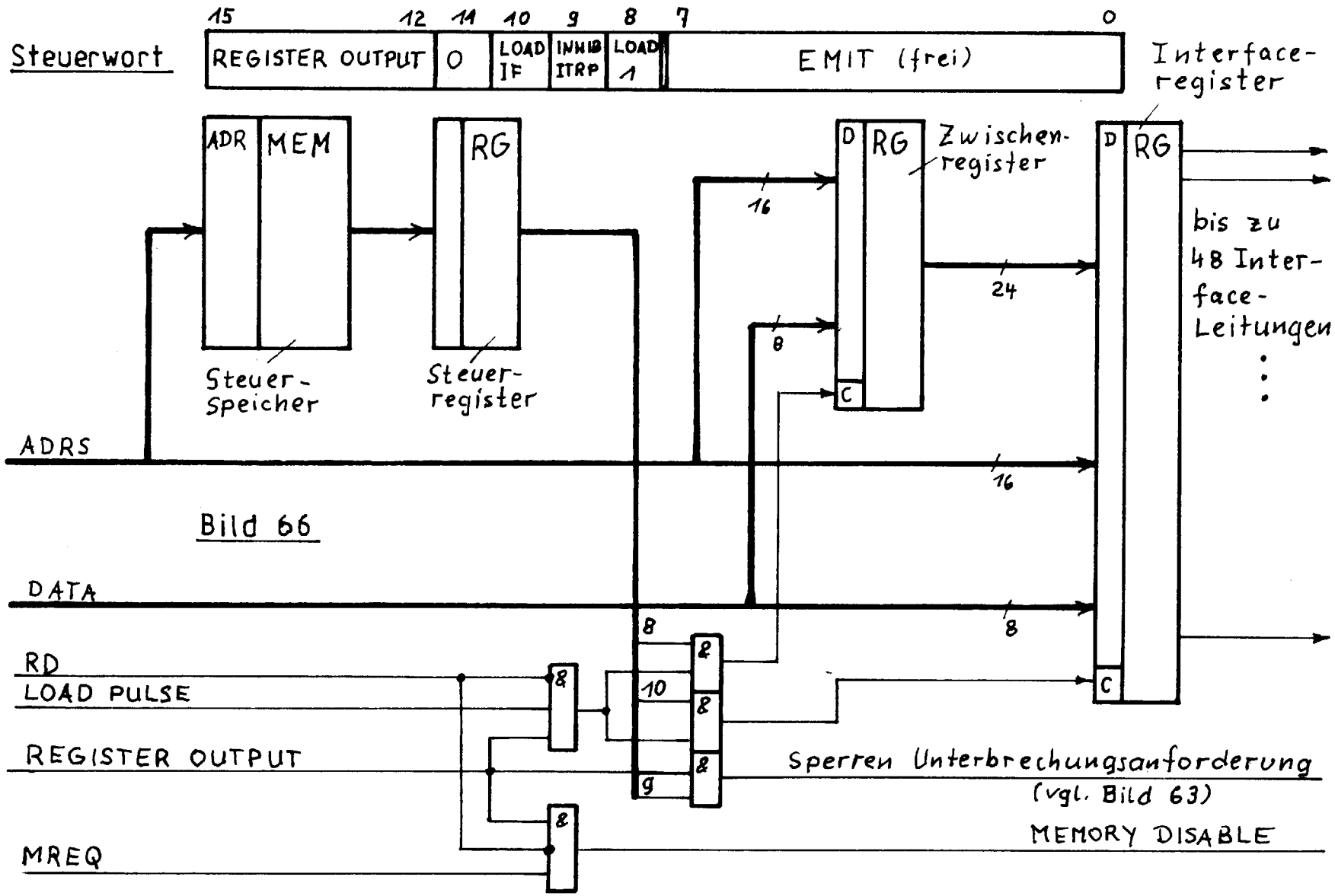

CPU
DAT
CPU-Datenweg
Koppelstufe
Speicher-Datenweg
CE
zu den Registern
NOP-Aufschaltg.
Festwert NOP-Opcode (Z80: 00H)
0
1
7
CE
Aufschalten
Freigeben
INSERT NOP
MREQ

Bild 65


Steuerwort
15
12
11
10
9
8
7
0
REGISTER OUTPUT
0
LOAD IF
INHIB ITRP
LOAD 1
EMIT (frei)
Interface-register
ADR
MEM
Steuer-Speicher
RG
Steuer-register
D
RG
C
Zwischen-register
16
24
8
D
RG
C
bis zu 48 Interface-Leitungen
ADRS
16
Bild 66
DATA
8
8
RD
LOAD PULSE
10
REGISTER OUTPUT
9
Sperren Unterbrechungsanforderung
(vgl. Bild 63)
MEMORY DISABLE
MREQ

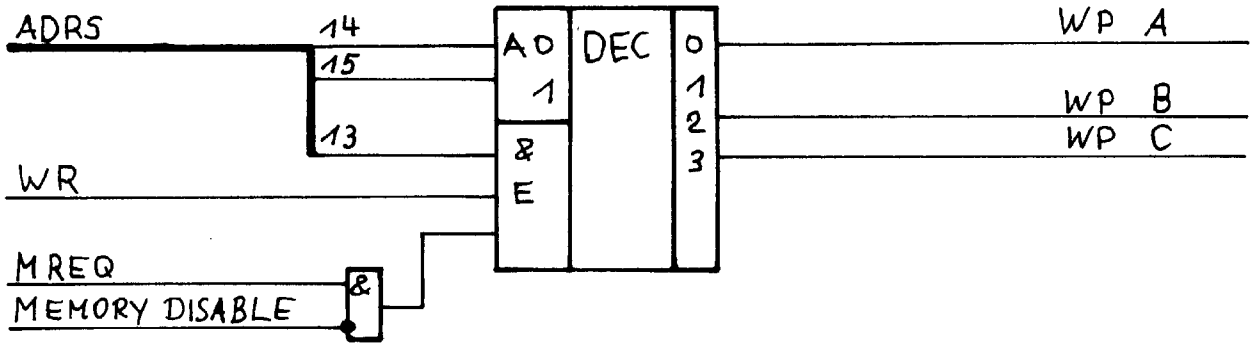
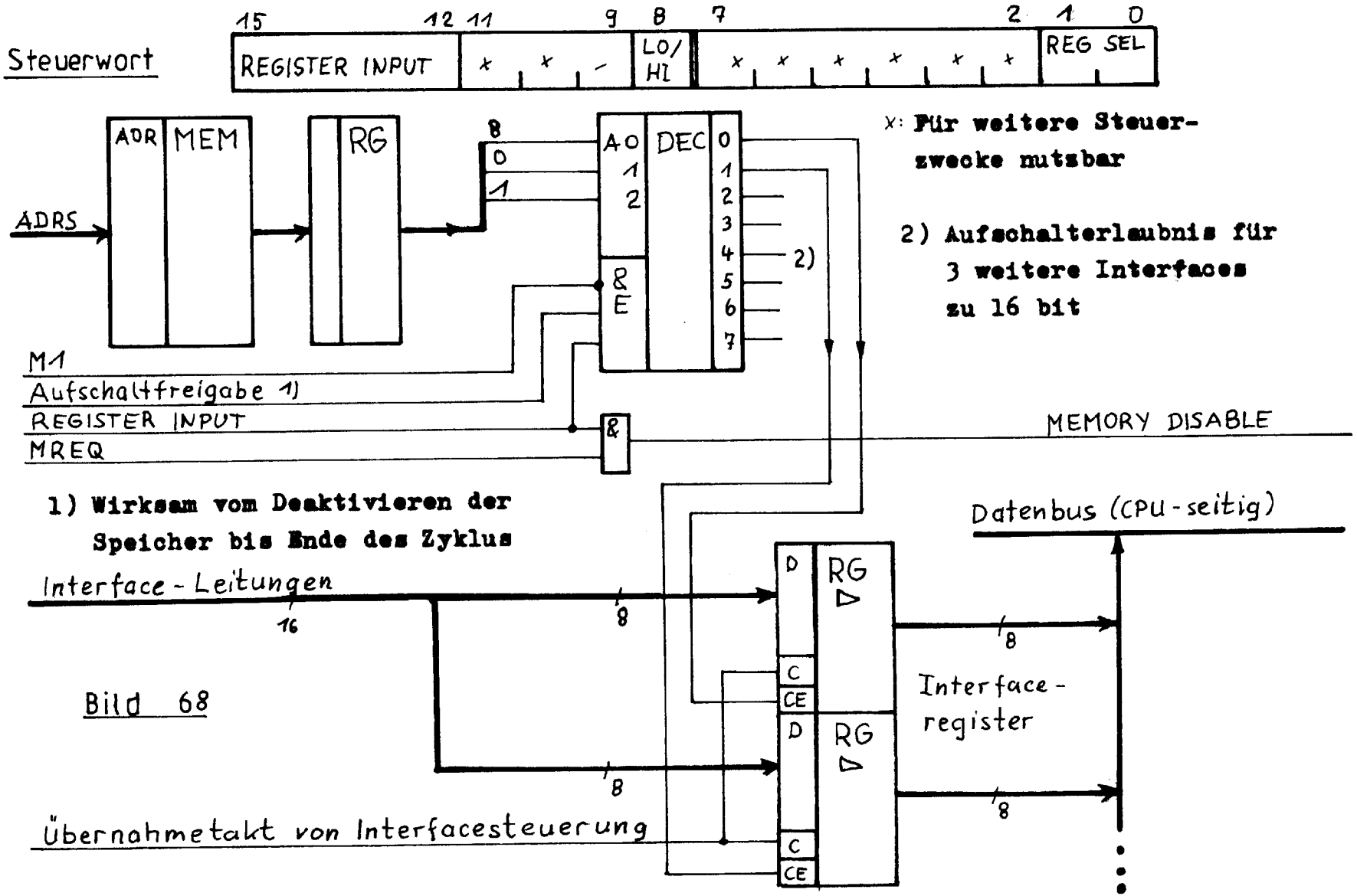


Bild 67



x: **Für weitere Steuerzwecke nutzbar**

2) **Aufschalterlaubnis für 3 weitere Interfaces zu 16 bit**

1) **Wirksam vom Deaktivieren der Speicher bis Ende des Zyklus**

Bild 68

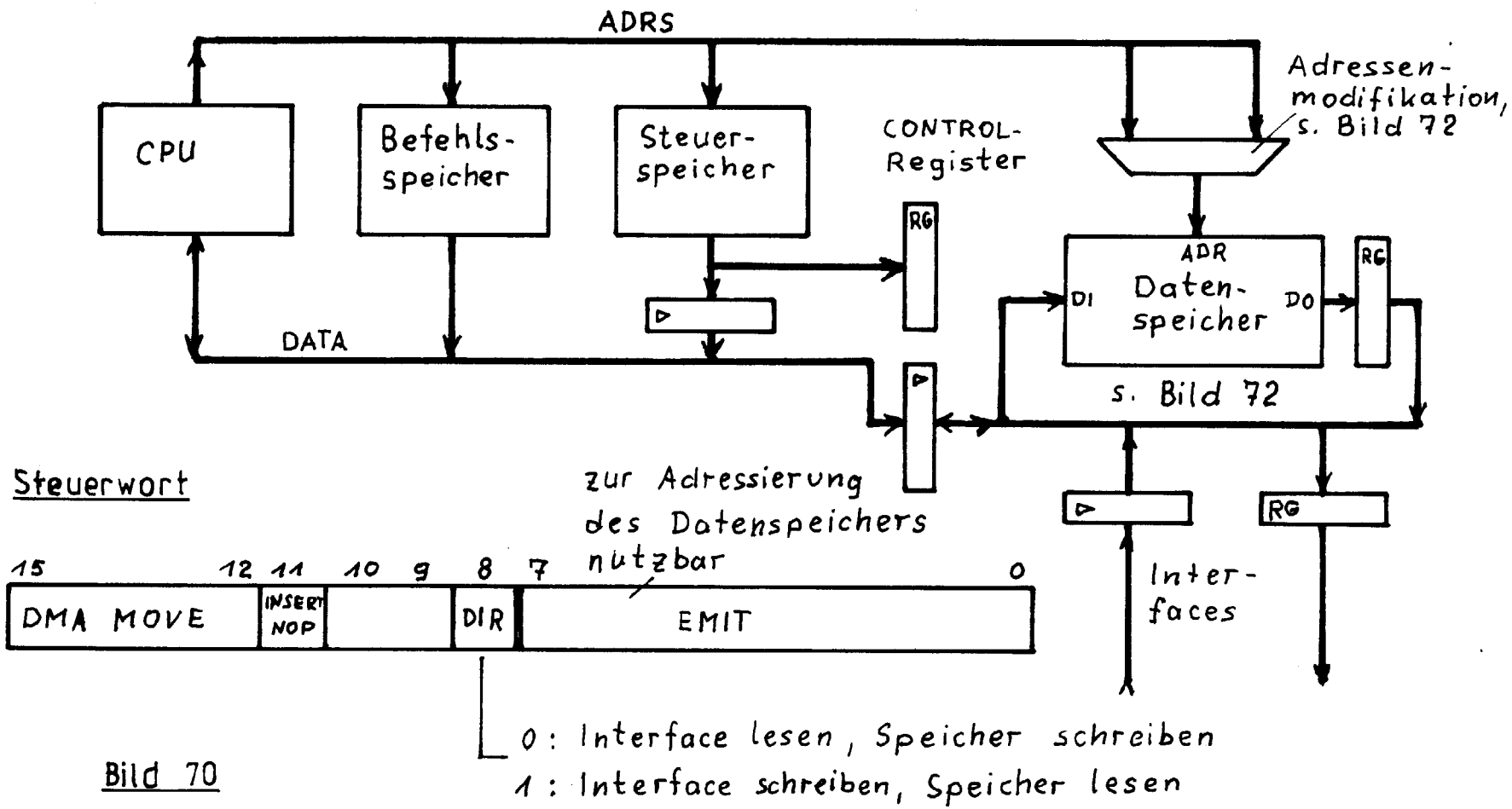
Befehlsspeicher

LD HL, mn Opcode
n
m

Steuerspeicher

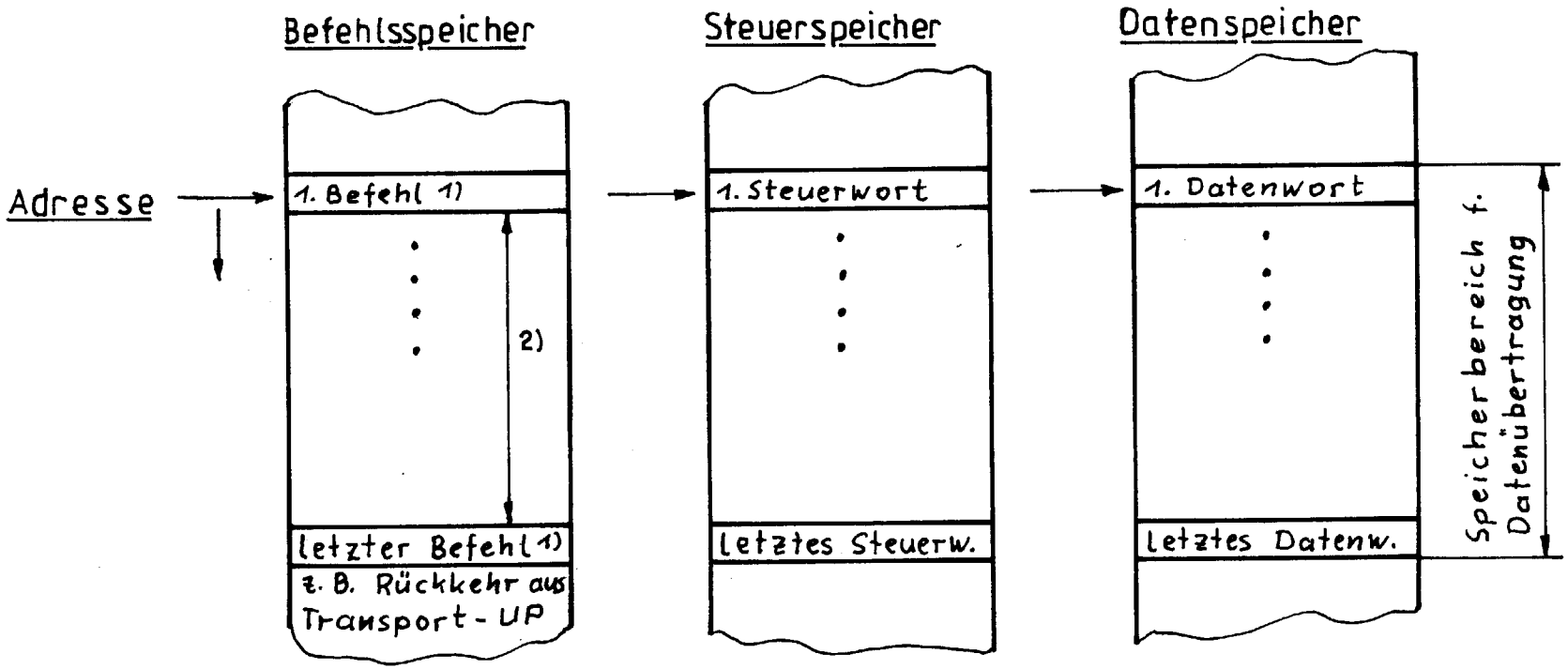
OH	x, x, x	x	x x x x x x x x	NO-OP-Belegung
REGISTER INPUT	x, x, x	0	Auswahl	
REGISTER INPUT	x, x, x	1	Auswahl	

Bild 69



Steuerwort

Bild 70



Der niedere Teil der Adresse ist für alle 3 Speicher gleich!

Bei NOP- Aufschaltung

1) **Speicherinhalt muß NOP sein.**

2) **An sich beliebiger Speicherinhalt (z. B. Programme außerhalb des CONTROL MODE).**

Bild 71

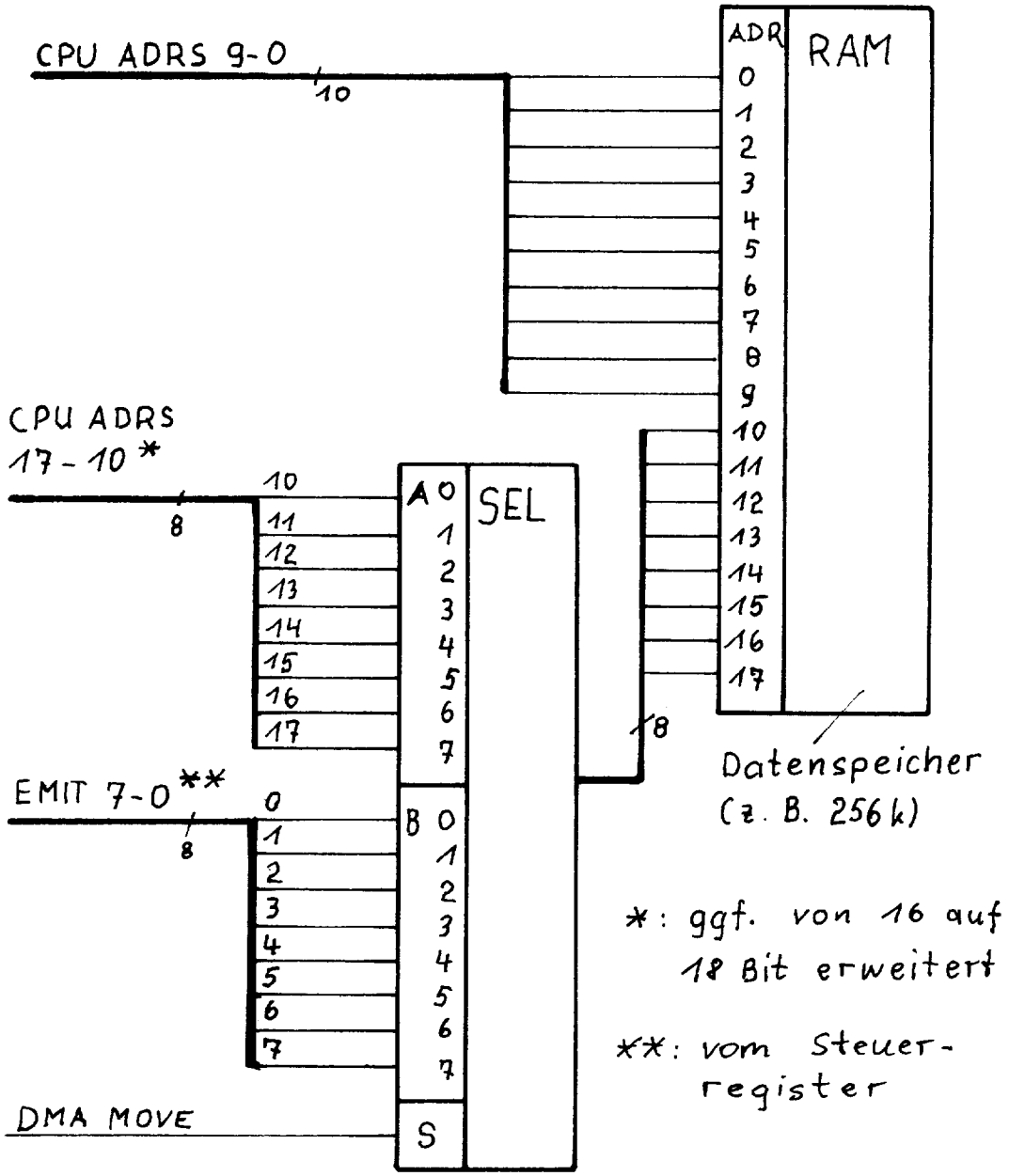

CPU ADRS 9-0
10
ADR
RAM
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
CPU ADRS
17 - 10 *
8
10
11
12
13
14
15
16
17
A 0
1
2
3
4
5
6
7
SEL
EMIT 7-0 **
8
0
1
2
3
4
5
6
7
B 0
1
2
3
4
5
6
7
S
DMA MOVE
18
Datenspeicher
(z. B. 256 k)
*: ggf. von 16 auf
18 Bit erweitert
**: vom Steuer-
register

Bild 72

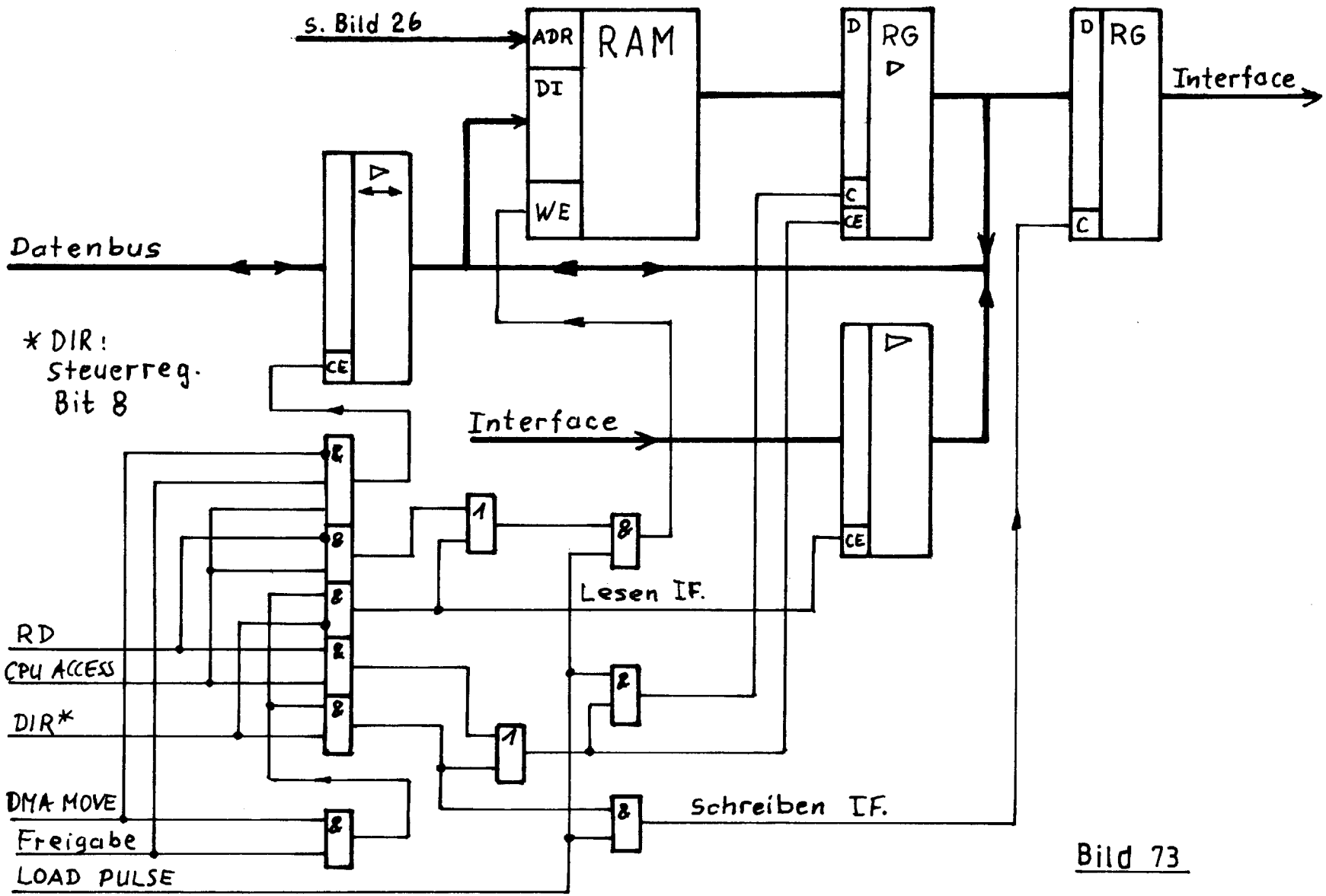

s. Bild 26
ADR
RAM
DI
WE
D
RG
C
CE
D
RG
C
Interface
Datenbus
CE
* DIR:
Steuerreg.
Bit 8
Interface
CE
Lesen IF.
RD
CPU ACCESS
DIR*
DMA MOVE
Freigabe
LOAD PULSE
Schreiben IF.

Bild 73

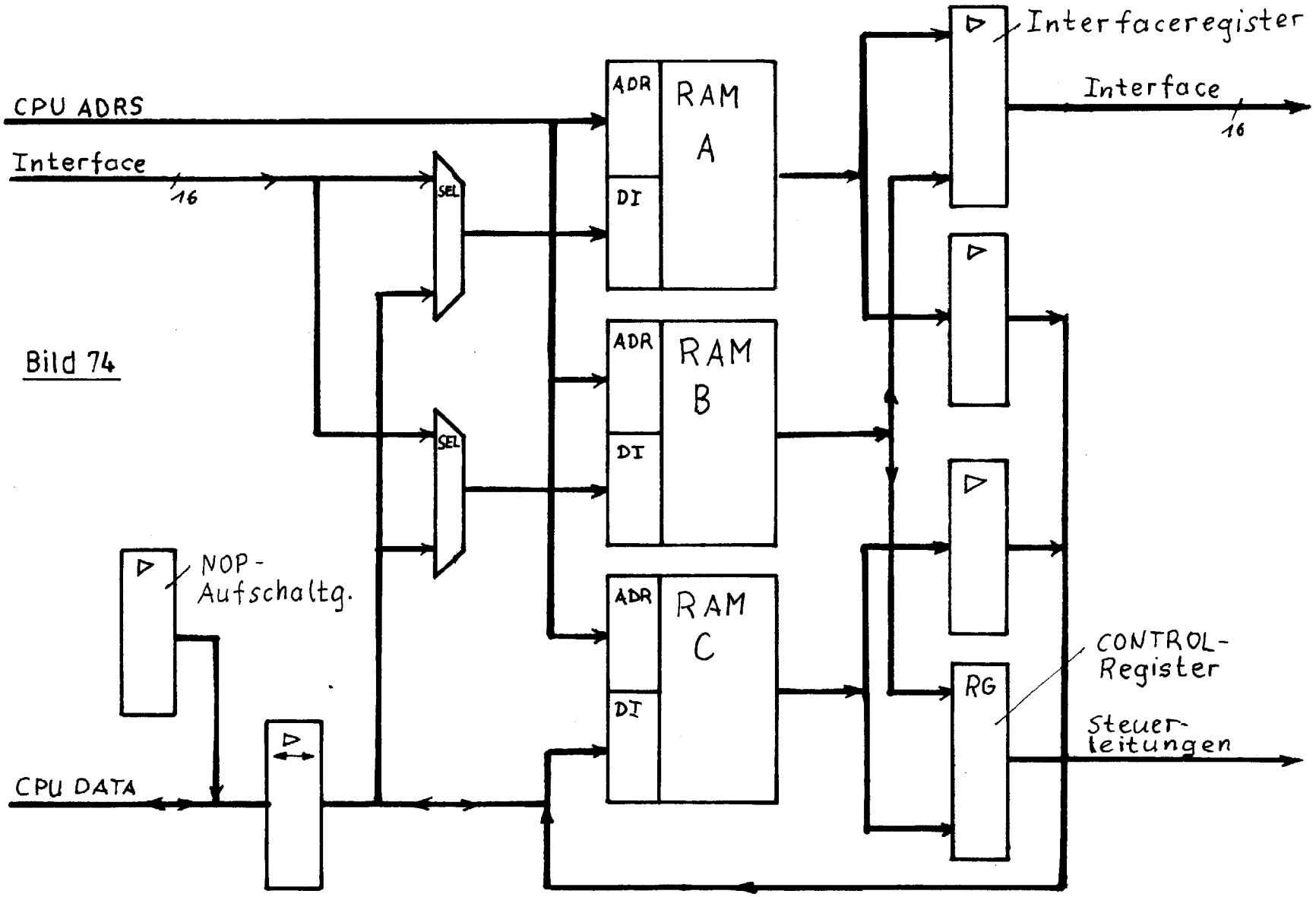

CPU ADRS
Interface
16
SEL
SEL
ADR
DI
RAM
A
ADR
DI
RAM
B
ADR
DI
RAM
C
Interfaceregister
Interface
16
NOP-
Aufschaltg.
CPU DATA
RG
CONTROL-
Register
Steuer-
leitungen

Bild 74

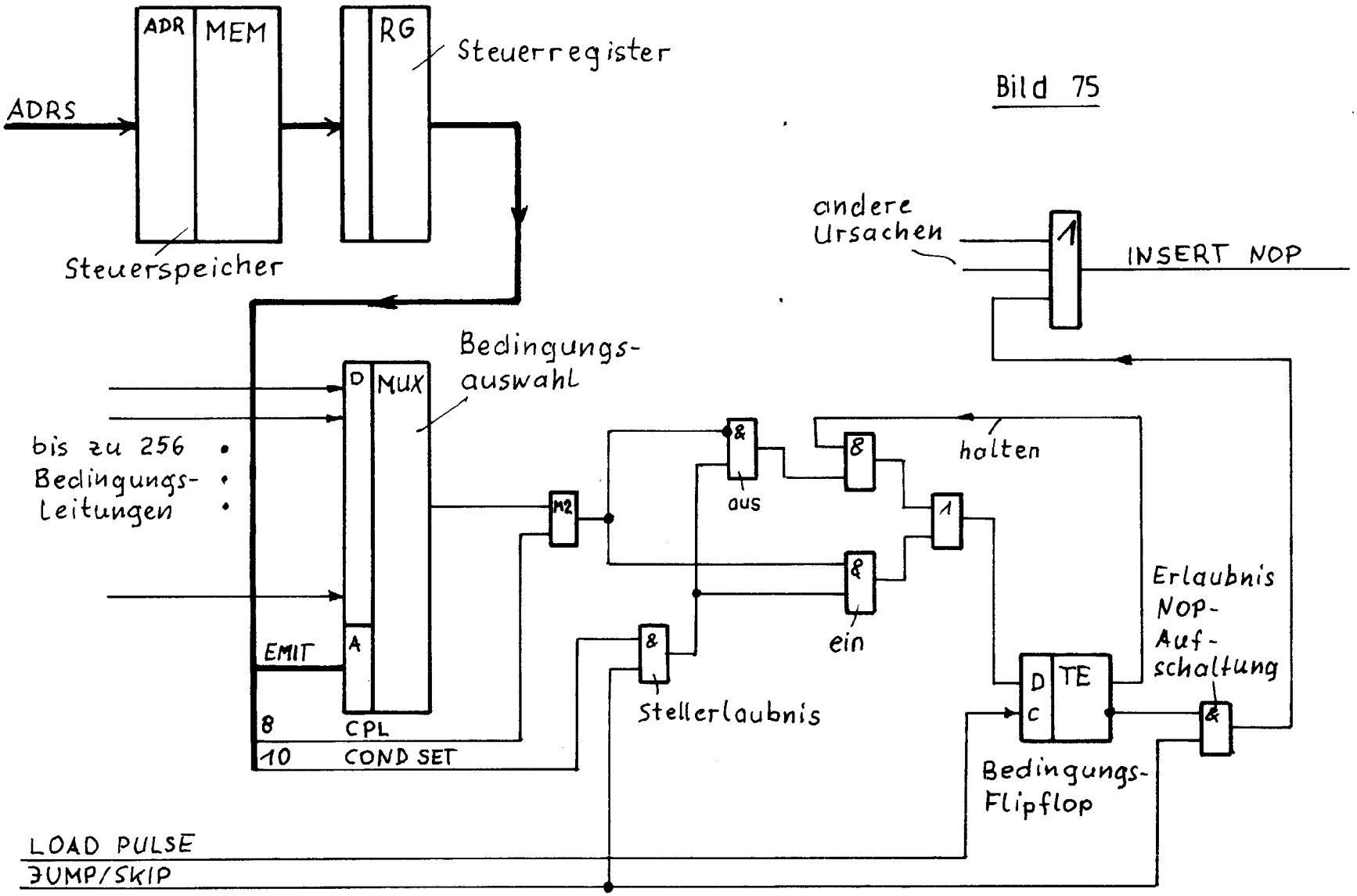
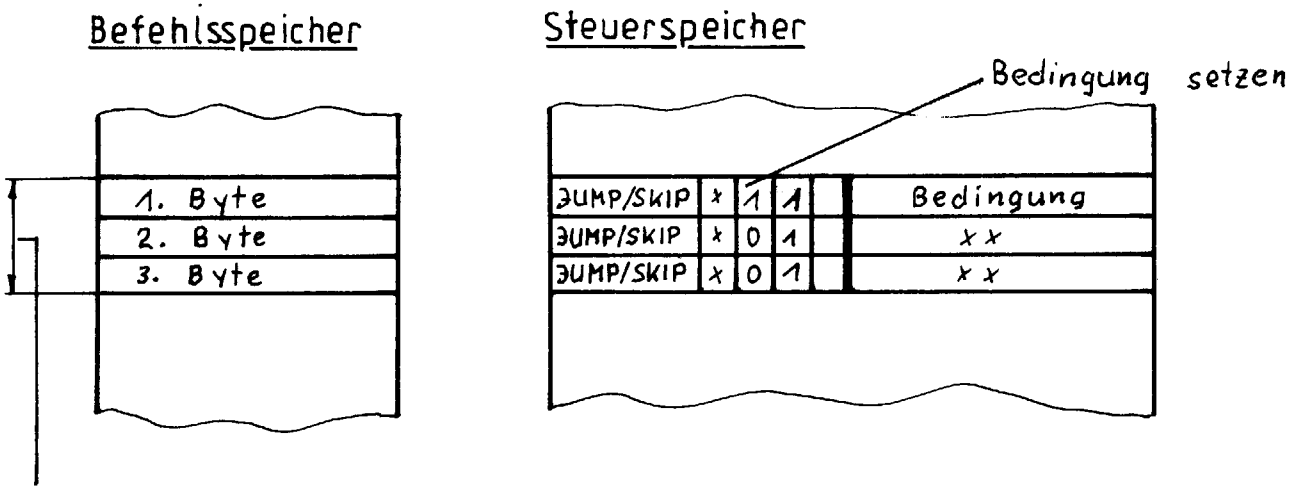

ADR
MEM
RG
Steuerregister
ADRS
Steuerspeicher
andere Ursachen
1
INSERT NOP
Bedingungs-auswahl
D
MUX
bis zu 256 Bedingungs-Leitungen
M2
&
aus
&
halten
1
&
ein
EMIT
A
&
Stellerlaubnis
Erlaubnis NOP-Auf-schaltung
D
TE
C
&
8
CPL
10
COND SET
Bedingungs-Flipflop
LOAD PULSE
JUMP/SKIP

Bild 75



Dieser Bereich wird entweder übergangen (an der CPU durch NOPs ersetzt) oder abgearbeitet.

Steuerwort

15 – 12	11	10	9	8	7 – 0
JUMP/SKIP	x	COND SET	INHIB ITRP	CPL	Bedingungsauswahl

Bild 76

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	NO-OP
0	1	S/D SELECT		ENABLE NMI	DIR	0	0	MOVE
1	0	SOURCE		ENABLE HI	ENABLE LO	0	0	INPUT
1	1	DESTINATION		EMIT				OUTPUT

Funktionsauswahl

Bild 77

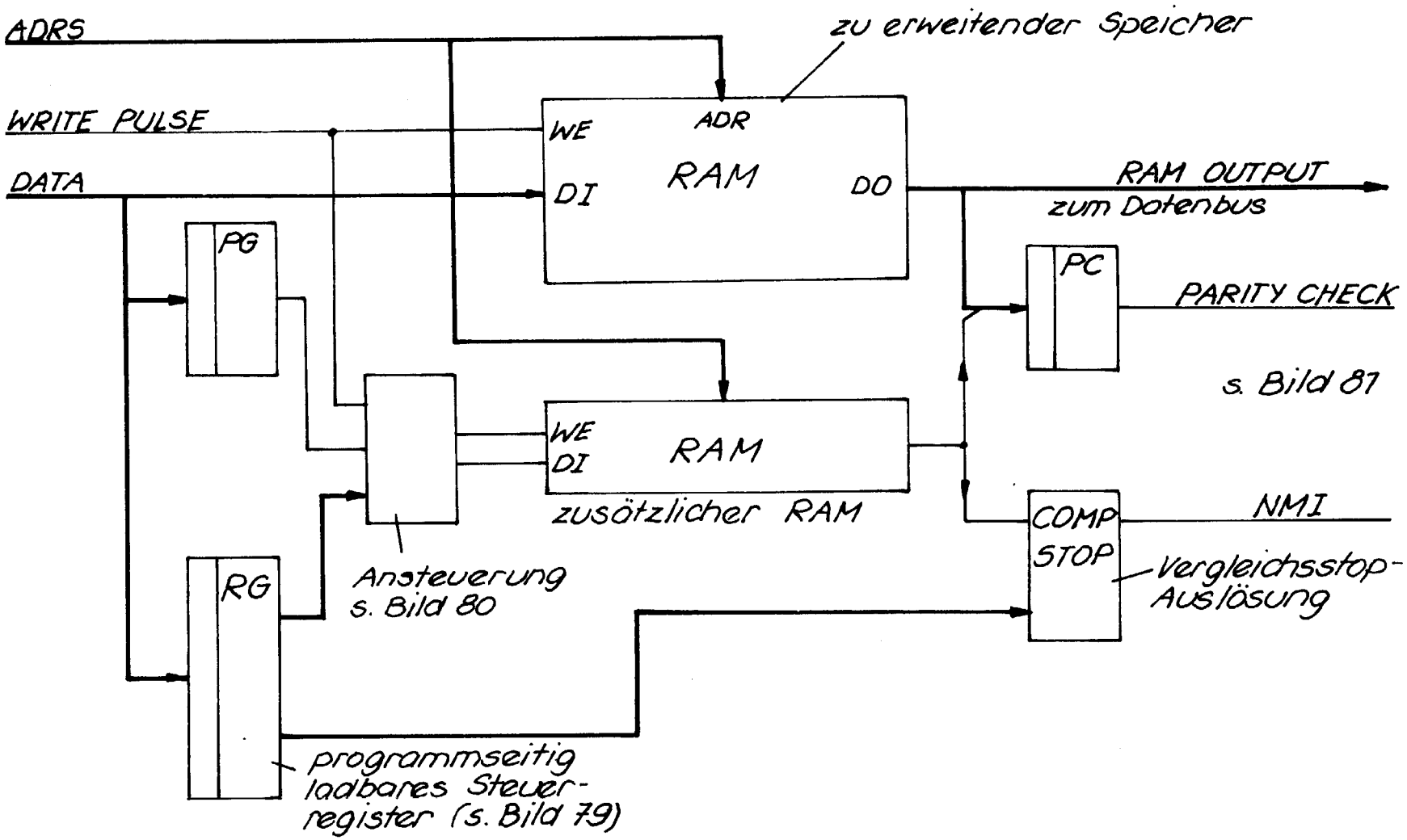

ADRS
WRITE PULSE
DATA
zu erweitender Speicher
WE
ADR
DI
RAM
DO
RAM OUTPUT
zum Datenbus
PG
PC
PARITY CHECK
s. Bild 87
WE
DI
RAM
zusätzlicher RAM
COMP
STOP
NMI
Vergleichsstop-
Auslösung
RG
Ansteuerung
s. Bild 80
programmseitig
ladbares Steuer-
register (s. Bild 79)

Bild 78

ENABLE PARITY	ENABLE COMPARE STOP	ENABLE INJECT	INJECT HI	STOP ON READ	STOP ON WRITE	ENABLE ERROR SIGNALIZATION

Bild 79

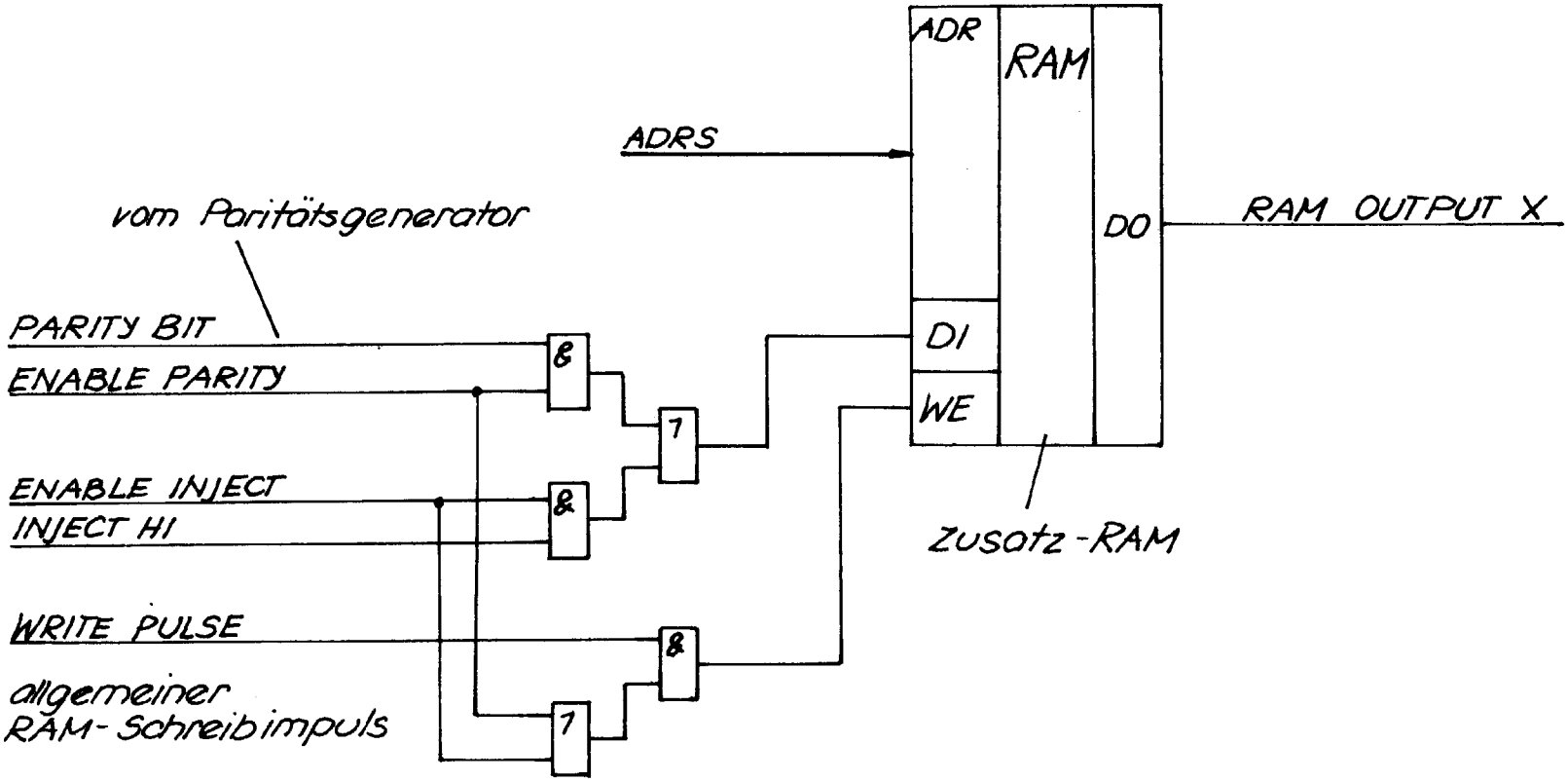

ADR
RAM
DO
ADRS
RAM OUTPUT X
vom Paritätsgenerator
PARITY BIT
ENABLE PARITY
DI
WE
&
1
ENABLE INJECT
INJECT HI
WRITE PULSE
allgemeiner
RAM-Schreibimpuls
Zusatz-RAM

Bild 80

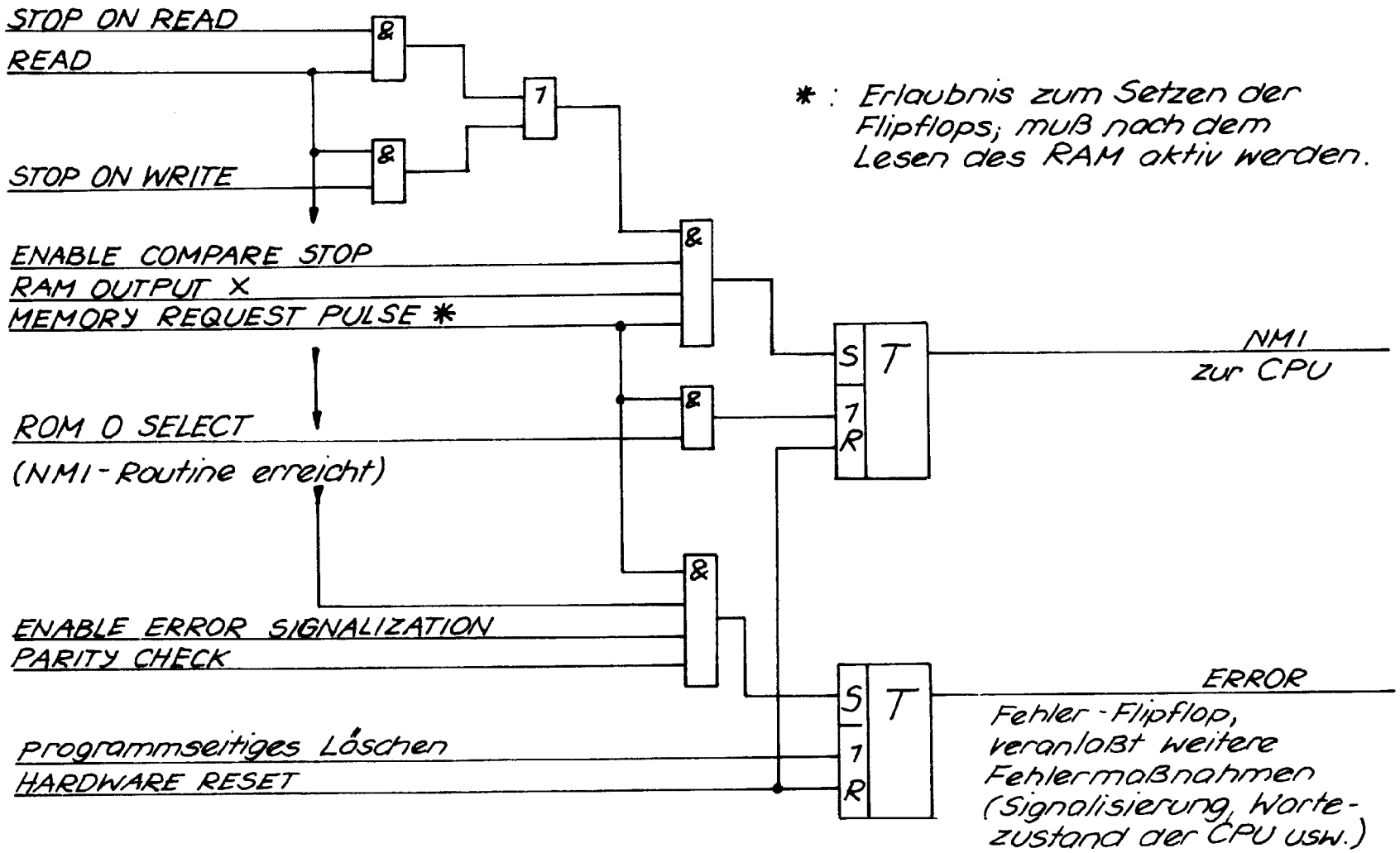

STOP ON READ
READ
STOP ON WRITE
&
&
1
ENABLE COMPARE STOP
RAM OUTPUT X
MEMORY REQUEST PULSE *
&
ROM 0 SELECT
(NMI-Routine erreicht)
&
ENABLE ERROR SIGNALIZATION
PARITY CHECK
&
programmseitiges Löschen
HARDWARE RESET
S
1
R
T
NMI
zur CPU
S
1
R
T
ERROR
Fehler-Flipflop,
veranlaßt weitere
Fehlermaßnahmen
(Signalisierung, Worte-
zustand der CPU usw.)
* : Erlaubnis zum Setzen der
Flipflops; muß nach dem
Lesen des RAM aktiv werden.

Bild 81

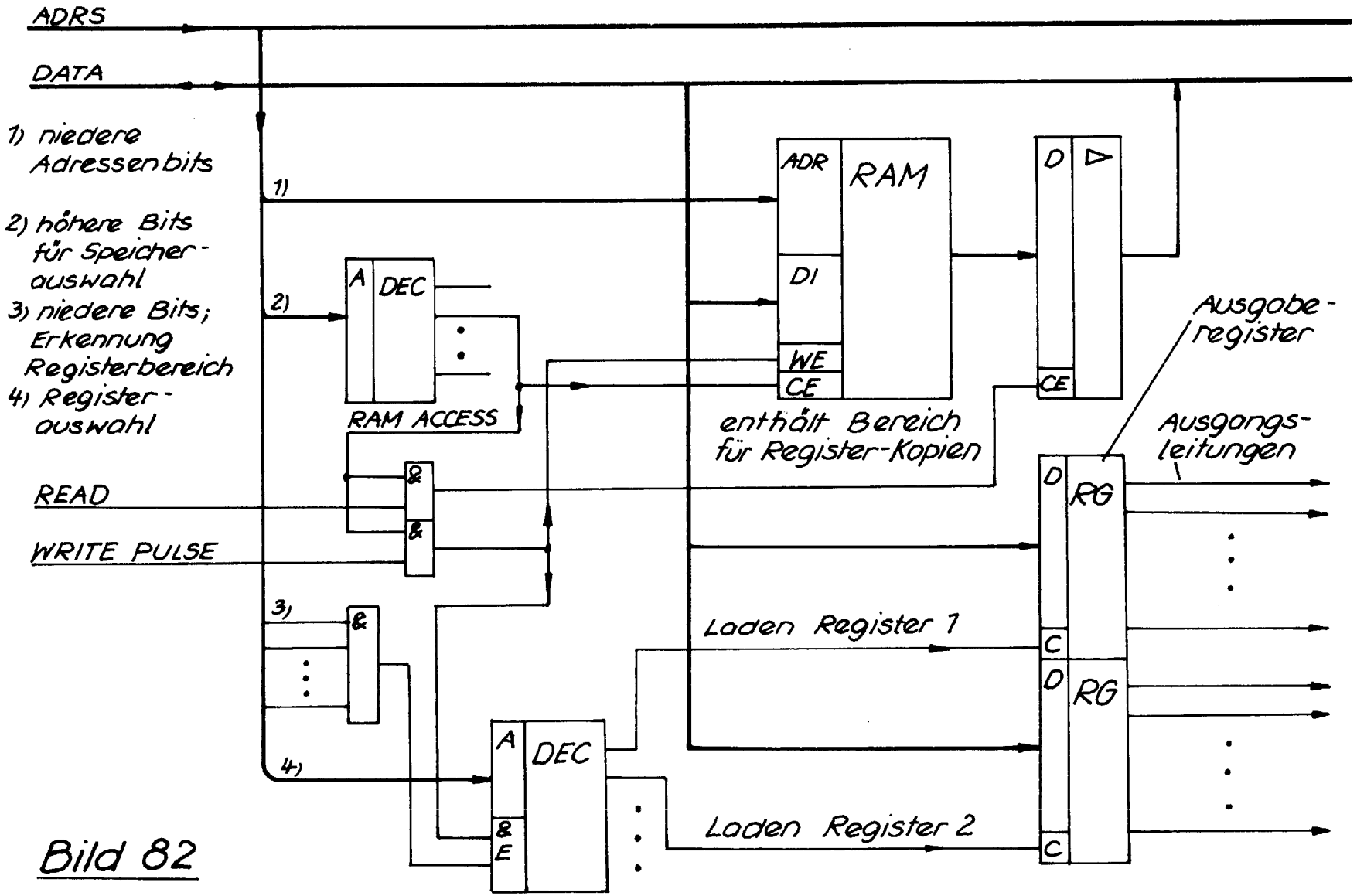

ADRS
DATA
1) niedere Adressenbits
2) höhere Bits für Speicher-auswahl
3) niedere Bits; Erkennung Registerbereich
4) Register-auswahl
1)
2)
3)
4)
A
DEC
RAM ACCESS
READ
WRITE PULSE
&
ADR
RAM
DI
WE
CE
enthält Bereich für Register-Kopien
D
CE
Ausgabe-register
Ausgangs-leitungen
RG
C
Laden Register 1
Laden Register 2
E

Bild 82

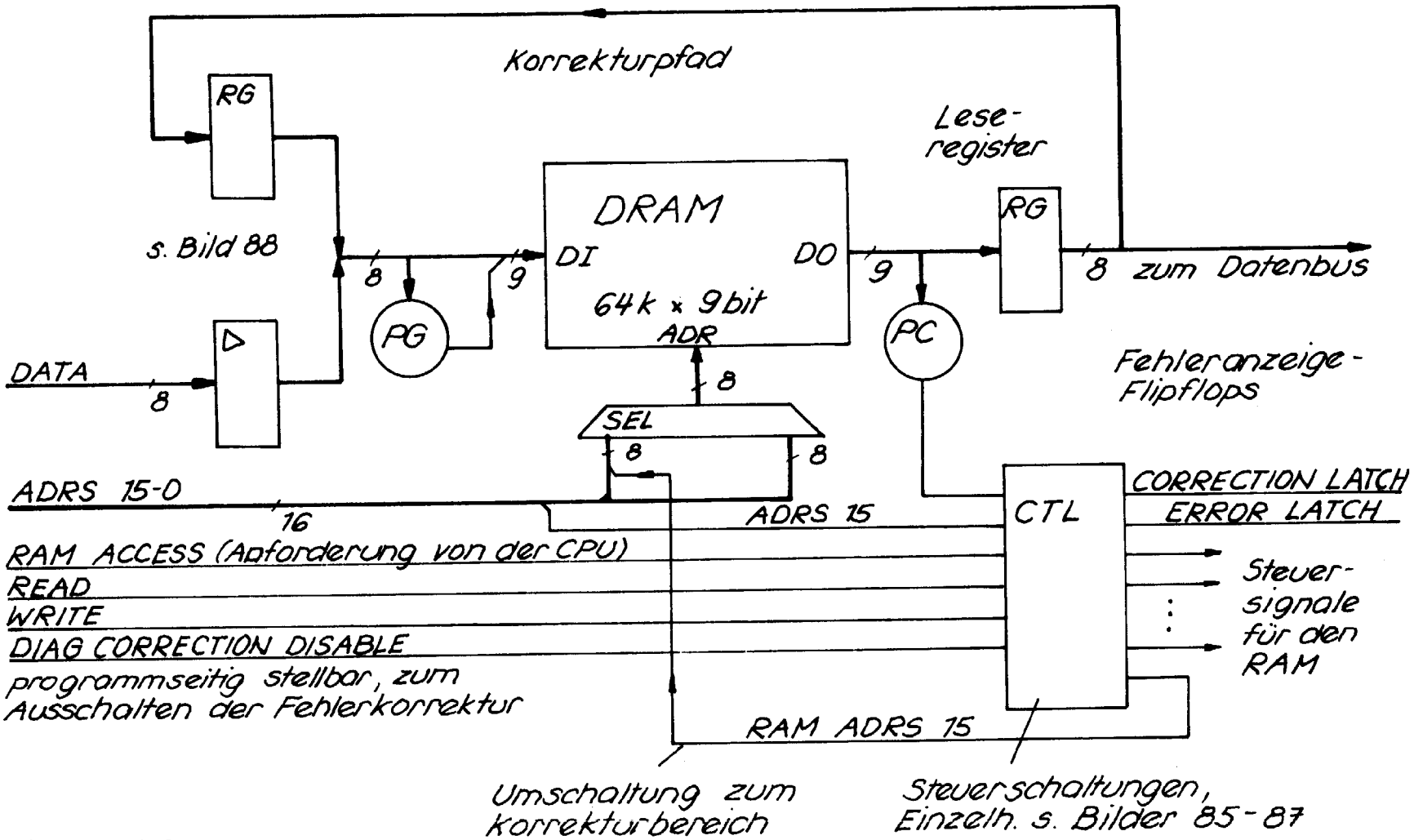
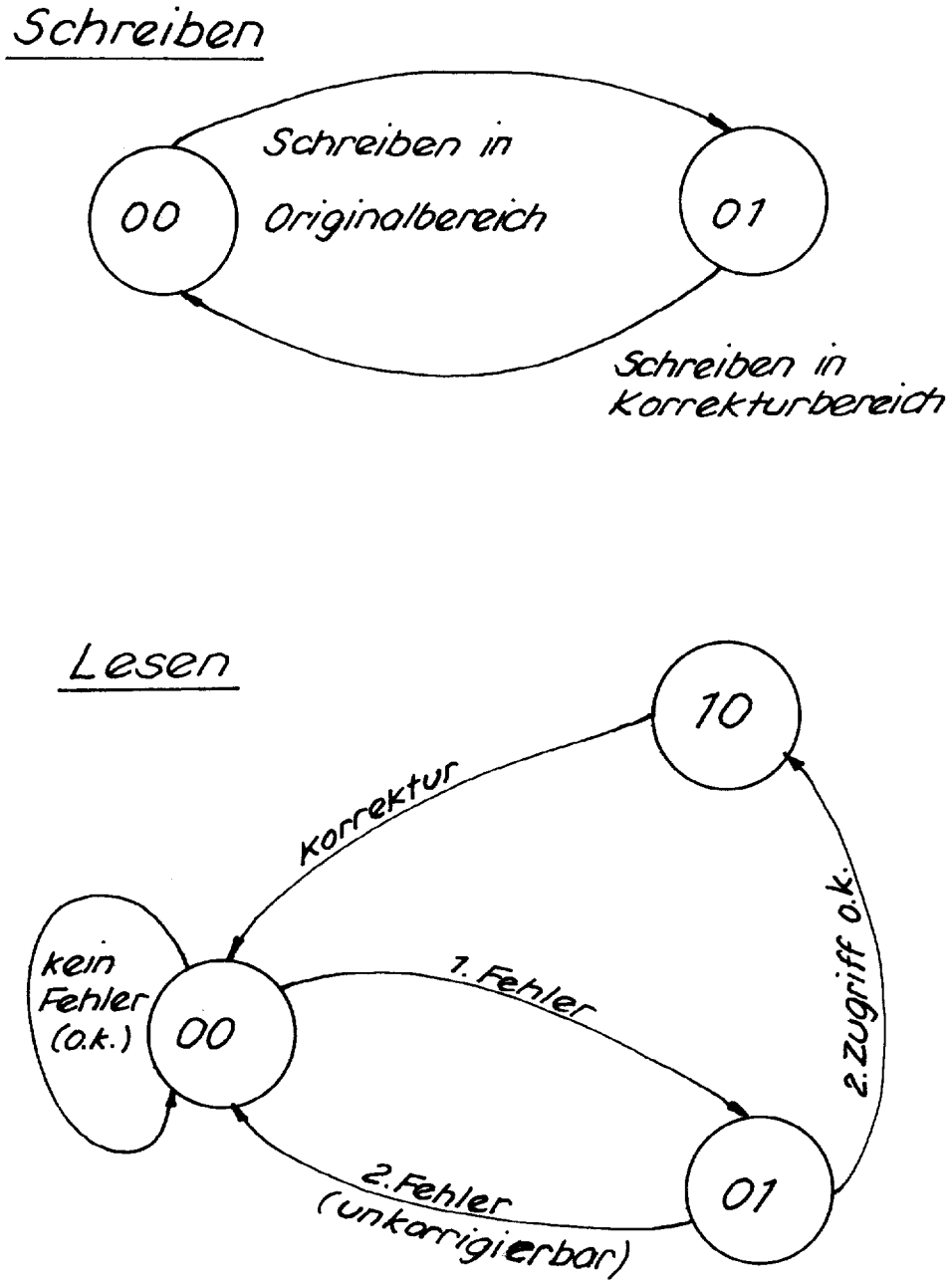

Korrekturpfad
RG
s. Bild 88
DATA
8
8
9
PG
DRAM
DI
DO
64k x 9bit
ADR
9
PC
Lese-
register
RG
8
zum Datenbus
Fehleranzeige-
Flipflops
8
SEL
8
8
ADRS 15-0
16
ADRS 15
CTL
CORRECTION LATCH
ERROR LATCH
RAM ACCESS (Anforderung von der CPU)
READ
WRITE
DIAG CORRECTION DISABLE
programmseitig stellbar, zum
Ausschalten der Fehlerkorrektur
Steuer-
signale
für den
RAM
RAM ADRS 15
Umschaltung zum
Korrekturbereich
Steuerschaltungen,
Einzeln. s. Bilder 85-87

Bild 83



In den Knoten sind die Belegungen der Flipflops STAT 0,1 (Bild 87) angegeben.

Bild 84

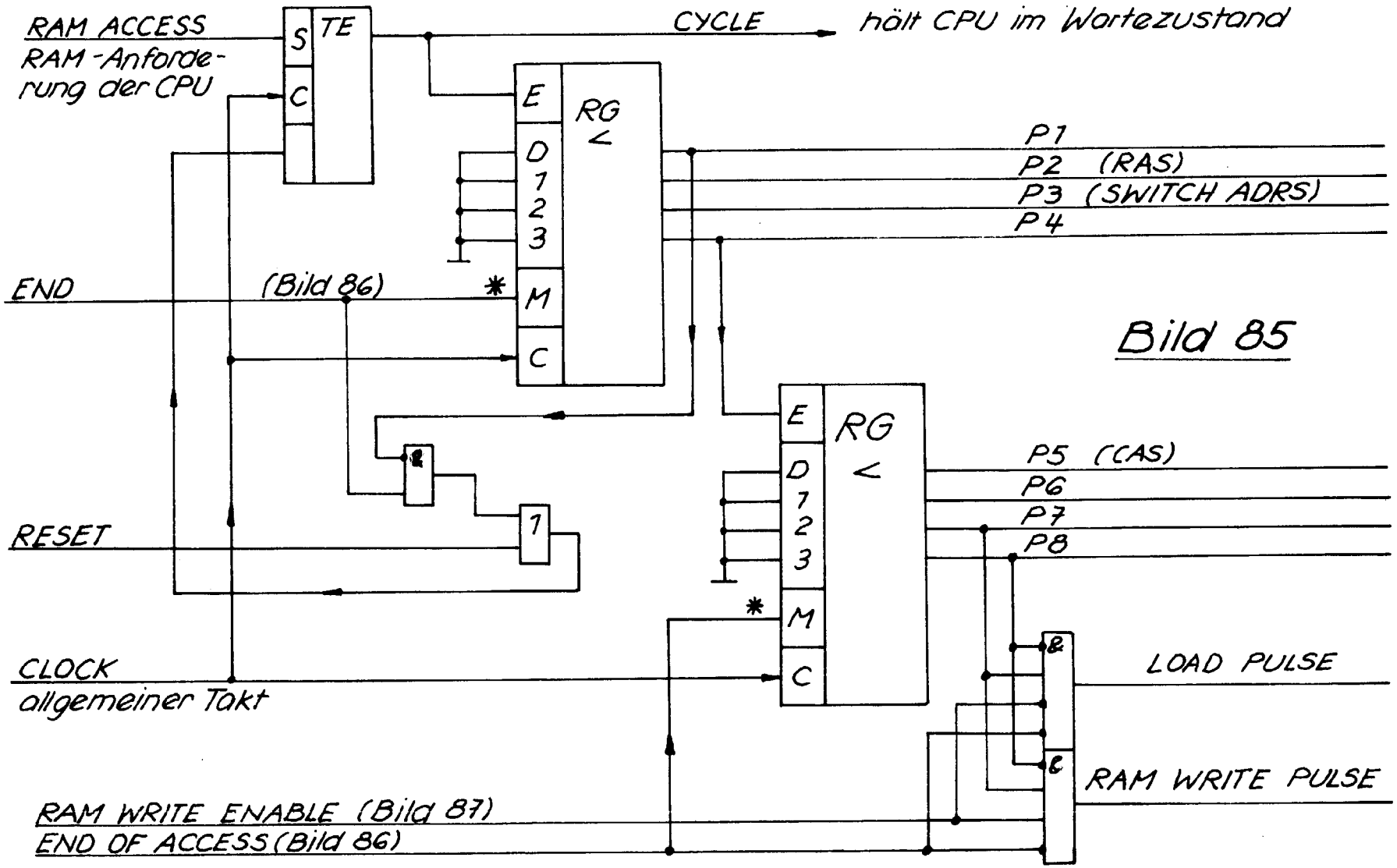


Bild 85

*: "1" ≙ Parallelübernahme

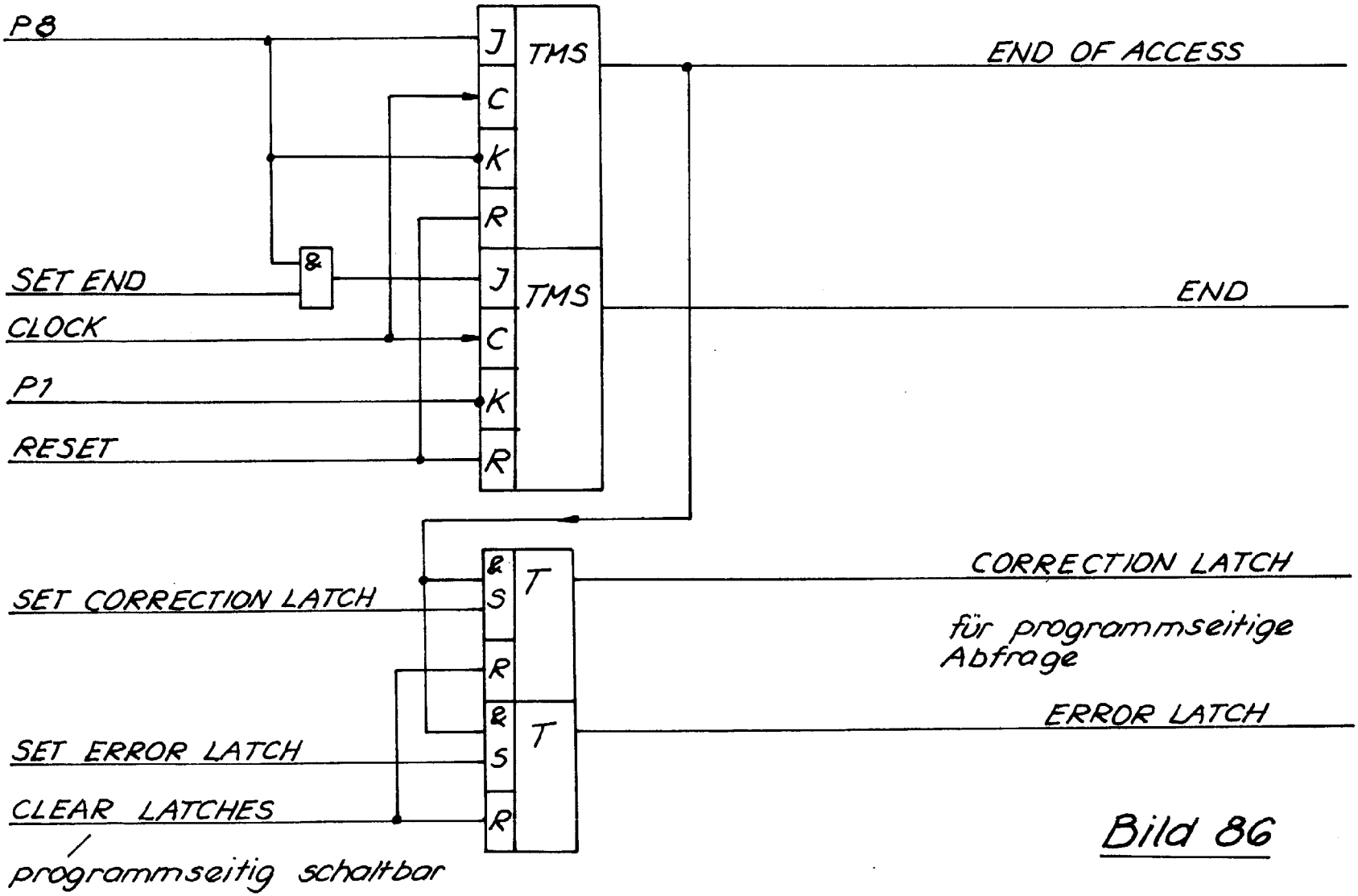
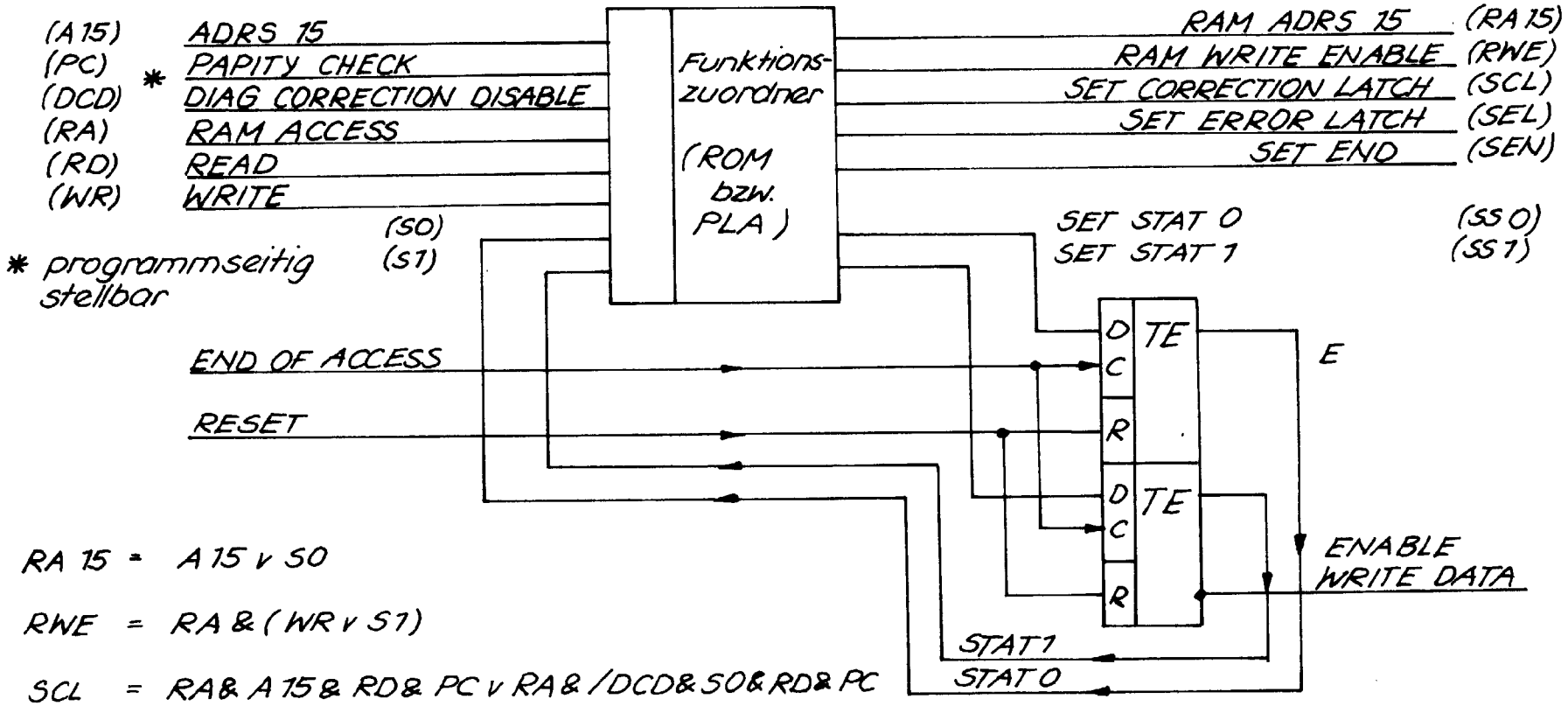

P8
SET END
CLOCK
P1
RESET
J
C
K
R
TMS
&
J
C
K
R
TMS
END OF ACCESS
END
SET CORRECTION LATCH
SET ERROR LATCH
CLEAR LATCHES
programmseitig schaltbar
&
S
R
T
&
S
R
T
CORRECTION LATCH
für programmseitige Abfrage
ERROR LATCH

Bild 86



RA 15 = A 15 v S0

RWE = RA & (WR v S1)

SCL = RA& A 15& RD& PC v RA& /DCD& S0& RD& PC

SEL = RA& DCD& RD& PC v RA& RD& S0& PC

SEN = RA& (DCD v A 15 v WR& S0 v RD& PC& S0 v S1)

SS0 = (RA& /DCD& /A 15& /S0& /S1)& (RD& PC v WR)

SS1 = S0& RA& RD& /PC& /S1

Bild 87

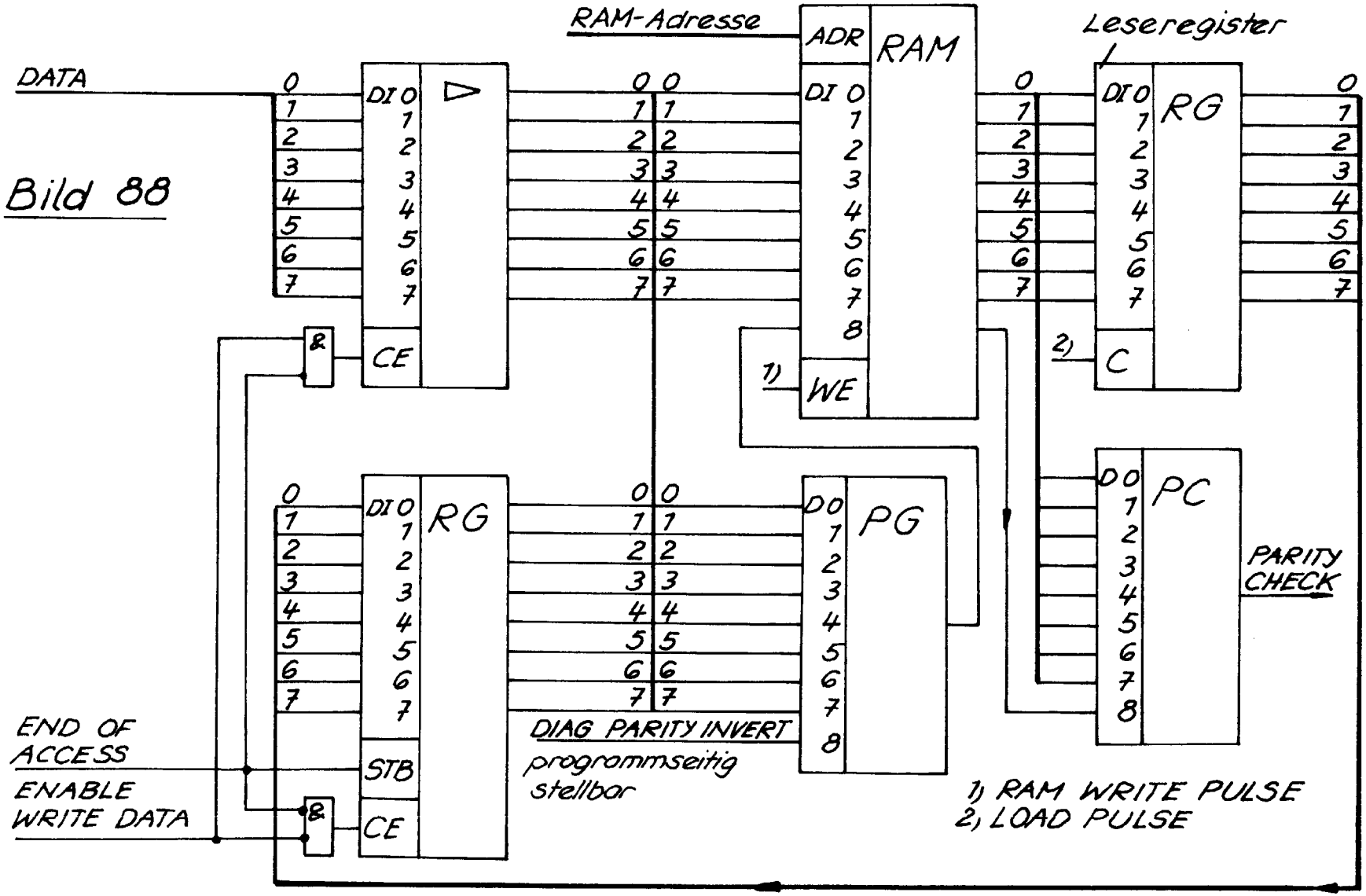

DATA
RAM-Adresse
ADR
RAM
Leseregister
DI
RG
CE
WE
C
PC
PG
STB
PARITY CHECK
END OF ACCESS
ENABLE WRITE DATA
DIAG PARITY INVERT
programmseitig stellbar
1) RAM WRITE PULSE
2) LOAD PULSE

Bild 88

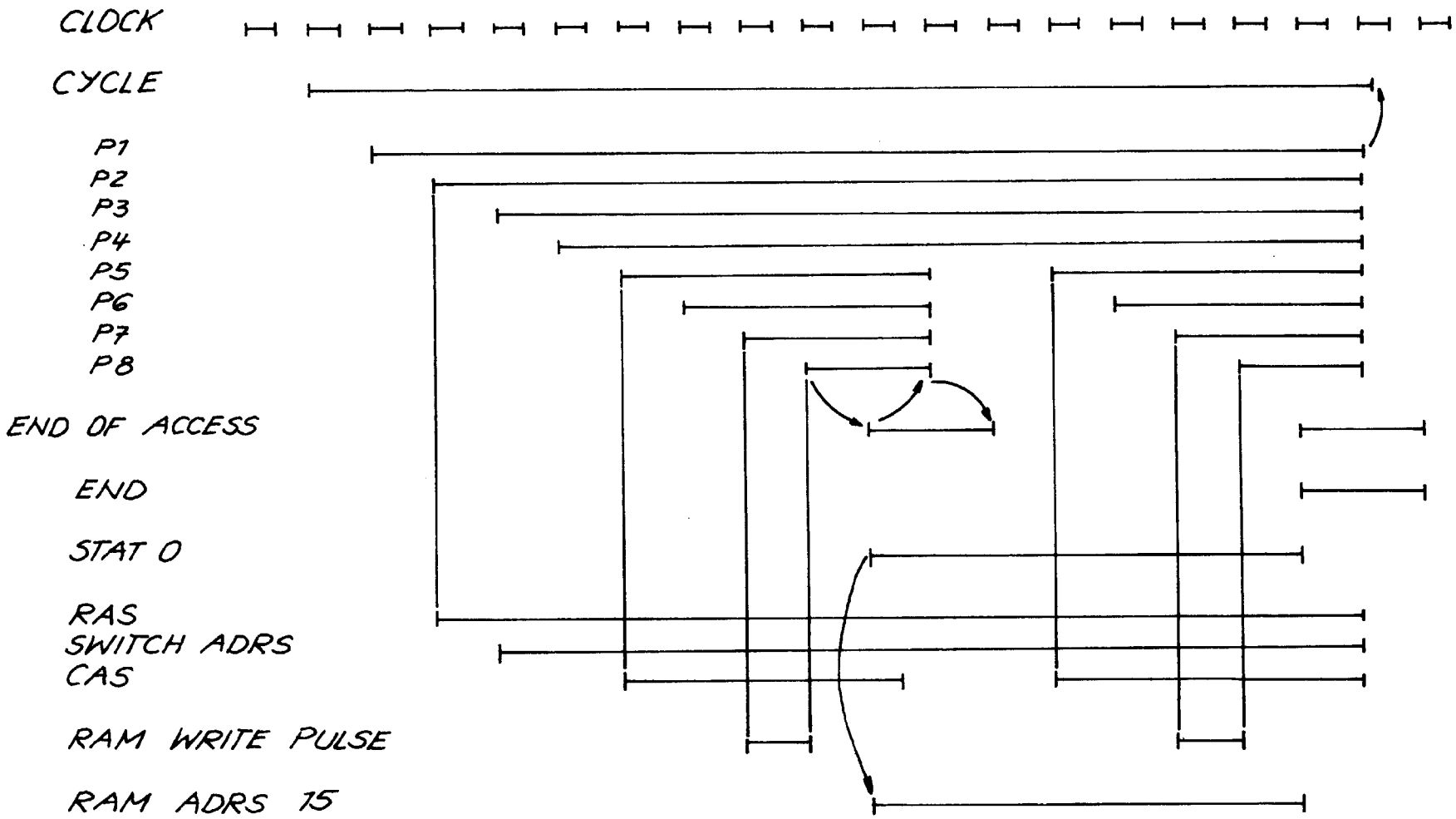

CLOCK
CYCLE
P1
P2
P3
P4
P5
P6
P7
P8
END OF ACCESS
END
STAT 0
RAS
SWITCH ADRS
CAS
RAM WRITE PULSE
RAM ADRS 15

Bild 89

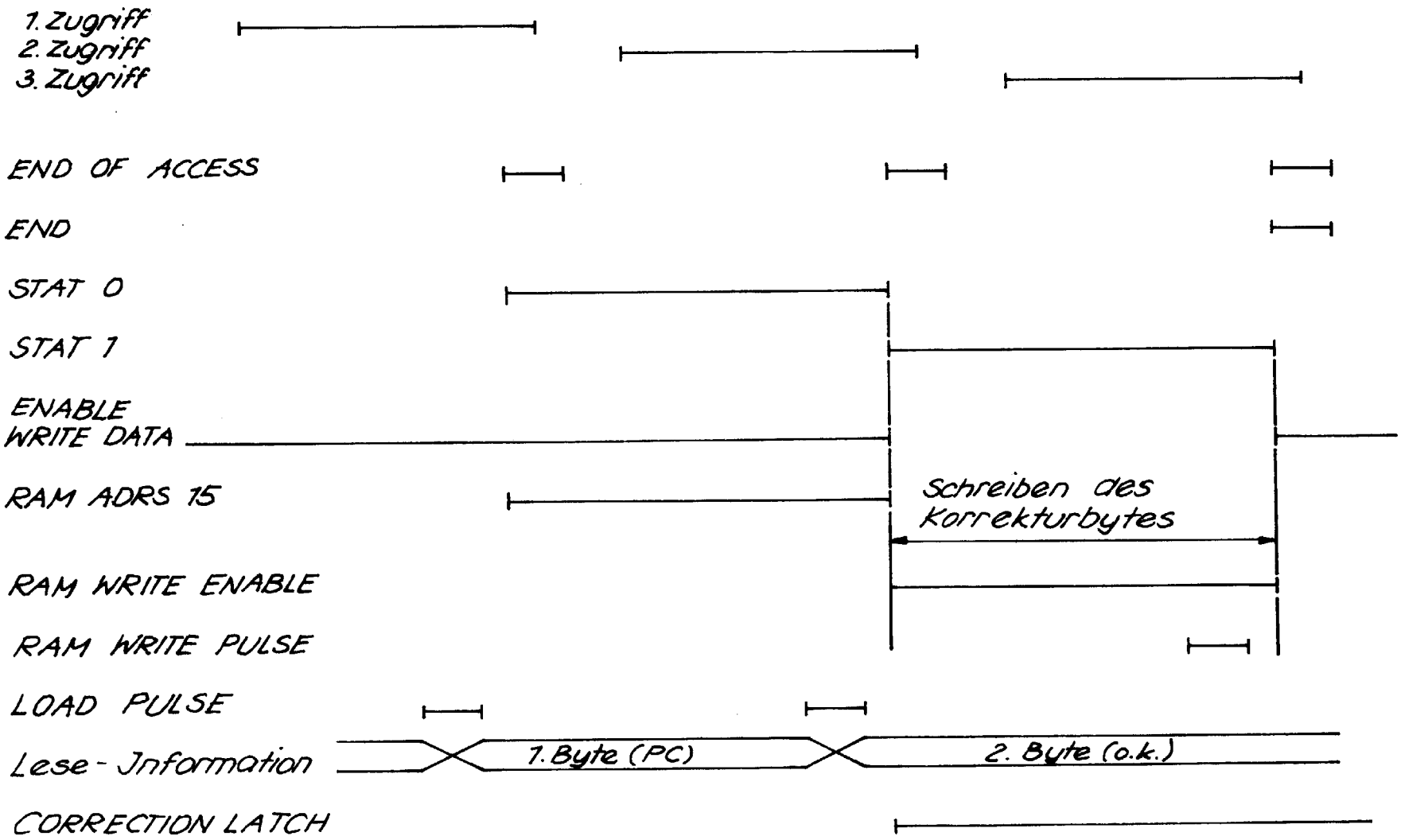

1. Zugriff
2. Zugriff
3. Zugriff
END OF ACCESS
END
STAT 0
STAT 1
ENABLE WRITE DATA
RAM ADRS 15
Schreiben des Korrekturbytes
RAM WRITE ENABLE
RAM WRITE PULSE
LOAD PULSE
Lese-Information
1. Byte (PC)
2. Byte (o.k.)
CORRECTION LATCH

Bild 90

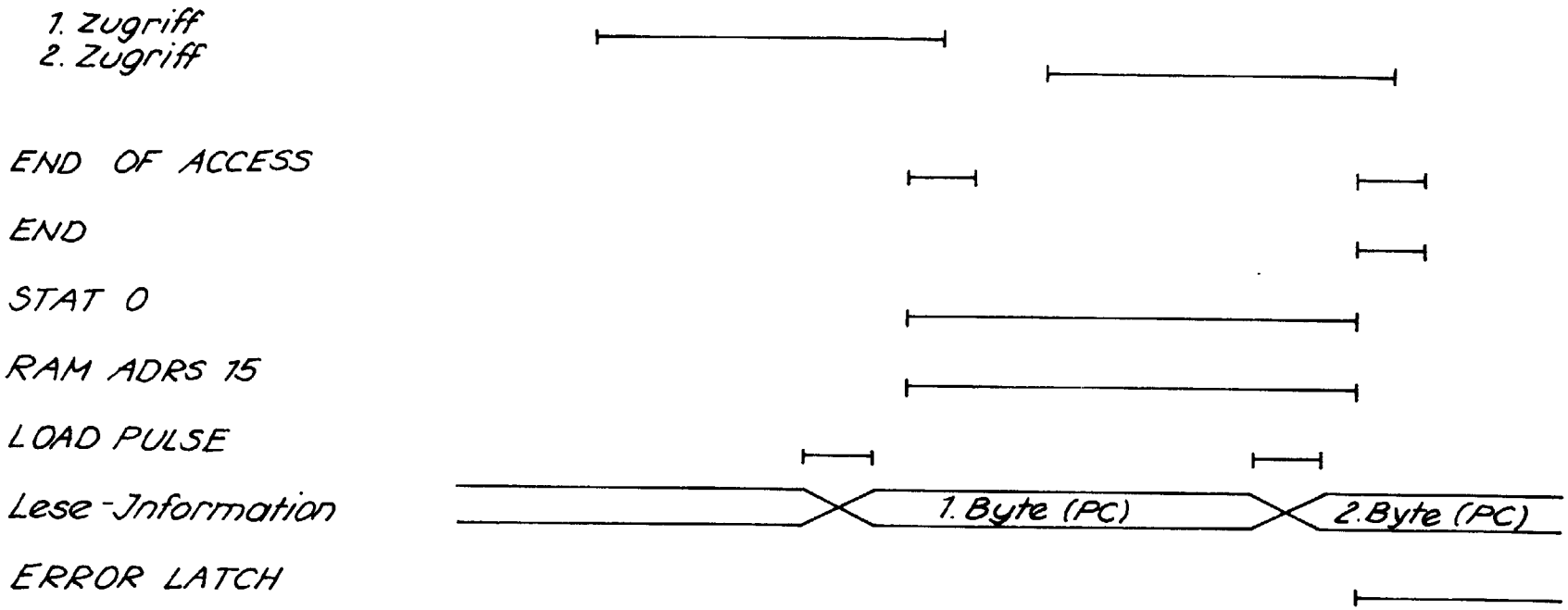

1. Zugriff
2. Zugriff
END OF ACCESS
END
STAT 0
RAM ADRS 15
LOAD PULSE
Lese-Information
1. Byte (PC)
2. Byte (PC)
ERROR LATCH

Bild 91

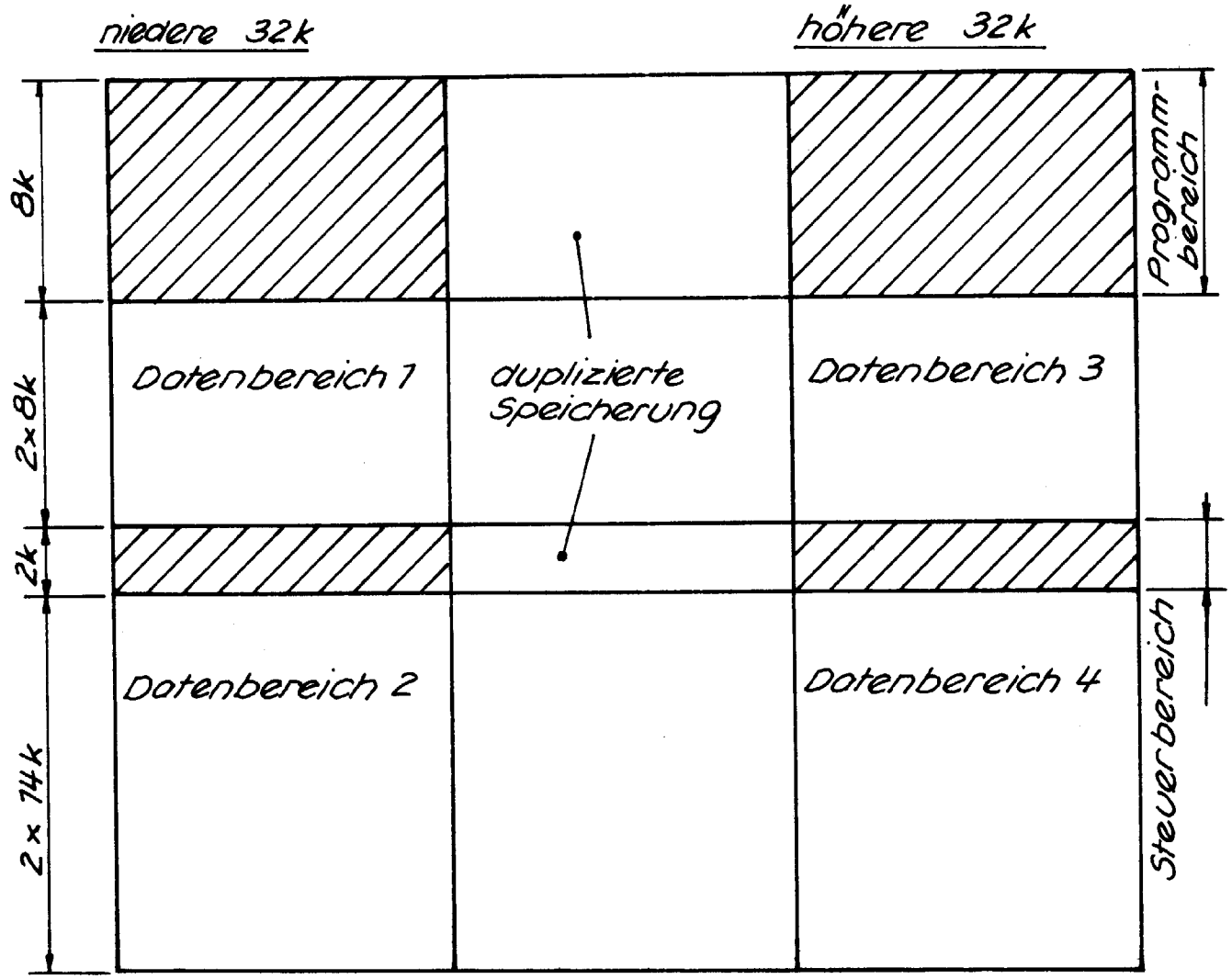


Bild 92 Beispiel für Aufteilung eines 64k-RAM

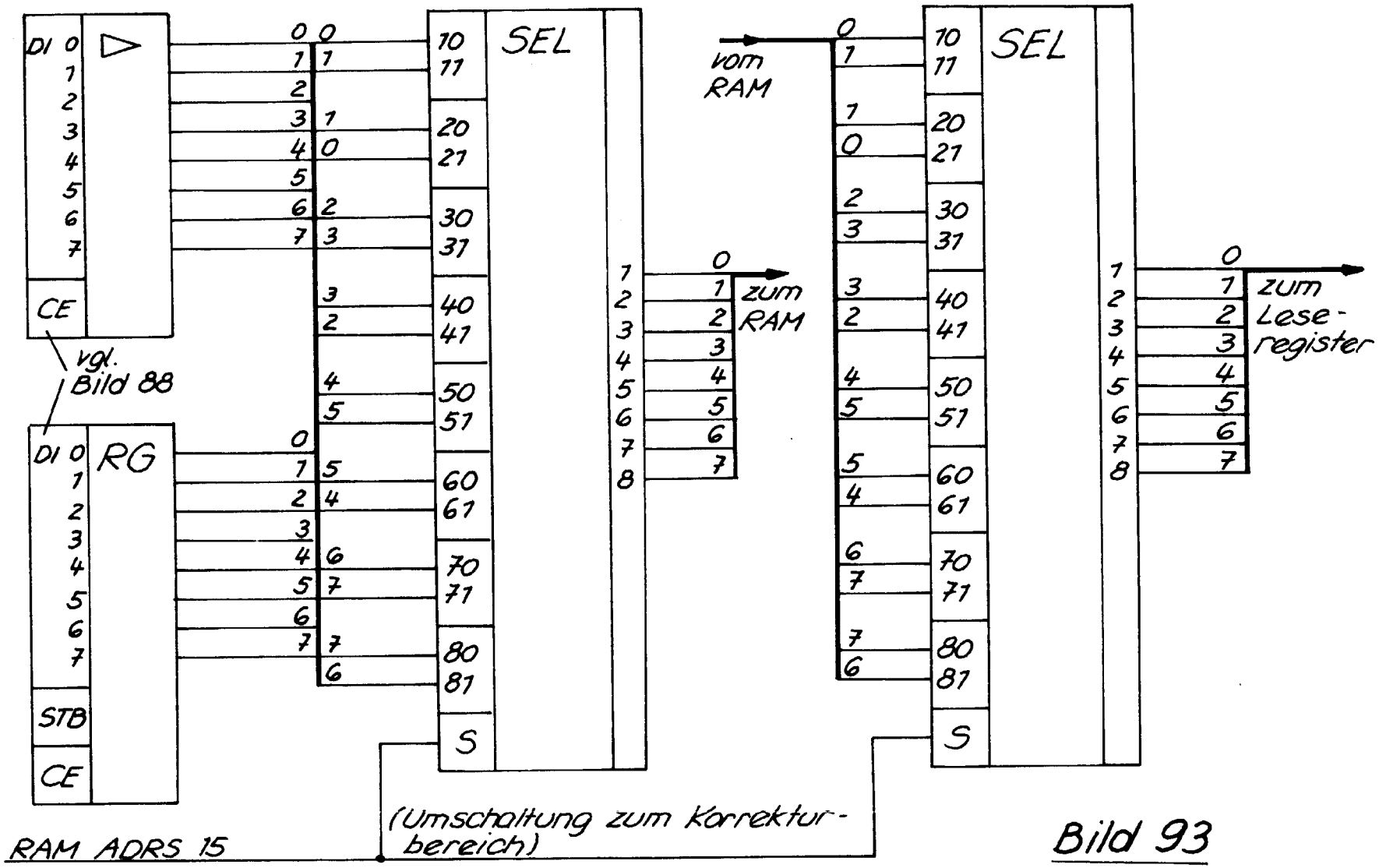

DI
CE
vgl. Bild 88
DI
RG
STB
CE
SEL
S
vom RAM
zum RAM
SEL
S
zum Lese-register
(Umschaltung zum Korrektur-bereich)
RAM ADRS 15

Bild 93

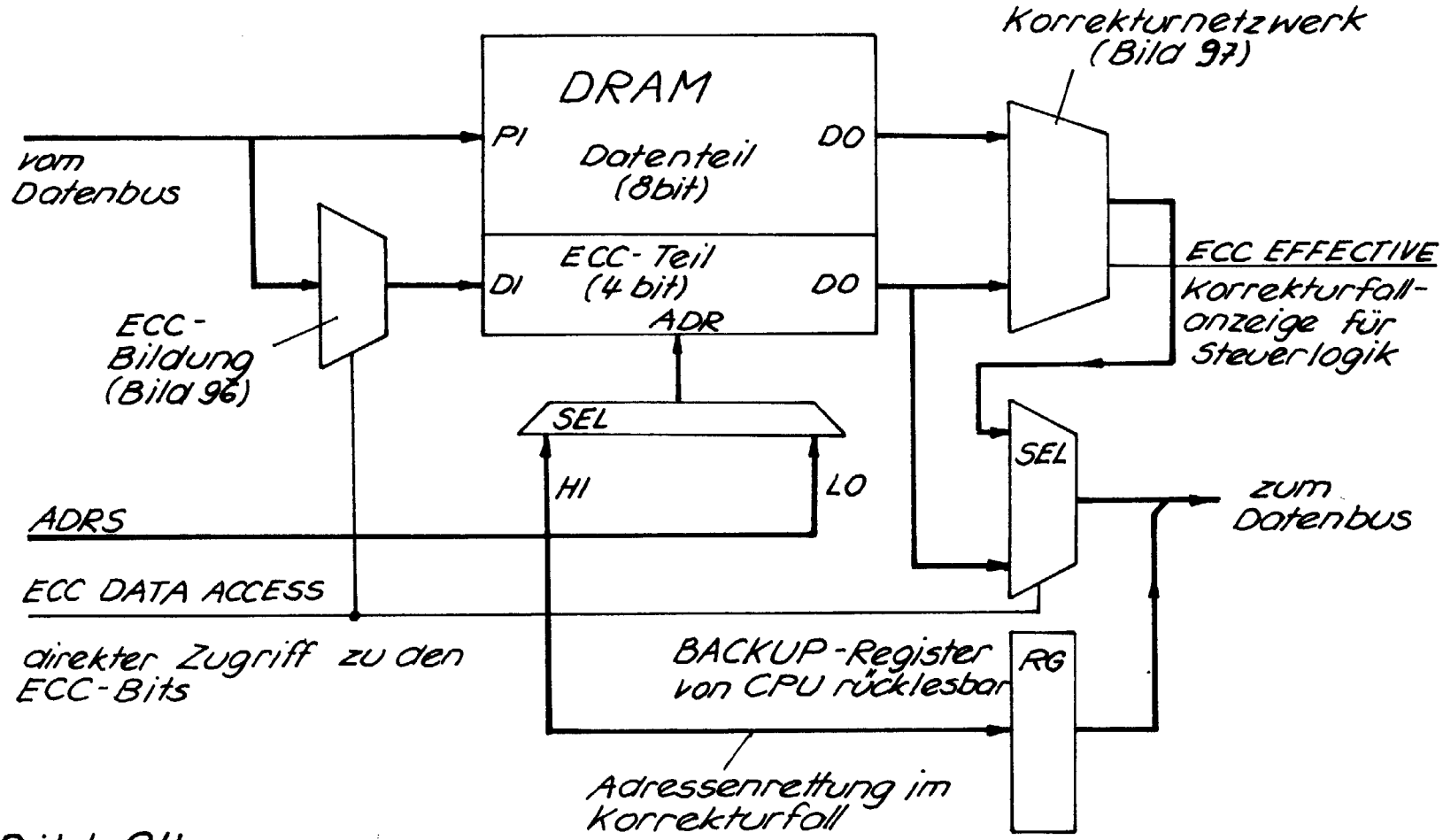

Korrekturnetzwerk
(Bild 97)
DRAM
PI
Datenteil
(8bit)
DO
vom
Datenbus
ECC-Teil
(4 bit)
DI
DO
ADR
ECC EFFECTIVE
Korrekturfall-
anzeige für
Steuerlogik
ECC-
Bildung
(Bild 96)
SEL
HI
LO
SEL
zum
Datenbus
ADRS
ECC DATA ACCESS
direkter Zugriff zu den
ECC-Bits
BACKUP-Register
von CPU rücklesbar
RG
Adressenrettung im
Korrekturfall

Bild 94

ECC-Bits				Datenbits								Paritätsprüfer
11	10	9	8	7	6	5	4	3	2	1	0	
			×	×	×		×		×		×	A (2^0)
		×	×		×	×			×	×		B (2^1)
	×	×	×		×	×	×	×				C (2^2)
×	×	×	×	×								D (2^3)
$AB\bar{C}D$	$AB\bar{C}\bar{D}$	$A\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}BC\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$A\bar{B}\bar{C}D$	$\bar{A}B\bar{C}D$	$AB\bar{C}D$	$\bar{A}\bar{B}CD$	$A\bar{B}CD$	$\bar{A}BCD$	Fehler in Bit [1]
7	3	1	0	6	8	9	A	B	C	D	E	korrekte Parität [2]
					0	1	2	3	4	5	6	decodierte Belegung [3]

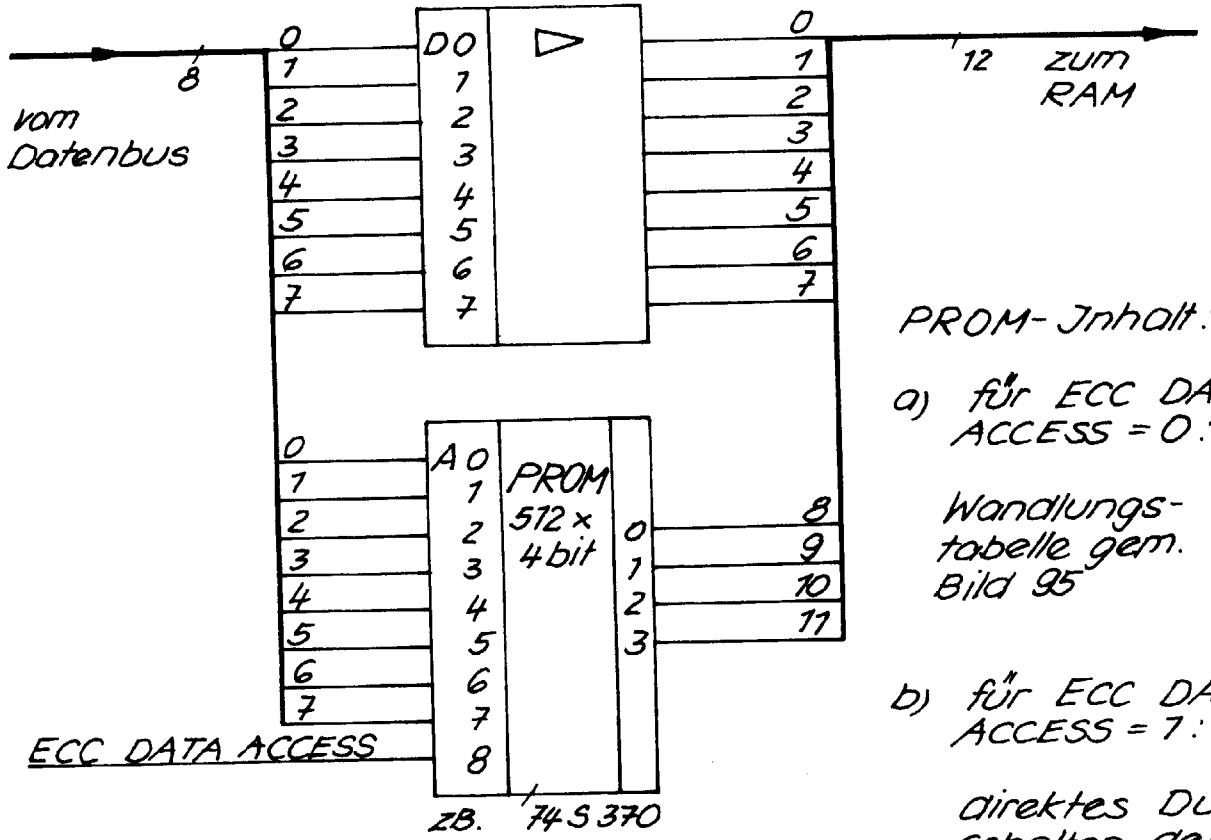
× : Bit in Paritätskontrolle einbezogen

1) : Negation kennzeichnet inkorrekte (gerade) Parität

2) : HEX-Darstellung der Paritätsprüfer-Ausgänge D... A

3) : Decodierung der Paritätsprüfer-Ausgänge C,B,A (mit D=1)

Bild 95



PROM-Inhalt:

a) für ECC DATA ACCESS = 0:

Wandlungstabelle gem. Bild 95

b) für ECC DATA ACCESS = 1:

direktes Durchschalten der Datenbits 3...0

Bild 96

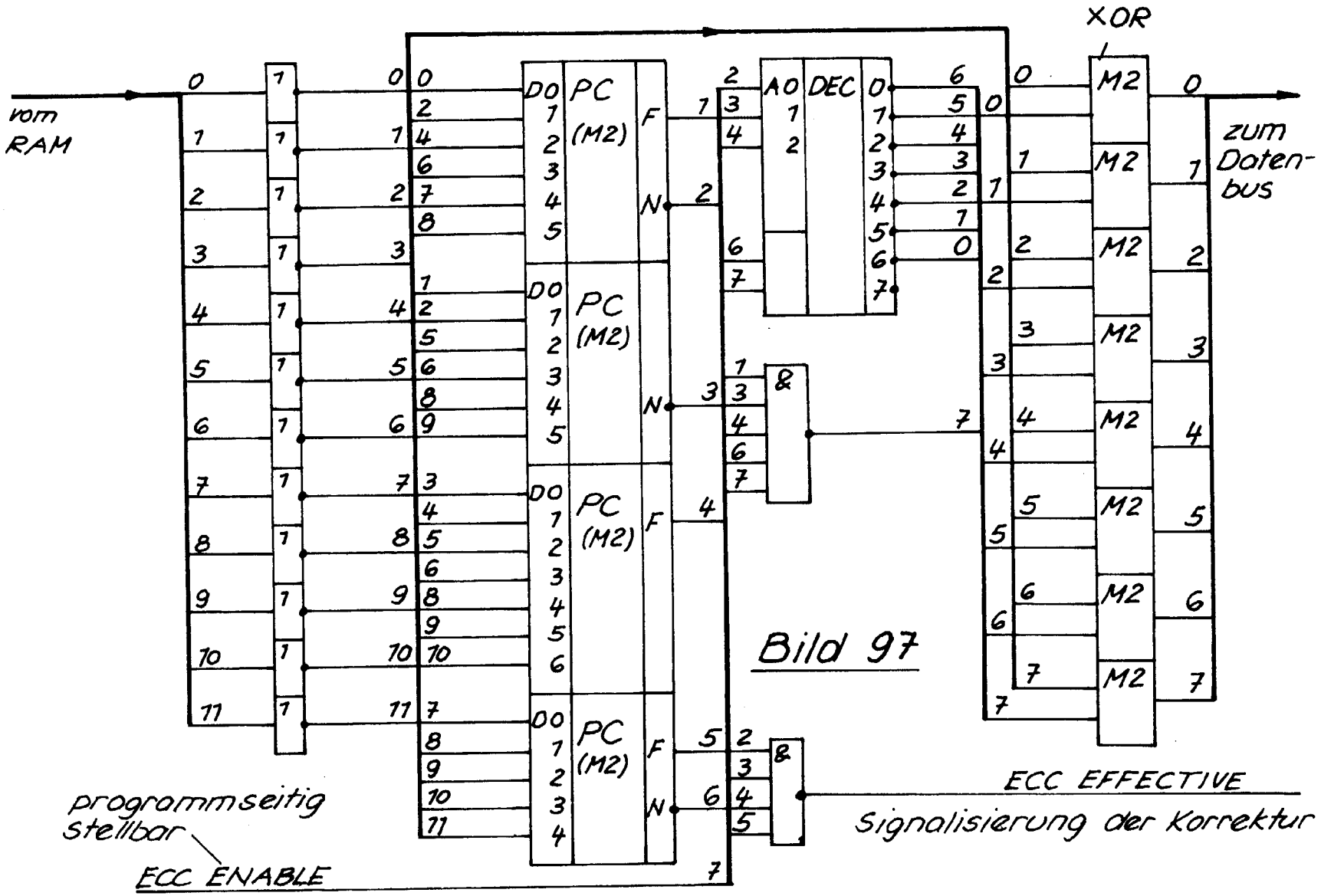

vom RAM
PC (M2)
DEC
XOR
M2
zum Datenbus
programmseitig stellbar
ECC ENABLE
ECC EFFECTIVE
Signalisierung der Korrektur

Bild 97

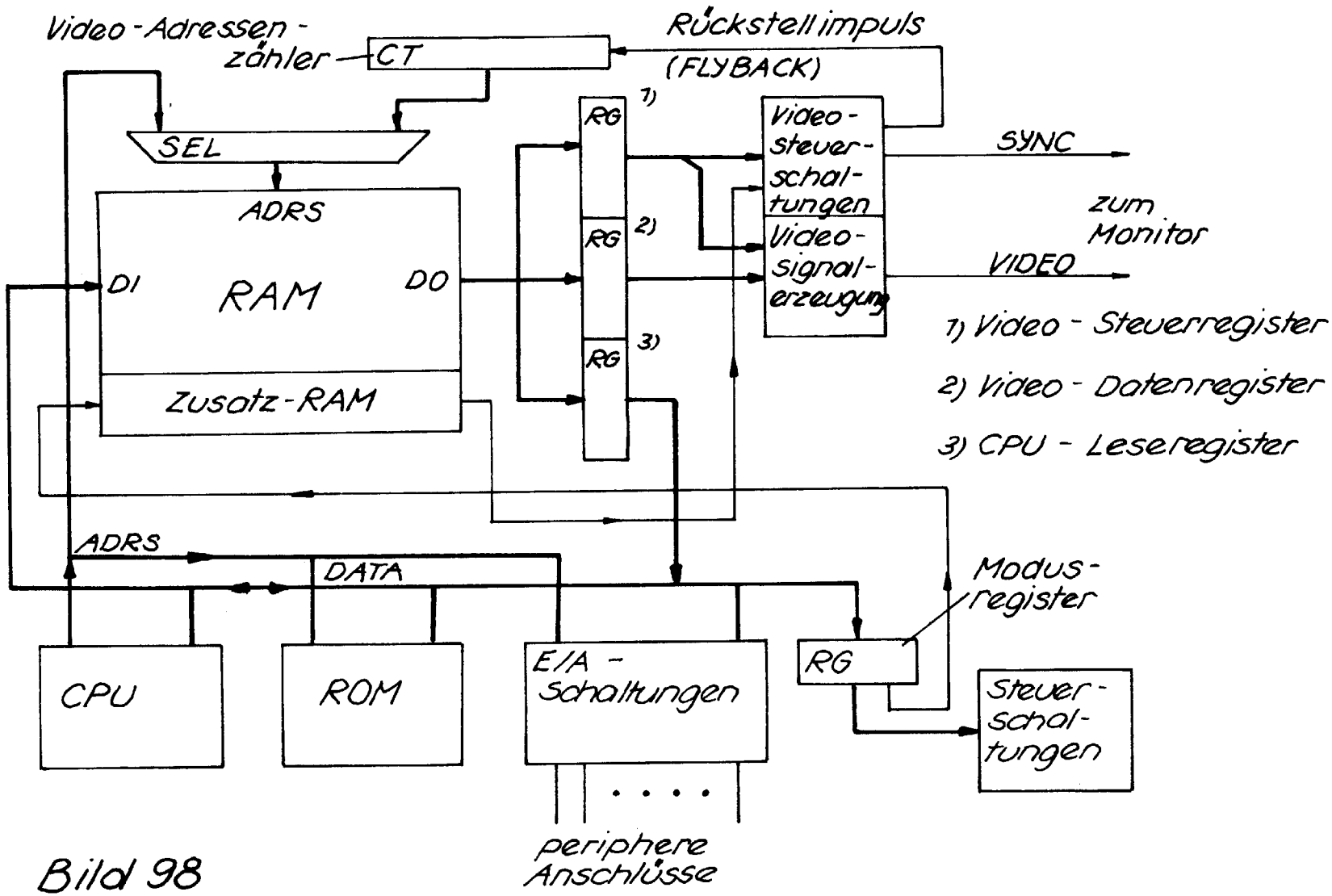

Video-Adressen-zähler
CT
Rückstellimpuls (FLYBACK)
SEL
ADRS
DI
RAM
DO
Zusatz-RAM
RG
1)
RG
2)
RG
3)
Video-steuer-schal-tungen
Video-signal-erzeugung
SYNC
zum Monitor
VIDEO
1) Video - Steuerregister
2) Video - Datenregister
3) CPU - Leseregister
ADRS
DATA
Modus-register
CPU
ROM
E/A - Schaltungen
RG
Steuer-schal-tungen
periphere Anschlüsse

Bild 98

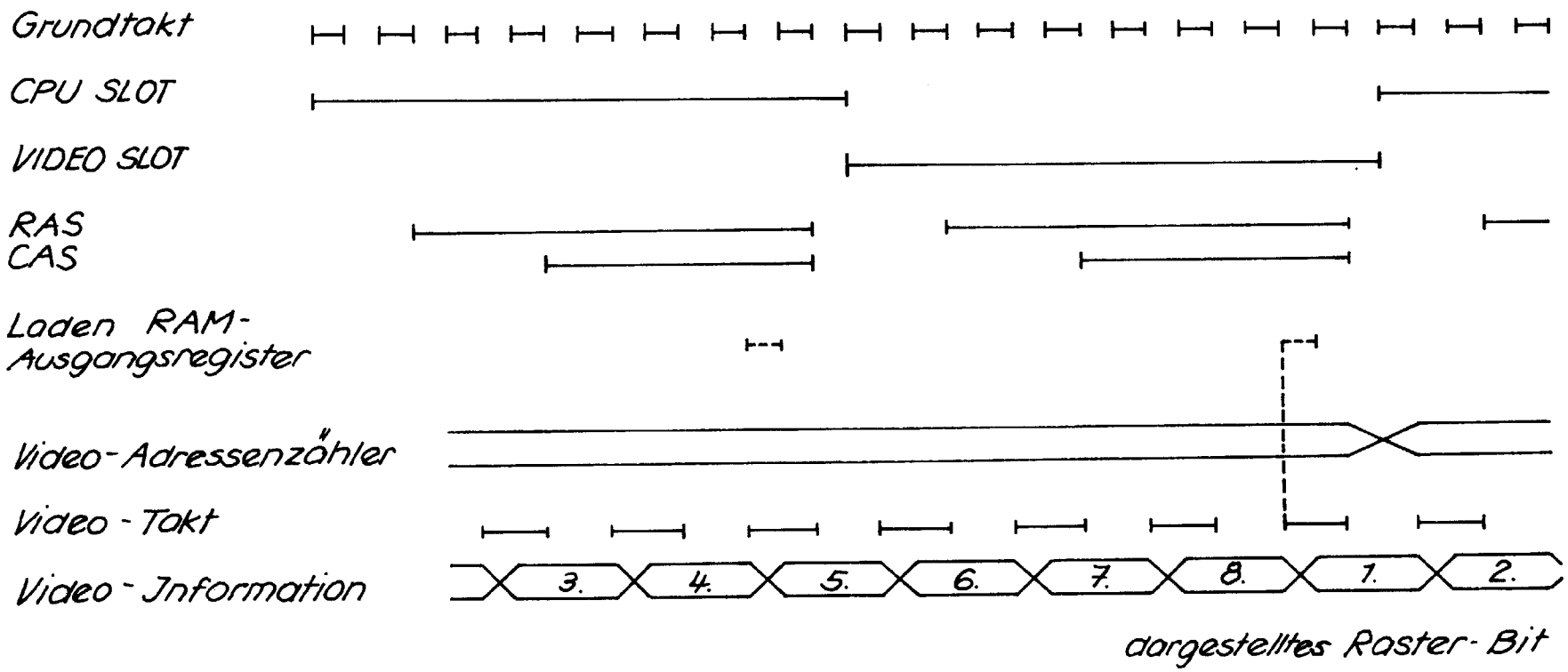
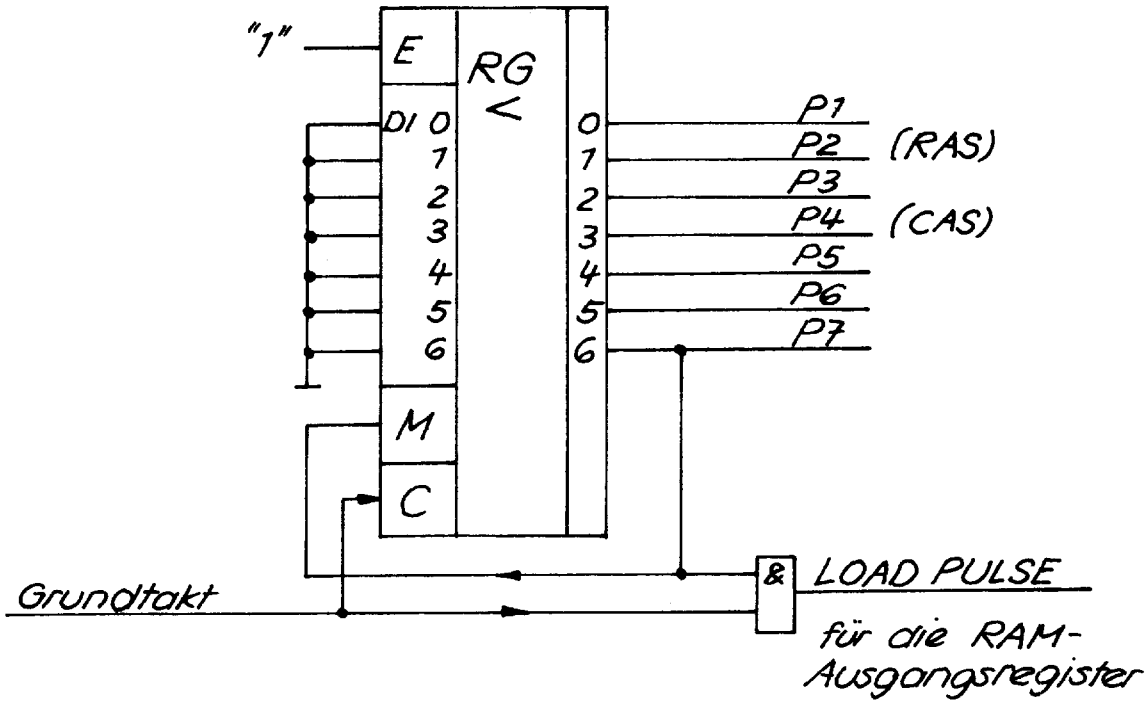
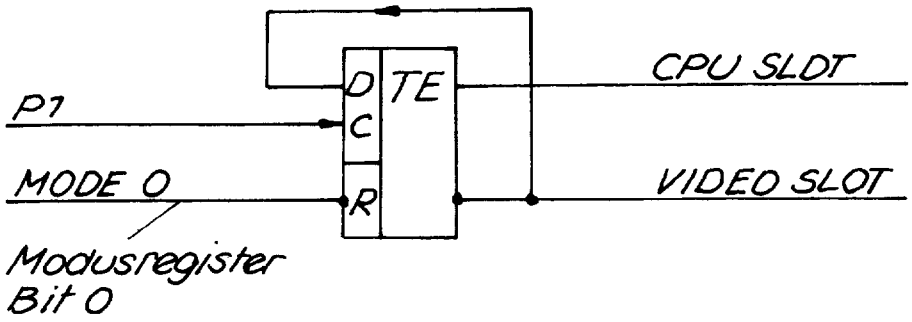

Grundtakt
CPU SLOT
VIDEO SLOT
RAS
CAS
Laden RAM-Ausgangsregister
Video-Adressenzähler
Video-Takt
Video-Information
3.
4.
5.
6.
7.
8.
1.
2.
dargestelltes Raster-Bit

Bild 99


"1"
E
RG
<
DI 0
1
2
3
4
5
6
0
1
2
3
4
5
6
P1
P2 (RAS)
P3
P4 (CAS)
P5
P6
P7
M
C
&
LOAD PULSE
für die RAM-
Ausgangsregister
Grundtakt


D
TE
C
R
P7
MODE 0
CPU SLOT
VIDEO SLOT
Modusregister
Bit 0

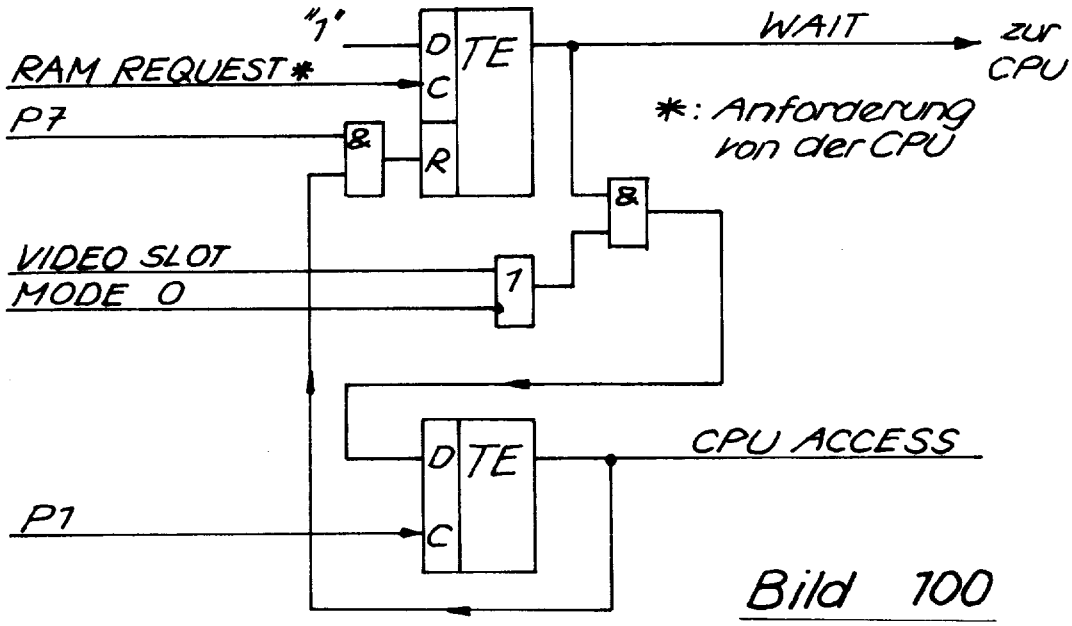

"1"
D
TE
C
R
RAM REQUEST *
P7
&
WAIT
zur
CPU
*: Anforderung
von der CPU
&
VIDEO SLOT
MODE 0
1
D
TE
C
CPU ACCESS
P7

Bild 100

Belegung des Modusregisters

3	2	1	0
INJECT HI	ENABLE INJECT	VIDEO ENABLE	INTERLEAVINGENABLE

Bit 0: schaltet Zeitteilung der RAM-Zugriffe ein

Bit 1: gibt Bilddarstellung frei

Bit 2: erlaubt Schreiben in Zusatz RAM

Bit 3: Schreib-Information für Zusatz-RAM

Besondere RAM-Belegungen

a) Video-Information

	7 … 0
0	8 Bit Video-Information (Punktraster)

b) Synchronsteuerung

	7		2	1	0
1	0			FLY-BACK	SYNC
			UNBLANK CRT		

c) Darstellattribute

	7	6	5	4	3	2	1	0
1	1	COLOR CODE			INTENSITY CODE		BLINK	BLANK OFF
Zusatzbit								

Bild 101

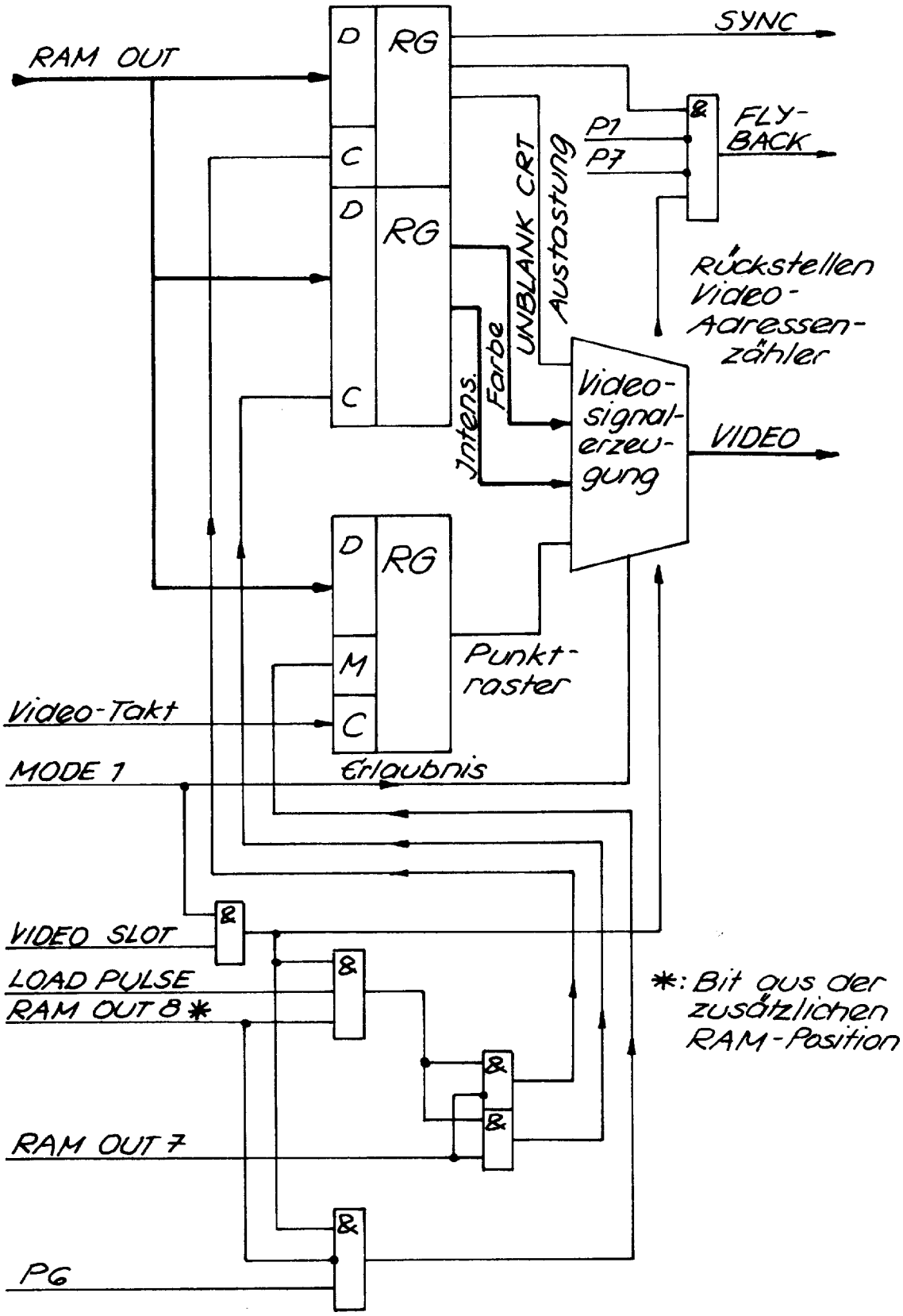

RAM OUT
D
RG
C
D
RG
C
SYNC
P1
P7
&
FLY-BACK
UNBLANK CRT
Austastung
Rückstellen Video-Adressen-zähler
Farbe
Intens.
Video-signal-erzeu-gung
VIDEO
D
RG
M
C
Punkt-raster
Video-Takt
MODE 1
Erlaubnis
VIDEO SLOT
LOAD PULSE
RAM OUT 8 *
RAM OUT 7
PG
*: Bit aus der zusätzlichen RAM-Position

Bild 102

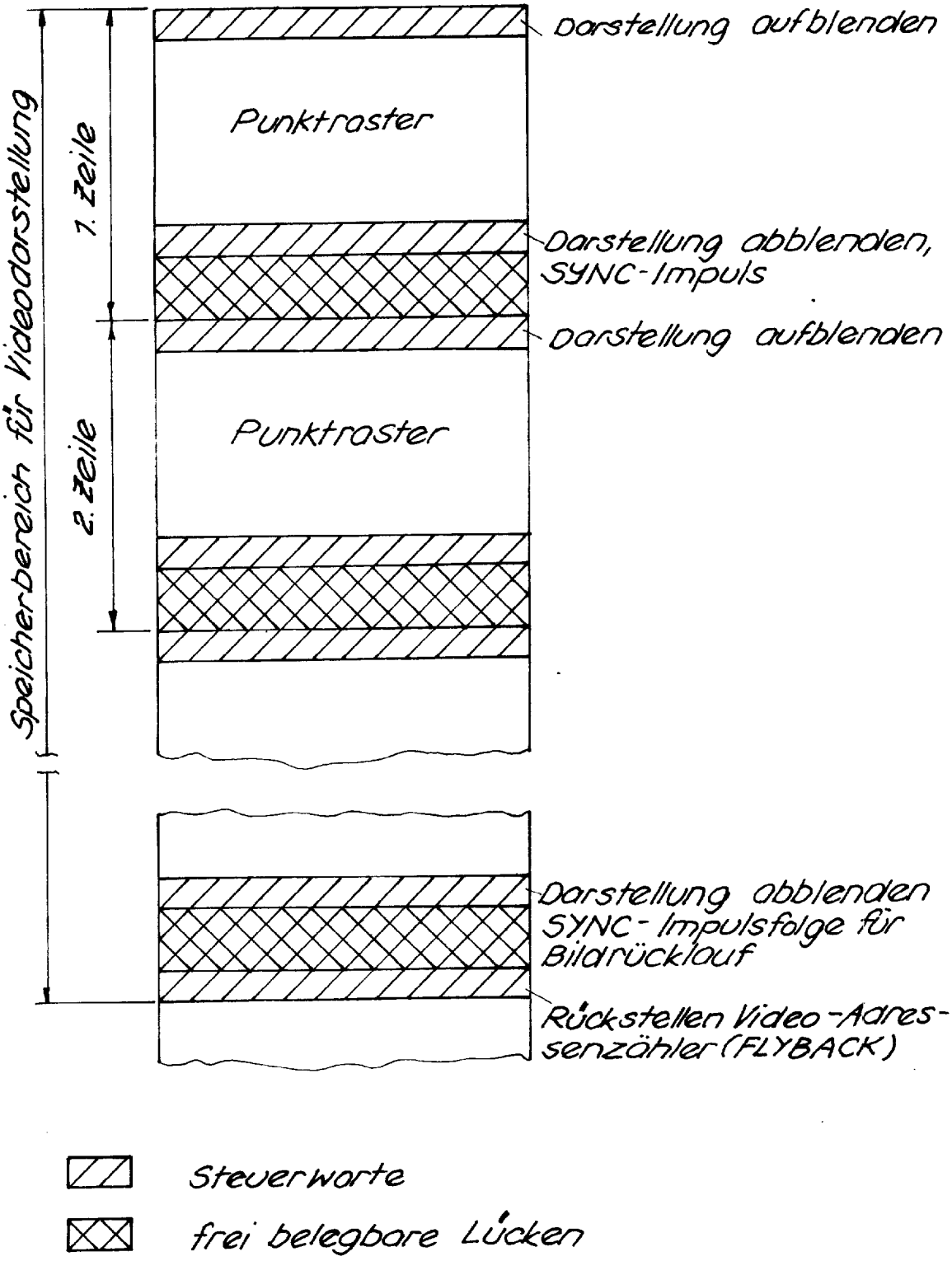

Darstellung aufblenden
Punktraster
Darstellung abblenden, SYNC-Impuls
Darstellung aufblenden
Punktraster
Darstellung abblenden SYNC-Impulsfolge für Bildrücklauf
Rückstellen Video-Adressenzähler (FLYBACK)
Speicherbereich für Videodarstellung
1. Zeile
2. Zeile
Steuerwarte
frei belegbare Lücken

Bild 103

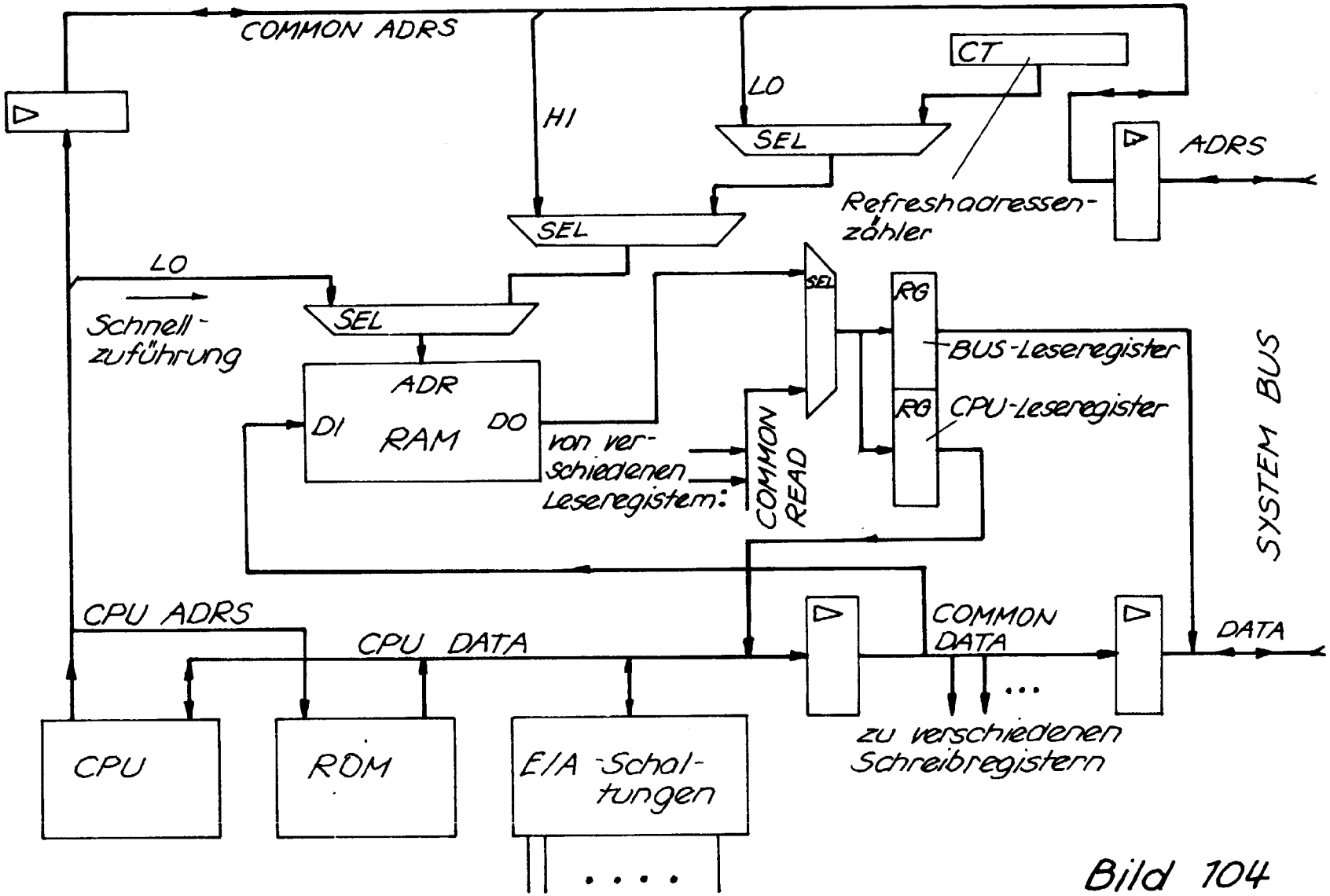

COMMON ADRS
HI
LO
CT
SEL
ADRS
Refreshadressen-
zähler
SEL
LO
Schnell-
zuführung
SEL
ADR
DI
RAM
DO
von ver-
schiedenen
Leseregistern:
COMMON
READ
SEL
RG
BUS-Leseregister
RG
CPU-Leseregister
SYSTEM BUS
CPU ADRS
CPU DATA
COMMON
DATA
DATA
zu verschiedenen
Schreibregistern
CPU
ROM
E/A -Schal-
tungen

Bild 104

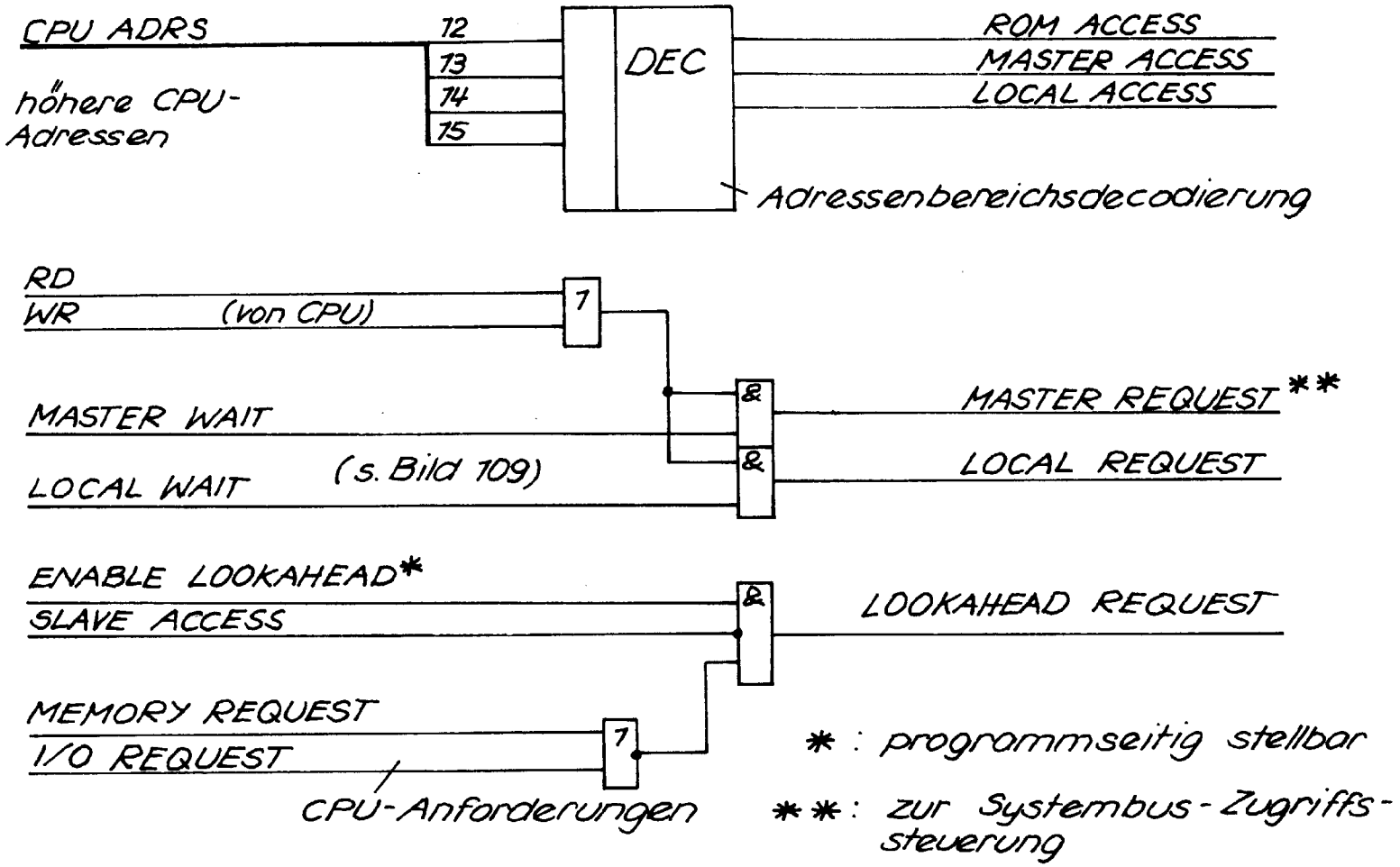

CPU ADRS
höhere CPU-Adressen
12
13
14
15
DEC
ROM ACCESS
MASTER ACCESS
LOCAL ACCESS
Adressenbereichsdecodierung
RD
WR
(von CPU)
1
MASTER WAIT
LOCAL WAIT
(s. Bild 109)
&
&
MASTER REQUEST **
LOCAL REQUEST
ENABLE LOOKAHEAD*
SLAVE ACCESS
&
LOOKAHEAD REQUEST
MEMORY REQUEST
I/O REQUEST
1
CPU-Anforderungen
* : programmseitig stellbar
** : zur Systembus-Zugriffs-steuerung

Bild 105

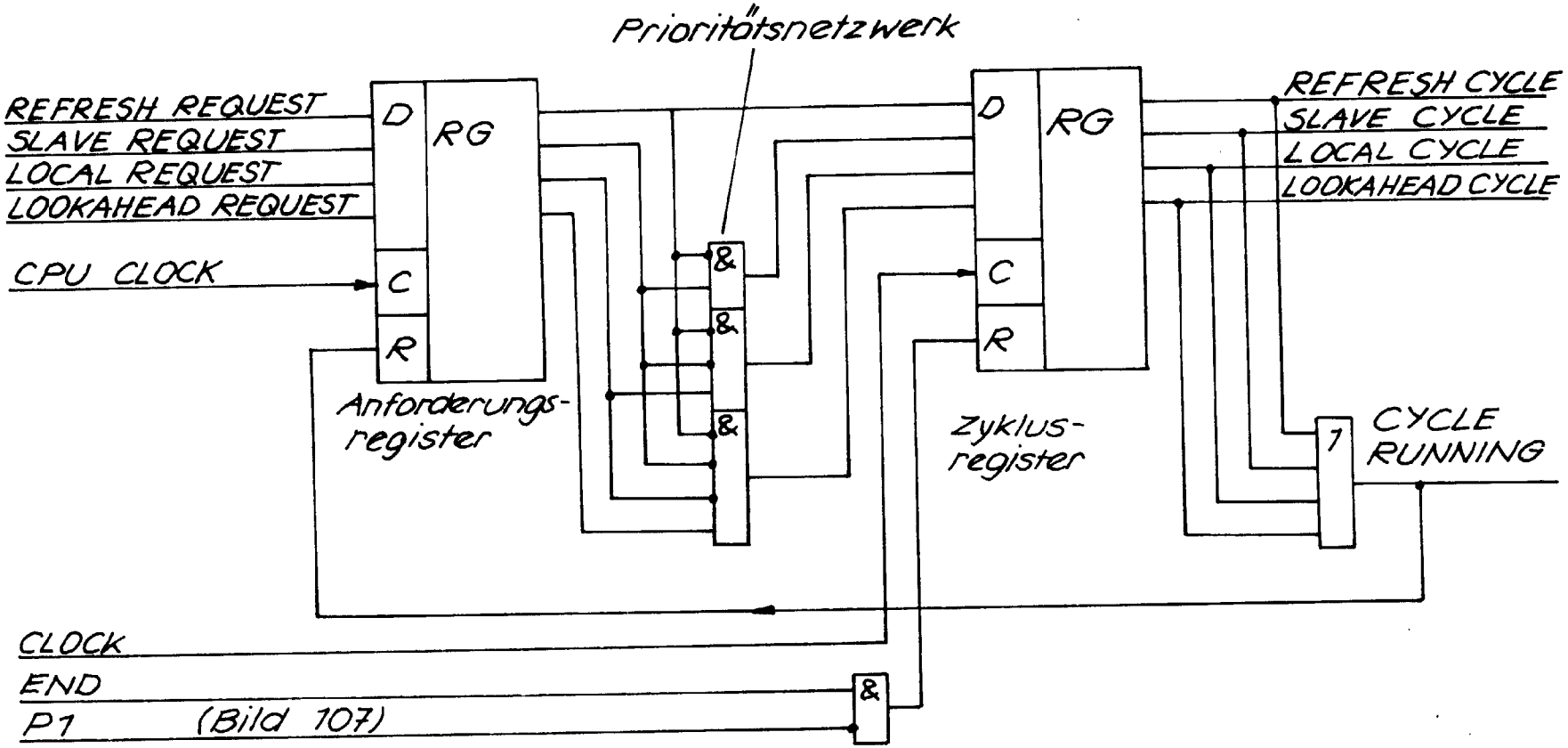

Prioritätsnetzwerk
REFRESH REQUEST
SLAVE REQUEST
LOCAL REQUEST
LOOKAHEAD REQUEST
CPU CLOCK
D
RG
C
R
Anforderungs-
register
&
&
&
D
RG
C
R
Zyklus-
register
REFRESH CYCLE
SLAVE CYCLE
LOCAL CYCLE
LOOKAHEAD CYCLE
1
CYCLE
RUNNING
CLOCK
END
P1 (Bild 107)
&

Bild 106

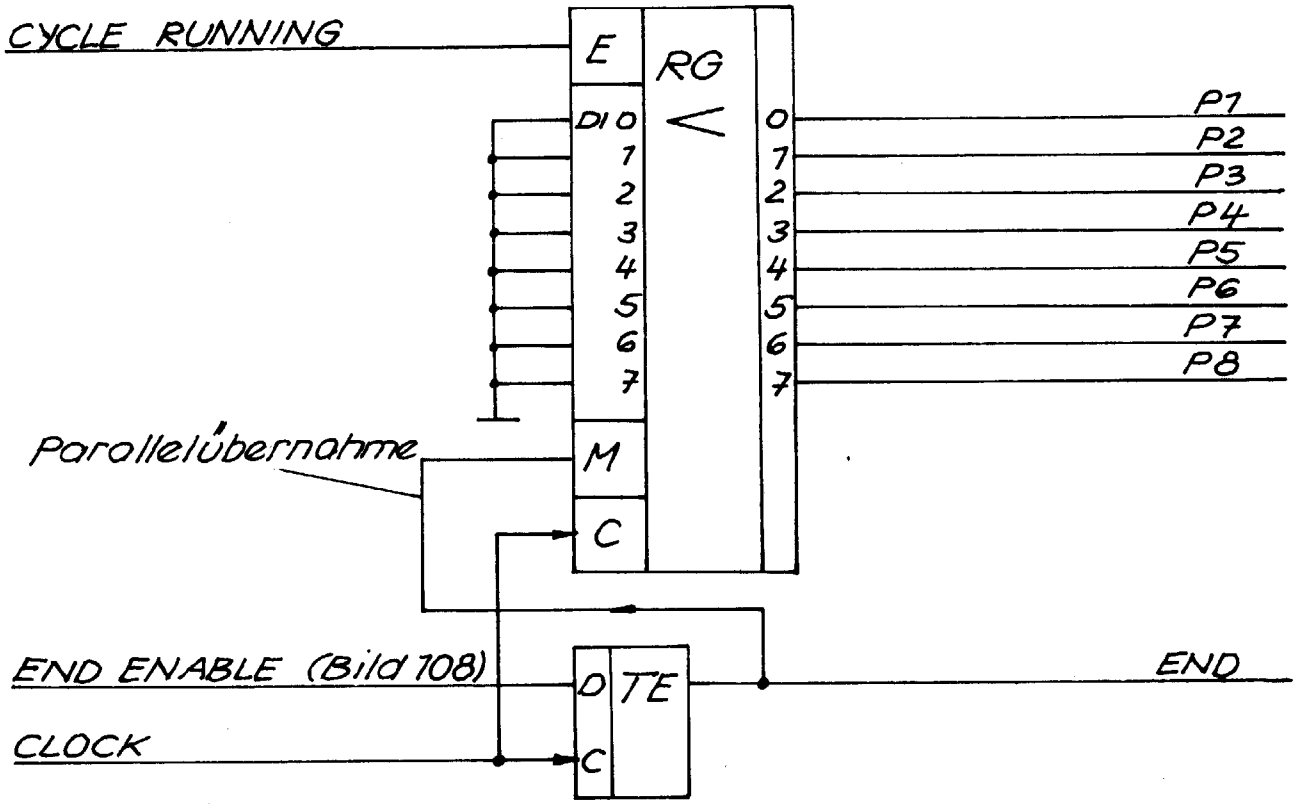

CYCLE RUNNING
E
RG
DI 0
1
2
3
4
5
6
7
0
7
2
3
4
5
6
7
P1
P2
P3
P4
P5
P6
P7
P8
Parallelübernahme
M
C
END ENABLE (Bild 108)
D
TE
END
CLOCK
C

Bild 107

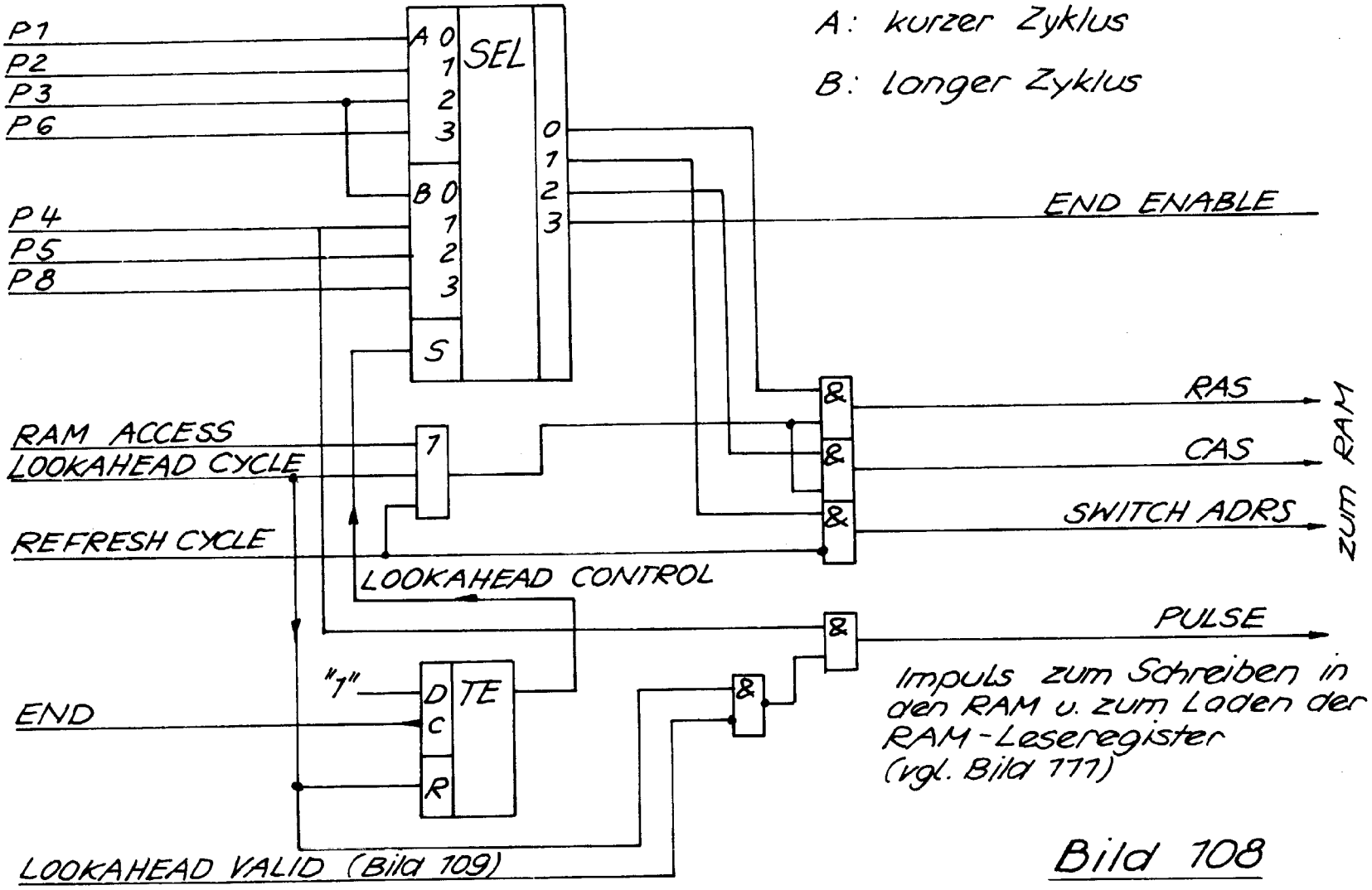

P1
P2
P3
P6
P4
P5
P8
A 0 1 2 3
B 0 1 2 3
S
SEL
0 1 2 3
A: kurzer Zyklus
B: longer Zyklus
END ENABLE
RAS
CAS
SWITCH ADRS
zum RAM
RAM ACCESS
LOOKAHEAD CYCLE
1
REFRESH CYCLE
LOOKAHEAD CONTROL
PULSE
Impuls zum Schreiben in den RAM u. zum Laden der RAM-Leseregister (vgl. Bild 111)
"1"
D
C
R
TE
END
LOOKAHEAD VALID (Bild 109)

Bild 108

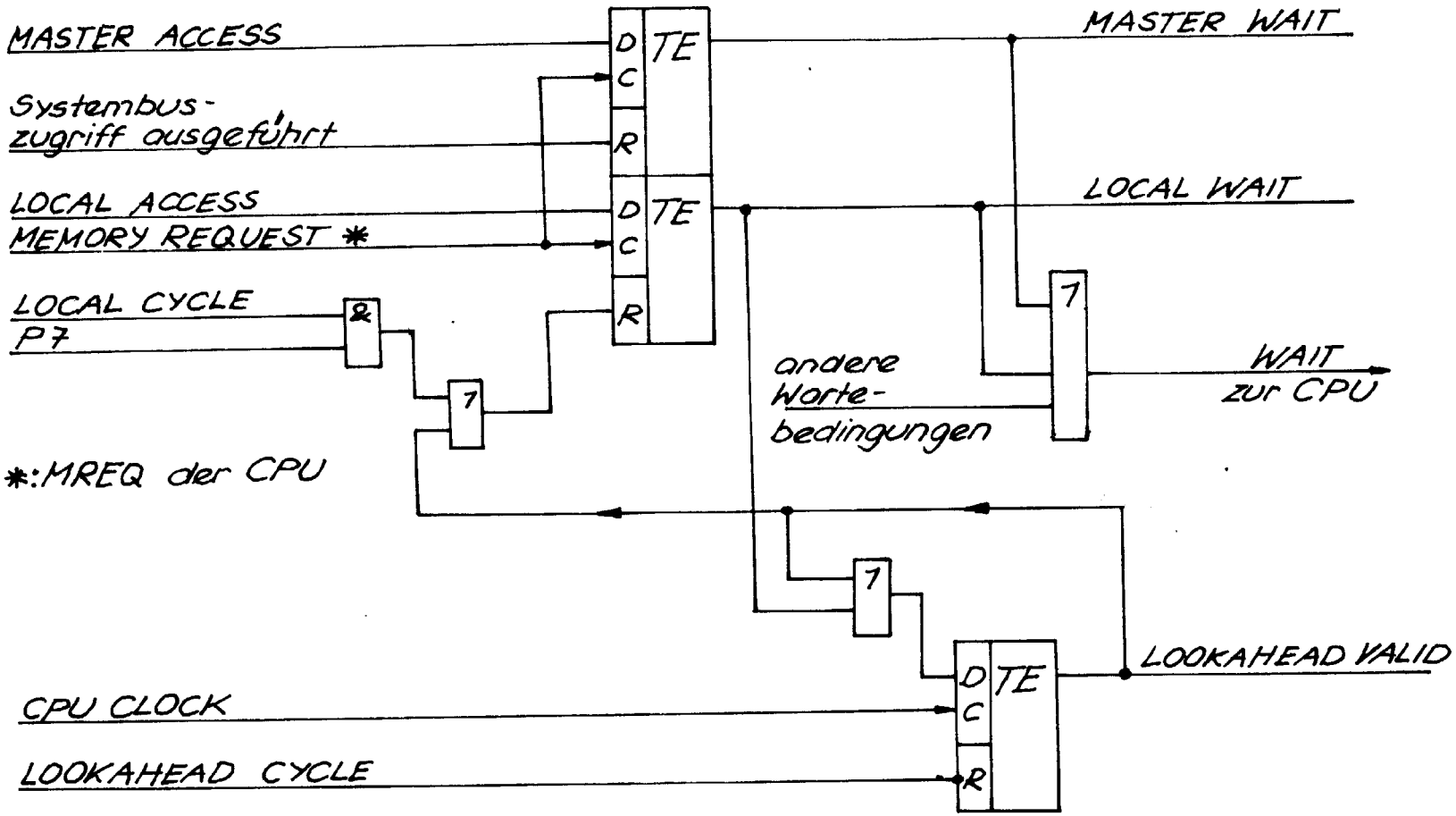

MASTER ACCESS
Systembus-
zugriff ausgeführt
LOCAL ACCESS
MEMORY REQUEST *
LOCAL CYCLE
P7
&
1
D
C
R
TE
D
C
R
TE
MASTER WAIT
LOCAL WAIT
1
andere
Warte-
bedingungen
WAIT
zur CPU
*:MREQ der CPU
1
D
C
R
TE
LOOKAHEAD VALID
CPU CLOCK
LOOKAHEAD CYCLE

Bild 109

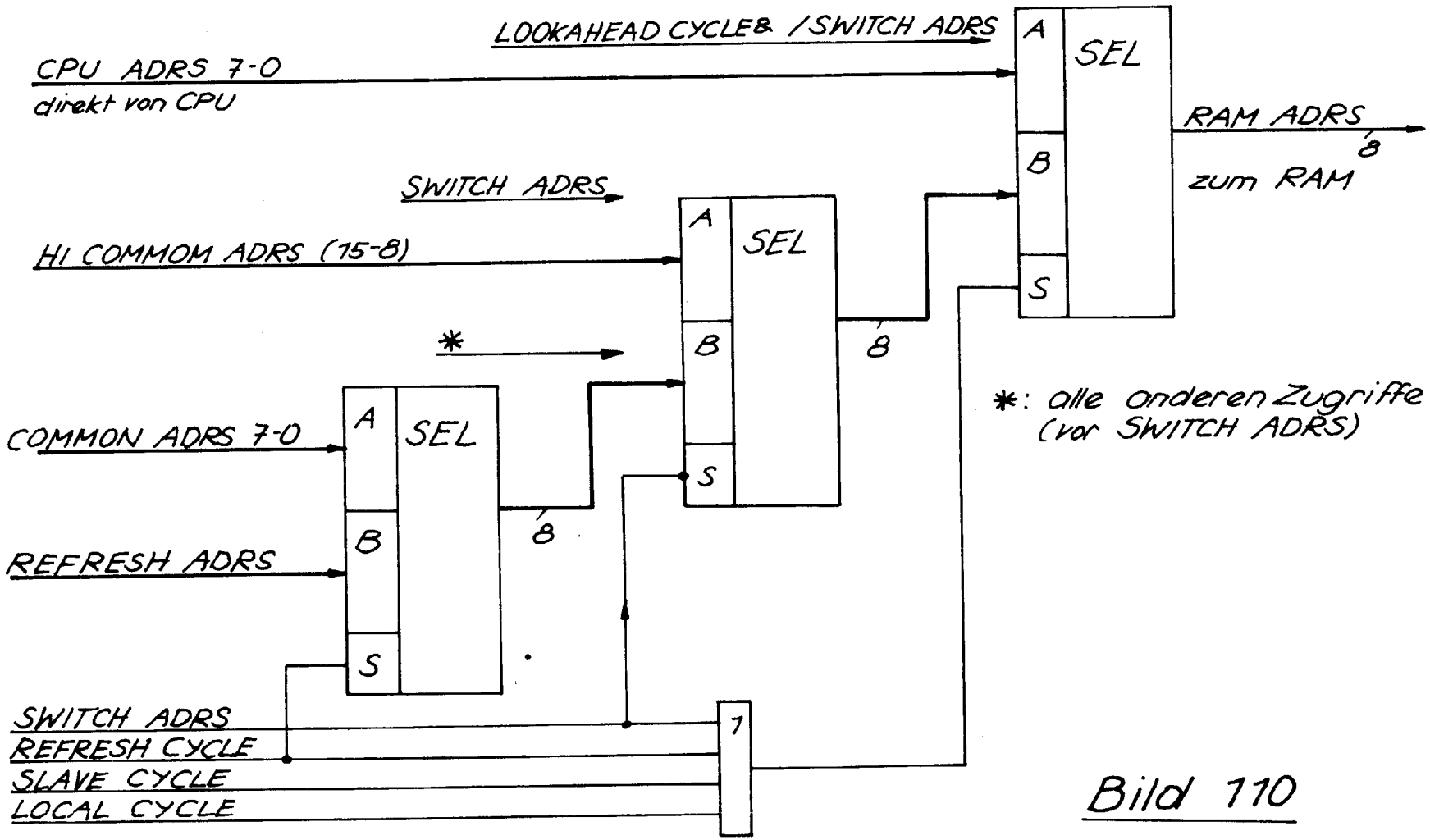

LOOKAHEAD CYCLE& /SWITCH ADRS
CPU ADRS 7-0
direkt von CPU
SWITCH ADRS
HI COMMOM ADRS (15-8)
*
COMMON ADRS 7-0
REFRESH ADRS
SWITCH ADRS
REFRESH CYCLE
SLAVE CYCLE
LOCAL CYCLE
A
B
S
SEL
8
1
RAM ADRS
zum RAM
*: alle anderen Zugriffe
(vor SWITCH ADRS)

Bild 110

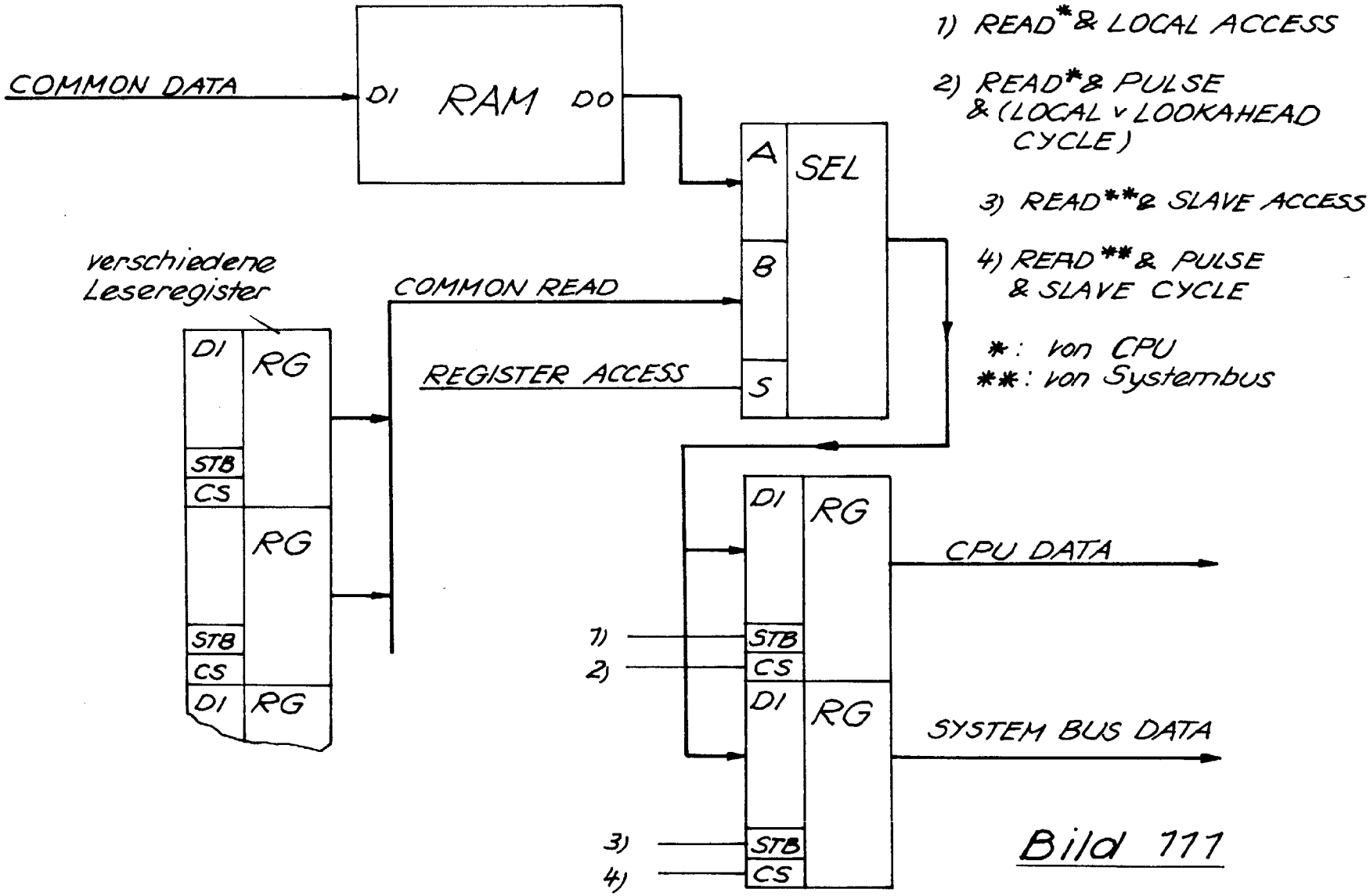

COMMON DATA
DI
RAM
DO
A
SEL
B
S
COMMON READ
REGISTER ACCESS
verschiedene Leseregister
DI
RG
STB
CS
RG
STB
CS
DI
RG
DI
RG
STB
CS
DI
RG
STB
CS
1)
2)
3)
4)
CPU DATA
SYSTEM BUS DATA
1) READ* & LOCAL ACCESS
2) READ* & PULSE & (LOCAL v LOOKAHEAD CYCLE)
3) READ** & SLAVE ACCESS
4) READ** & PULSE & SLAVE CYCLE
*: von CPU
**: von Systembus

Bild 111

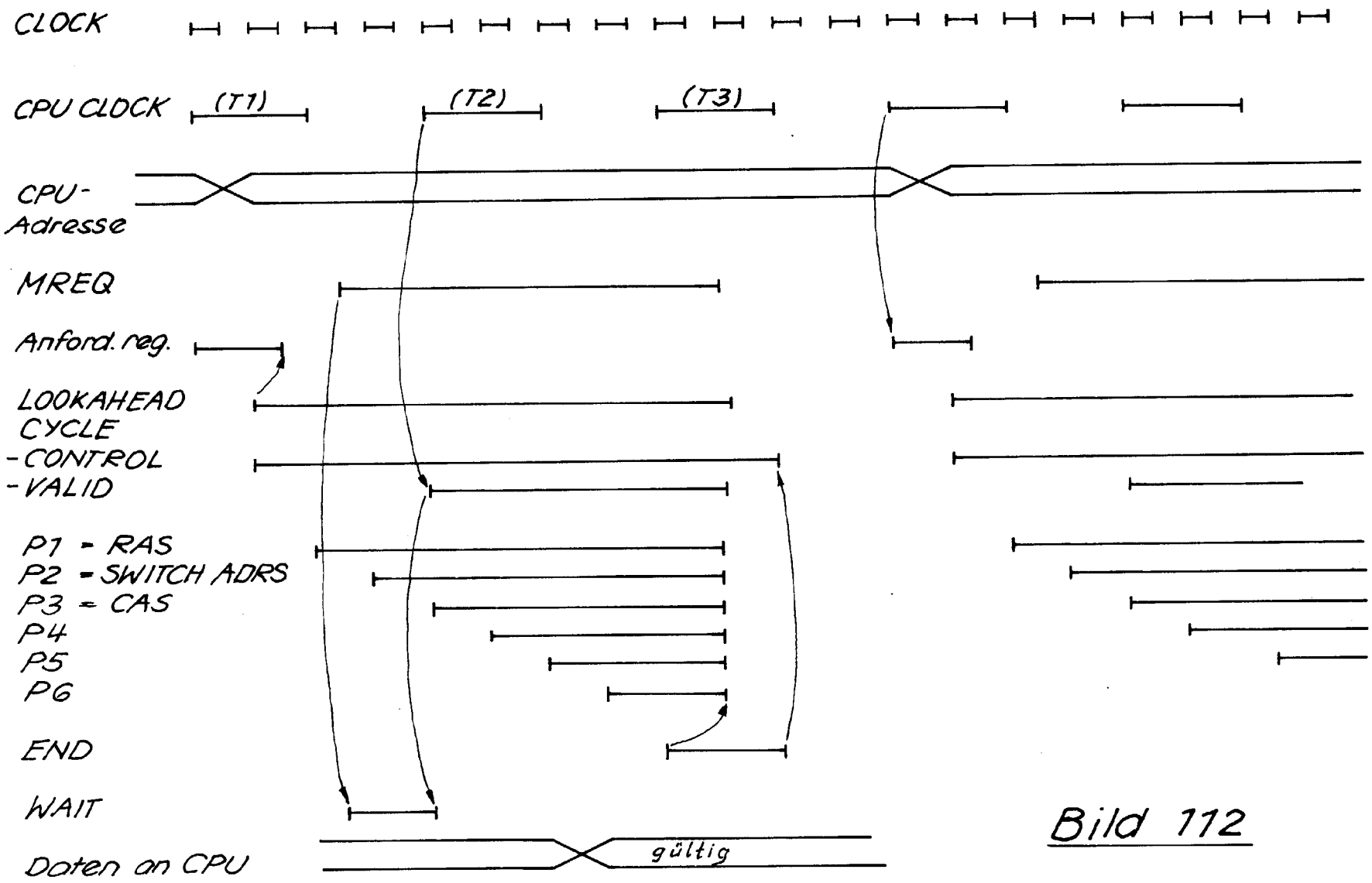

CLOCK
CPU CLOCK
(T1)
(T2)
(T3)
CPU-
Adresse
MREQ
Anford. reg.
LOOKAHEAD
CYCLE
-CONTROL
-VALID
P1 = RAS
P2 = SWITCH ADRS
P3 = CAS
P4
P5
P6
END
WAIT
Daten an CPU
gültig

Bild 112

CLOCK
CPU CLOCK
Anford. reg.
CYCLE
P1
P2
P3 = RAS
P4 = SWITCH ADRS
P5 = CAS
P6
P7
P8
WAIT

Bild 113