– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Dumm dürft Ihr sein, aber laßt Euch was einfallen ...
M. Grundig, E. Heinkel, W. Messerschmitt, H. Nixdorf u. v. a. m.

There are nine and sixty ways of constructing tribal lays,
And every single one of them is right!
Rudyard Kipling

Es ist immer von Vorteil, über eine gut gefüllte Werkzeug- und Trickkiste zu verfügen – und nicht nur über einen einzigen Hammer.

Es schadet also nichts, weitere Alternativen der Schaltungsentwicklung und Problemlösung kennenzulernen (Horizonterweiterung).

Hier geht es darum, das altbewährte Prinzip der Mikroprogrammsteuerung wieder zu beleben, natürlich auf aktueller Grundlage. Im Vergleich zum fertigen Prozessor (als IP Core) braucht das Mikroprogrammsteuerwerk weniger Ressourcen, ist schneller und weist ein definiertes Realzeitverhalten auf, wenn es sein muß, bis auf den einzelnen Maschinentakt.

Die Erläuterungen beginnen mit einem kurzen Überblick, der diese Bemühungen rechtfertigen soll.

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Die Grundsatzfrage: Schaltungsentwurf oder Programmierung?

Sie ist nicht definitiv zu entscheiden, sondern immer wieder von neuem zu untersuchen.

Hier geht unsere Absicht in Richtung Programmieren, aber auf Grundlage eigener Plattformen. Dafür greifen wir auf Ideen zurück, die vor Jahrzehnten zum Stand der Technik gehört haben. Sie werden erneut ventiliert und an die aktuellen Gegebenheiten angepaßt. Dieses Vorhaben soll zunächst begründet werden.

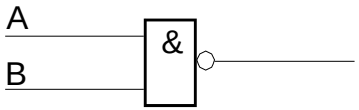
- Der Schaltungsentwurf bietet die größtmögliche Gestaltungsfreiheit.
- Die Schaltung muß aber implementiert werden. Das ist ein Problem der Booleschen Algebra. Es hat NP-vollständige Zeitkomplexität. Es kann sein, daß alles in kurzer Zeit erledigt ist. Der Zeit-und Speicherplatzbedarf kann aber auch in die Größenordnung kommen, die erforderlich wäre, die gesamte Lösungsmenge des Anwendungsproblems zu bilden und zu durchmustern (exponentielle Abhängigkeit mit $O(2^n)$ oder höher). Mit anderen Worten, es kann dauern.
- Das ist auch denn der Fall, wenn die Entwurfsabsicht auf eine Weise erfaßt wird, die wie gewöhnliches Programmieren aussieht (Verhaltensbeschreibung). Das Beschreiben ist leicht, das Synthetisieren dauert, das Fehlersuchen ist schwierig.
- Sehr komplexe Anwendungsprobleme ausschließlich mit Schaltungsentwurf zu lösen (Spezialhardware), kostet Zeit und Geld, und zwar ausgesprochen viel. Es ist vergleichsweise aufwendig, Entwurfsfehler zu erkennen und zu beseitigen (Fehlersuche in der Hardware, mehrere Iterationen der Schaltungssynthese).
- Demgegenüber ist es eine Erfahrungstatsache, daß man auch sehr komplexe Probleme durch Programmieren lösen kann. Das gelingt deshalb, weil alles von Anfang an auf elementare Verarbeitungsschritte zurückgeführt wird und die natürliche Intelligenz bereits bei der Problemerfassung zur Wirkung kommt. Programmieren ist irgendwie immer Tricksen und Hacken ...
- Die eigentliche Komplexität der Anwendungslösung steckt im Programm. Die meisten Fehler sind somit Programmierfehler, die mit gewohnten Verfahren gesucht und beseitigt werden können (Ablaufverfolgung, Programmänderung).

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Zwei typische Entwurfsfehler

Viele Entwurfsfehler sind im Grunde Kleinigkeiten. Wie lange dauert es, sie zu beseitigen?

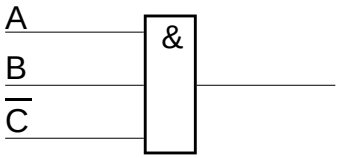
Das wurde implementiert:



Verzweige, wenn A = B

```
CMP A, B
JPE
```

Und so ist zu ändern:



Verzweige, wenn A = B UND C = 0

```
    CMP A, B
    JPNE next
    AND C, C
    JPZ
next:
```

Wie lange dauert das?

Denken wir an eine Maschine mit einigen hunderttausend Gattern.

Ganz früher: Leiterplatte raus, sehen, in welchen Schaltkreisen noch passende Gatter frei sind, Leiterzüge auffräsen, Draht löten. Wenn man schustern darf, dauert es etwa eine halbe Stunde. Wenn es ein förmlicher Änderungsantrag sein muß, mit Genehmigung vom Chef und Ausführung in der Zentralwerkstatt, ist es nach oben offen ...

Wie lange aber dauert die Schaltungssynthese, wenn es ein einziges großes FPGA ist? Schustern von Hand unmöglich ...

Zu Zeiten der PDP-11 und S/360 konnte man die Befehle am Bedienpult mit Kippschaltern eingeben. Wort für Wort einstellen und mit DEPOSIT NEXT laden. Dauerte ein paar Minuten. Ehrensache, die Befehle auswendig zu wissen ...

Wenn man heutige Software vernünftig modular strukturiert hat, sollte das Compilieren und Laden auch nicht länger dauern. Auf jeden Fall ist die Zeitkomplexität weit geringer als NP-vollständig.

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Was ist schneller?

Eine zweckgerecht entworfene Schaltung im FPGA oder ein fertiger Prozessor und das dort laufende Programm?

FPGA = programmierbare Logik:

- Alles muß mit Logikzellen aufgebaut und über vorgegebene Signalwege untereinander verbunden werden. Auch die Taktverteilung ist vorgefertigt.
- Die Taktfrequenzen im Datenblatt betreffen die Maximalwerte der Taktverteilung und Taktaufbereitung.
- 1 GHz heißt, daß die Taktverteilung, die Taktsignalwege usw. mit 1 GHz zurechtkommen.

Prozessorkern = hart verdrahtete Logik:

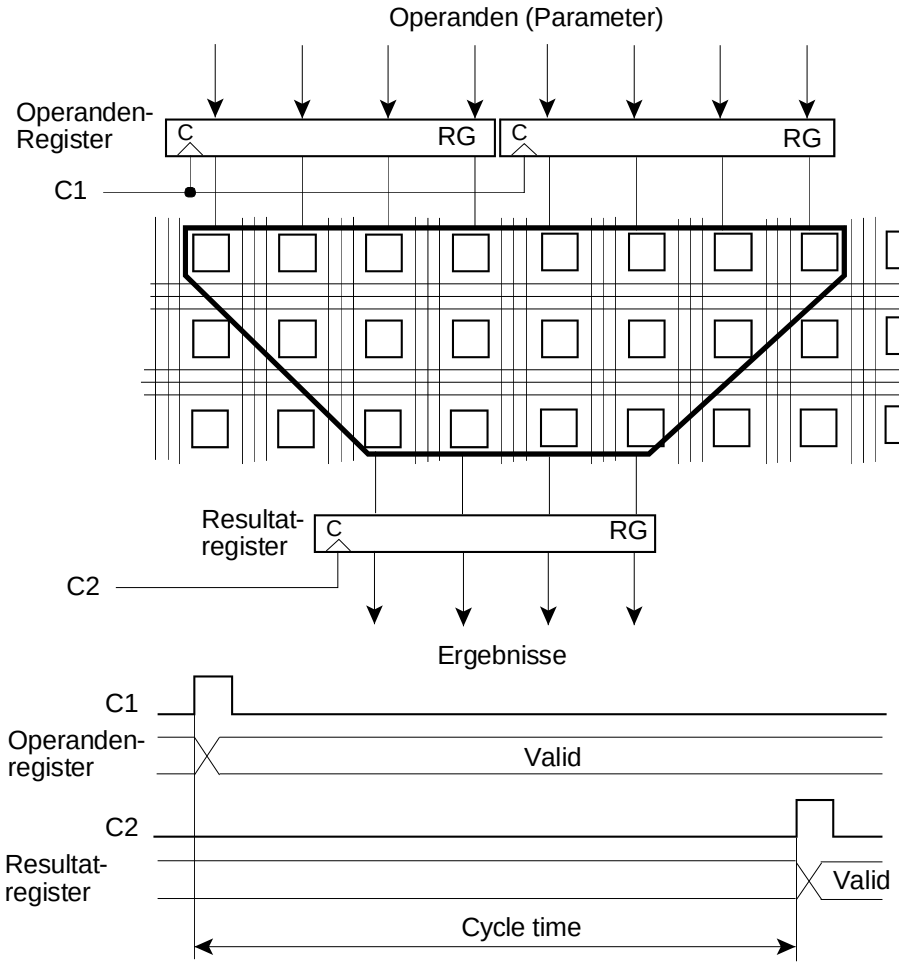
- Die Gatter und Schaltnetze sind bis auf den Transistor optimiert.
- Die Signalwege werden so kurz wie möglich gehalten.
- Die Taktfrequenzen im Datenblatt betreffen die Taktfrequenzen der Anwendungspraxis.
- 1 GHz heißt, daß die typische binäre Addition zweier Maschinenwörter 1 ns dauert.

Zahlenbeispiele aus der Praxis:

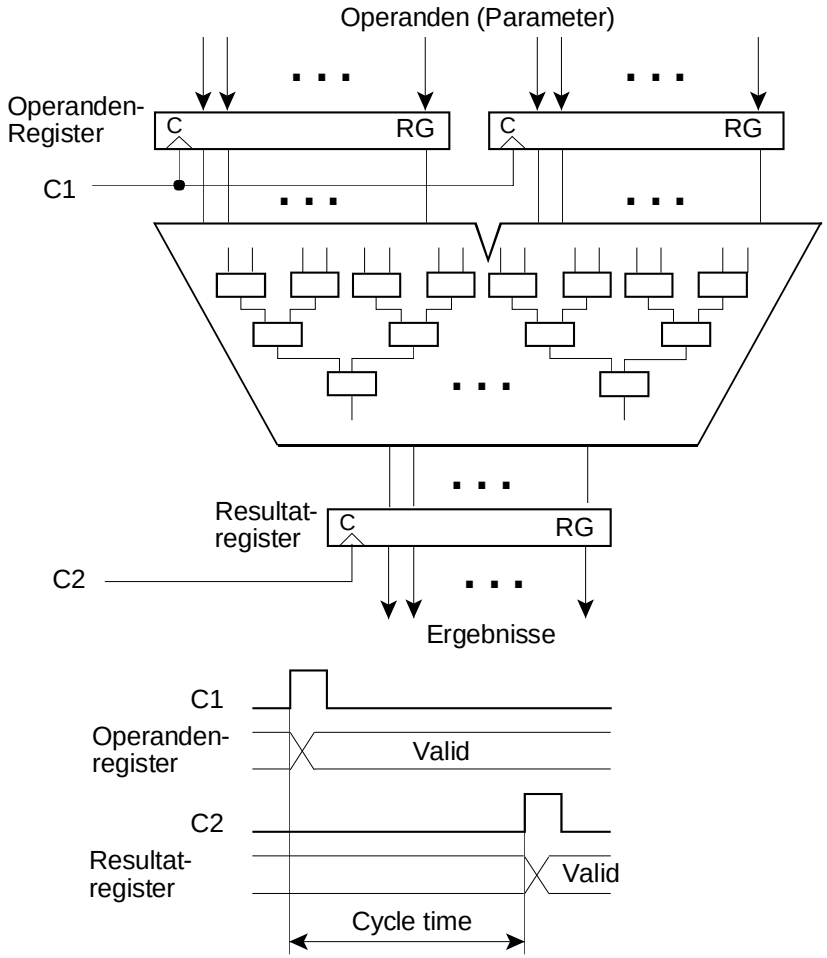
- Transistorverbrauch: Ein FPGA mit 75 Millionen Transistoren ist nicht in der Lage, eine Prozessorschaltung aufzunehmen, die hart verdrahtet 7,5 Millionen Transistoren erfordert.
- Taktfrequenzen: Der fertige Prozessor ist so schnell, wie es im Datenblatt steht. Mit welcher Taktfrequenz kann aber die Anwendungsschaltung im FPGA betrieben werden? Das hängt von der Schaltungstiefe, den genutzten Signalwegen usw. ab. Es ist auf jeden Fall langsamer, als es im Datenblatt steht (Slowdown). Für pauschale Überlegungen ist eine Verlangsamung im Verhältns 10:1 eine zweckmäßige Annahme (fertiger Universalprozessor 2 GHz, Soft Core im FPGA 200 MHz). Je höher die Taktfrequenz der Anwendungs-schaltung sein soll (als Entwurfsanforderung), desto größer die Entwurfsschwierigkeiten.

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Die Schaltung im FPGA



Die Schaltung im Prozessorkern



– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Was ist schneller?

Eine Schaltung im FPGA oder ein fertiger Prozessor und das dort laufende Programm?

Wann lohnt sich die Verlagerung in ein FPGA?

- Taktverlangsamung (Slowdown) = 1:10. 2 GHz im Datenblatt bedeuten, man kann eine einprogrammierte Prozessorschaltung (Soft Core) mit 200 MHz betreiben.
- Ein Superskalarprozessor kann 2 bis 4 Befehle gleichzeitig erledigen, die nützliche Arbeit verrichten. Das ergibt sich u. a. beim Ausführen kürzester innerster Schleifen. Der Schleifenkörper bestehe aus einem Adreßrechenbefehl und einem Verarbeitungsbefehl, der Schleifenkopf aus einem Iterationsbefehl und einem Verzweigungsbefehl. Wenn die Maschine 4 Werke hat, kann jedes einen dieser Befehle ausführen. Unsere Annahme ist somit plausibel.
- Das FPGA müßte also in einem Taktzyklus mehr leisten als ein Programmstück von 20 bis 40 Befehlen.

Ja, es ist eine Milchmädchenrechnung:

- Die 40 Befehle sind nur ein statistischer Wert. Wenn Datenabhängigkeiten vorkommen oder auf die Ein- und Ausgabe gewartet werden muß, sind es viel weniger.
- Die Schaltung im FPGA kann direkt mit der Außenwelt bzw. der Peripherie zusammenwirken. Sie kann bis auf den Maschinentakt genau arbeiten (streng deterministisches Realzeitverhalten).
- Der Schaltungsaufwand des Prozessors wurde nicht berücksichtigt.
- Superskalarprozessoren, die man zu kaufen bekommt, können nicht so eingesetzt werden wie hier beschrieben. Sie sind zum Verarbeiten gespeicherter Daten optimiert, nicht zur Interaktion mit der Einsatzumgebung. Wenn man E-A-Befehle braucht, geht die Milchmädchenrechnung nicht mehr auf.
- Deshalb die Geschäftsidee: Superskalarprozessoren bauen, die man tatsächlich im Sinne der Milchmädchen-rechnung einsetzen kann, die also gute Chancen haben, beim Wettrechnen gegen Einzweckschaltungen im FPGA zu gewinnen. Hierzu gibt es mehrere Ansätze und Entwurfsgedanken. Im folgenden betrachten wir eine dieser Alternativen.

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Programmieren ist VIEL bequemer ...

Man kann hacken und schustern, bis es einigermaßen funktioniert, ohne daß dabei Synthesealgorithmen mit NP-vollständiger Komplexität durchlaufen werden müssen.

Es ist experimentell bewiesen worden, daß man auf diese Weise bis zum Mond und zurück kommt, und zwar mit weit weniger als 100 kBytes Speicherkapazität ...

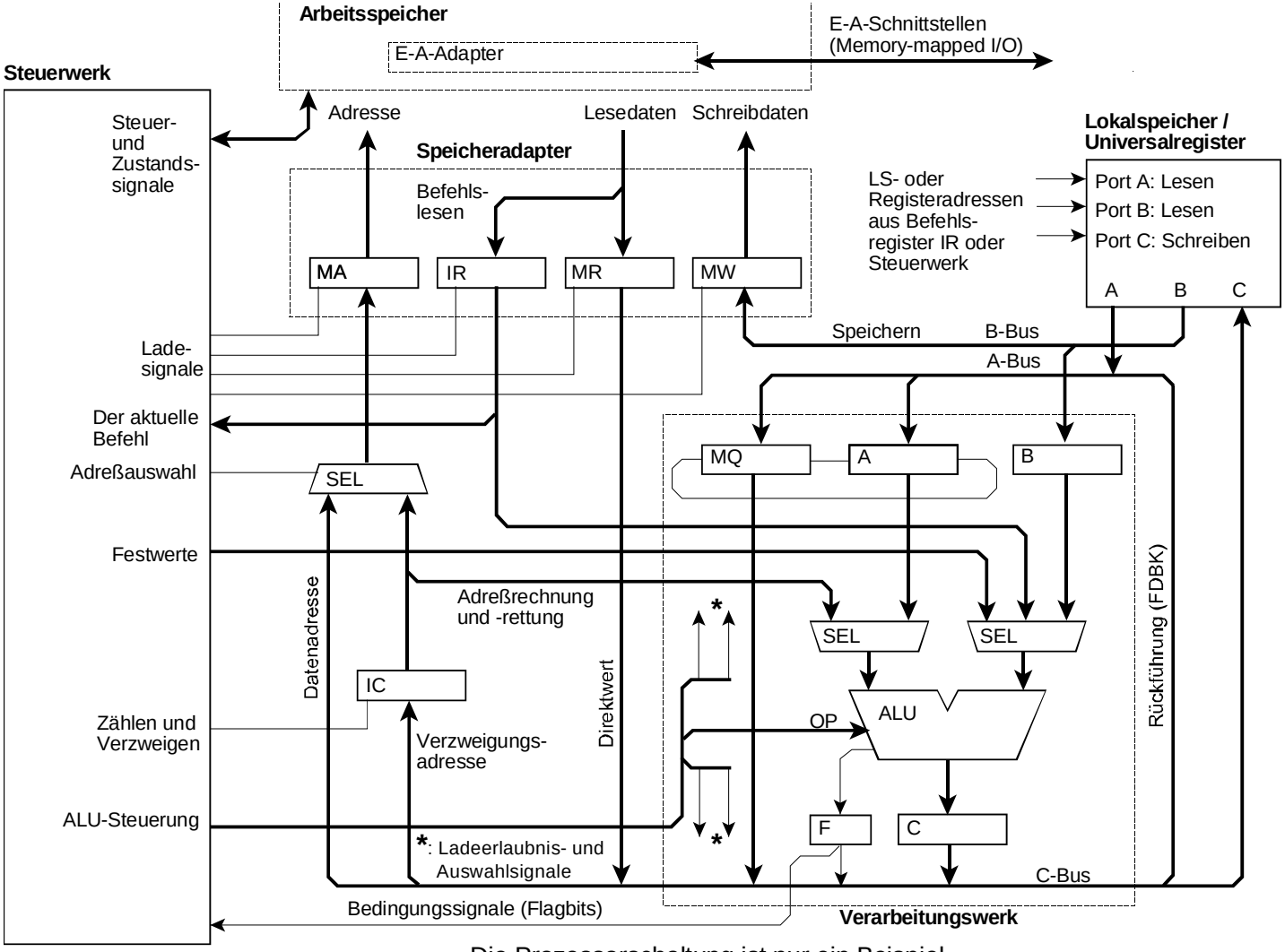
Daraus ergibt sich eine naheliegende Aufgabenstellung:
Architekturen und Schaltungslösungen für frei programmierbare Maschinen, um anwendungsspezifische Schaltungen zu ersetzen oder zu ergänzen.

Unser Ansatz besteht darin, ein altbewährtes Prinzip erneut zu ventilieren: die Mikroprogrammsteuerung.

- Eine Mikroprogrammsteuerung kann nicht rechnen, sondern nur verzweigen und Steuerwirkungen ausüben. Mikroprogramme bestehen aus Mikrobefehlen, die in Formatgestaltung und Wirkungsweise an die zu steuernden Einrichtungen angepaßt sind.
- Das Prinzip wurde ursprünglich entwickelt, um Steuerwerke für universelle Prozessoren zu entwerfen.
- Man hat die Mikroprogrammsteuerung nicht selten auch dann eingesetzt, wenn das Problem auch mit einer sequentiellen Einzweckschaltung zu erledigen gewesen wäre, man also gar nicht gezwungen wäre, zu programmieren.
- Im Grunde macht sie sogar mehr Arbeit: Man muß zunächst das Steuerwerk entwerfen und sich um die Entwicklungsumgebung kümmern. Erst dann kann man die eigentliche Anwendungsaufgabe durch Programmieren lösen.
- Aber: man verlagert die Komplexität von der Schaltung in einen Speicherinhalt.
- Die meisten Änderungen sind dann keine Schaltungsänderungen, sondern Programmänderungen, die meisten Entwurfsfehler sind keine Schaltungsfehler, sondern Programmierfehler.
- Mit anderen Worten: wenn das Steuerwerk erst einmal läuft, kann man unbekümmert drauflos hacken, wenn alles soft ist, kann man solange beim Kunden ändern, wie der sich das gefallen läßt ...
- Mikroprogramme laufen schneller. Wenn man Anwendungsprogrammfunktionen ins Mikroprogramm verlagert, ergibt sich ein Speedup in der Größenordnung 1:2 bis ca. 1:40 (Erfahrungswerte).
- Beim heutigen Stand der Technik ist es ein weiteres Entwurfsprinzip, das man in den Werkzeugkasten zur Problemlösung aufnehmen sollte. Damit kann man im Grunde alles steuern, was zu steuern ist.

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

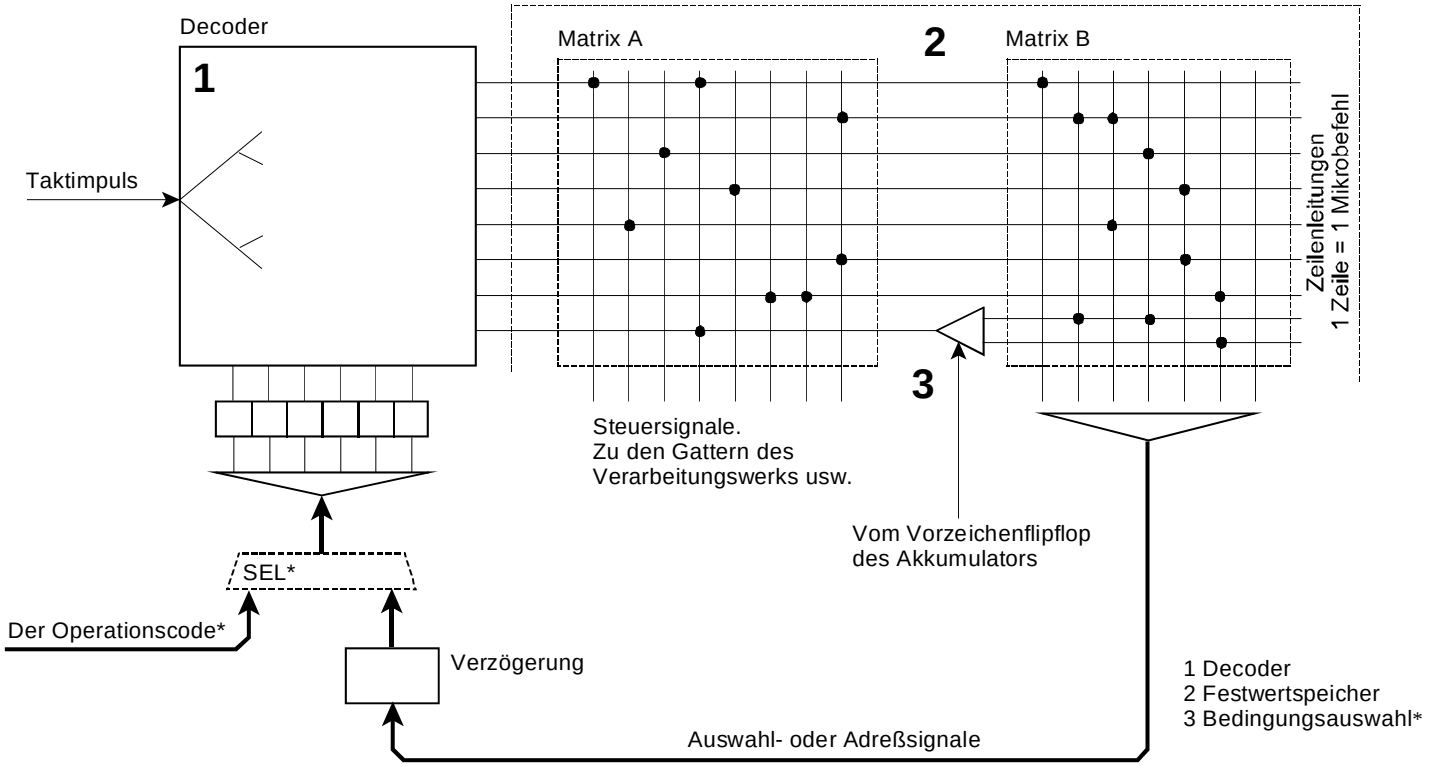
Die ursprüngliche Steuerungsaufgabe: der Universalprozessor. Die Maschine braucht ein Steuerwerk



– Die Prozessorschaltung ist nur ein Beispiel –

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Das Prinzip der Mikroprogrammsteuerung. So wurde es in der Originalarbeit von M. V. Wilkes (1951) veranschaulicht:



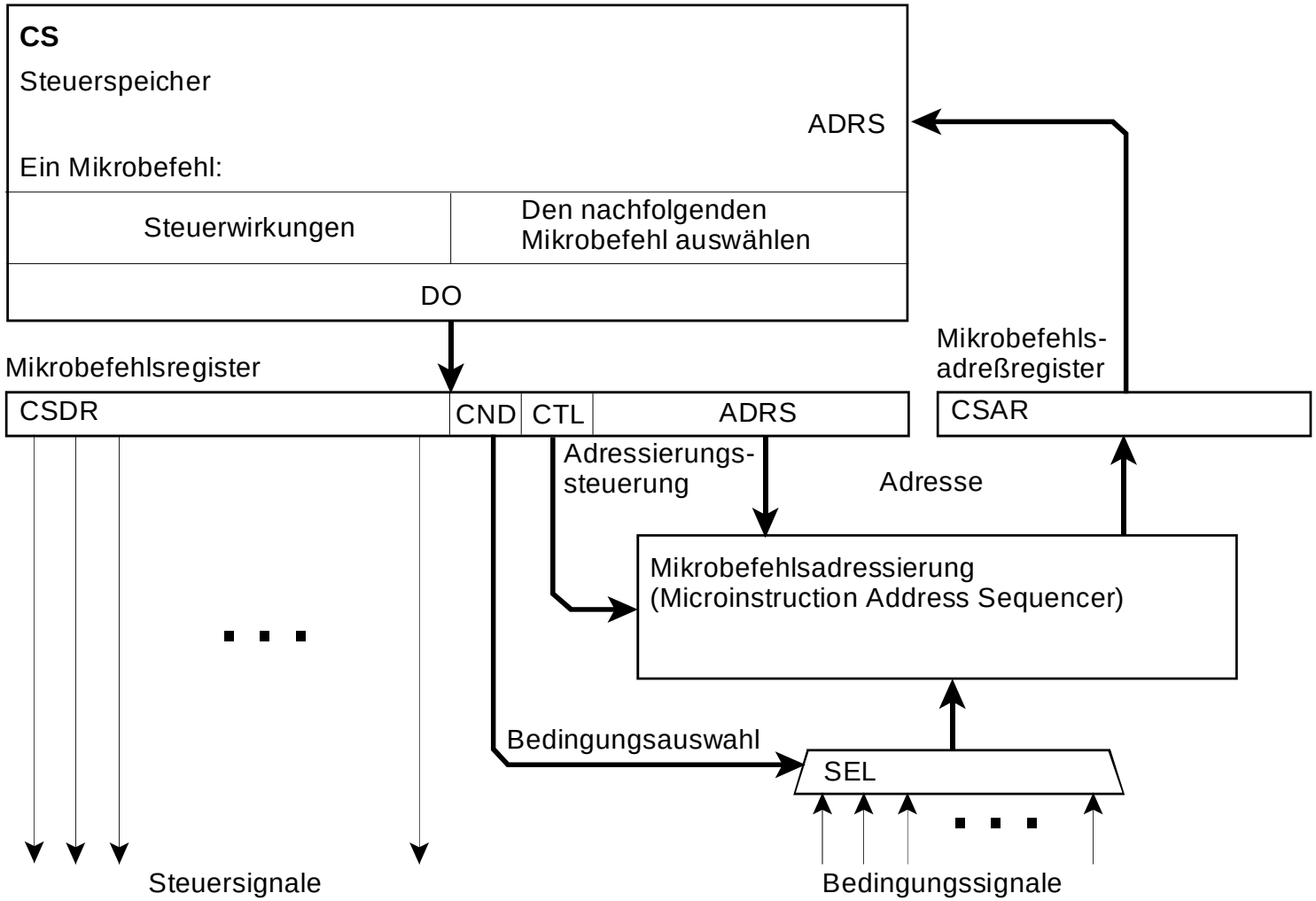
	Matrix A	Matrix B
Das grundsätzliche Mikrobefehlsformat*:	Steuerwirkungen	Den nachfolgenden Mikrobefehl auswählen
Die Mikrobefehlsadresse*:		
Mikrobefehlsadressierung vom Operationscode*:	0 0 ... 0	Der Operationscode

*: Bedingungsauswahl:
Hier sind einer Zeilenleitung der Matrix A über eine Auswahlstufe 3 zwei Zeilenleitungen der Matrix B nachgeschaltet. So werden zwei alternative Folgeadressen gespeichert. Die Auswahl wird hier vom Vorzeichenflipflop des Akkumulators gesteuert. Ist das Vorzeichen beispielsweise positiv, wird der nächste Mikrobefehl von der ersten der beiden Folgeadressen gelesen, ist das Vorzeichen negativ, von der zweiten.

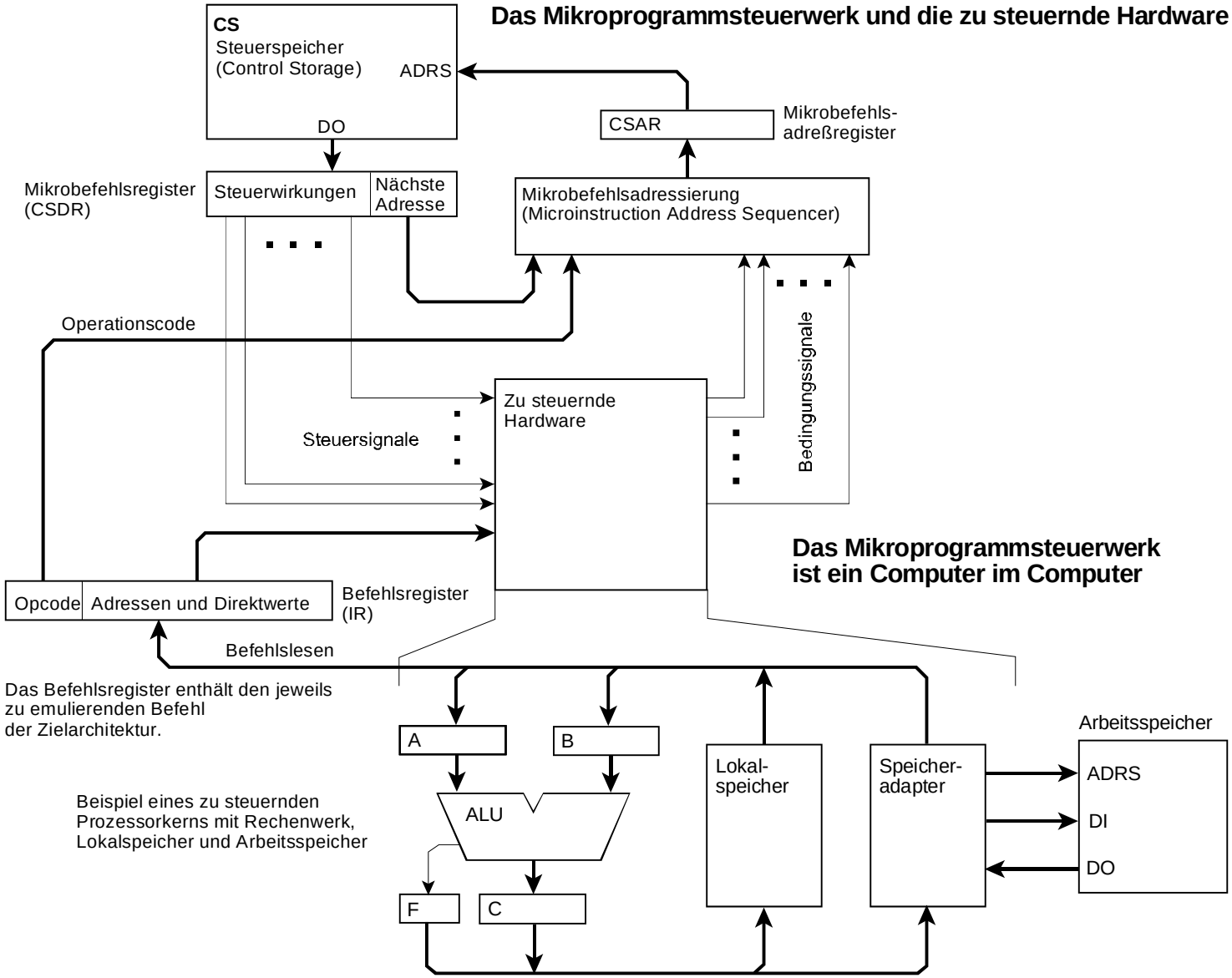
*: Nicht in der Originalarbeit. Ergänzungen im Sinne der Anschaulichkeit.

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Ein Mikroprogrammsteuerwerk im Blockschaltbild



– Mikroprogrammierung und Mikroprogrammsteuerwerke –



– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Mikrobefehlsformate

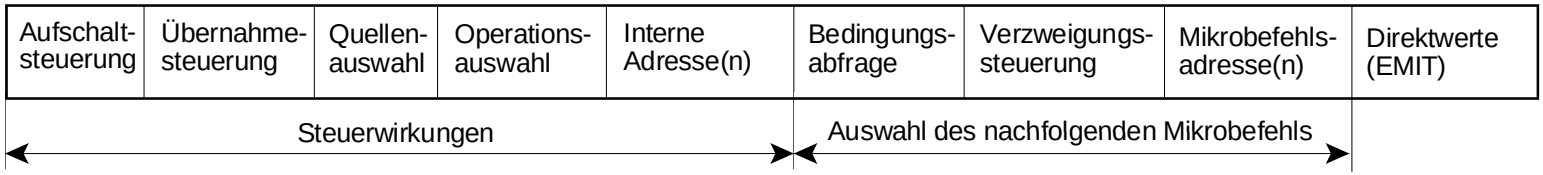
Die grundsätzlichen Mikrobefehlsformate werden mit den bildhaften Begriffen “horizontal” und “vertikal” bezeichnet.

a) Grundformat eines Mikrobefehls

Steuerwirkungen	Auswahl des nachfolgenden Mikrobefehls	Direktwerte (EMIT)

b) Horizontales Mikrobefehlsfomat

Ein Mikrobefehlsformat heißt horizontal, wenn alle Steuerwirkungen, die in einem Maschinenzyklus überhaupt ausgeführt werden können, in einem einzigen Mikrobefehl untergebracht sind.



c) Vertikales Mikrobefehlsformat

Operationscode	Anweisungs- und Direktwertfelder

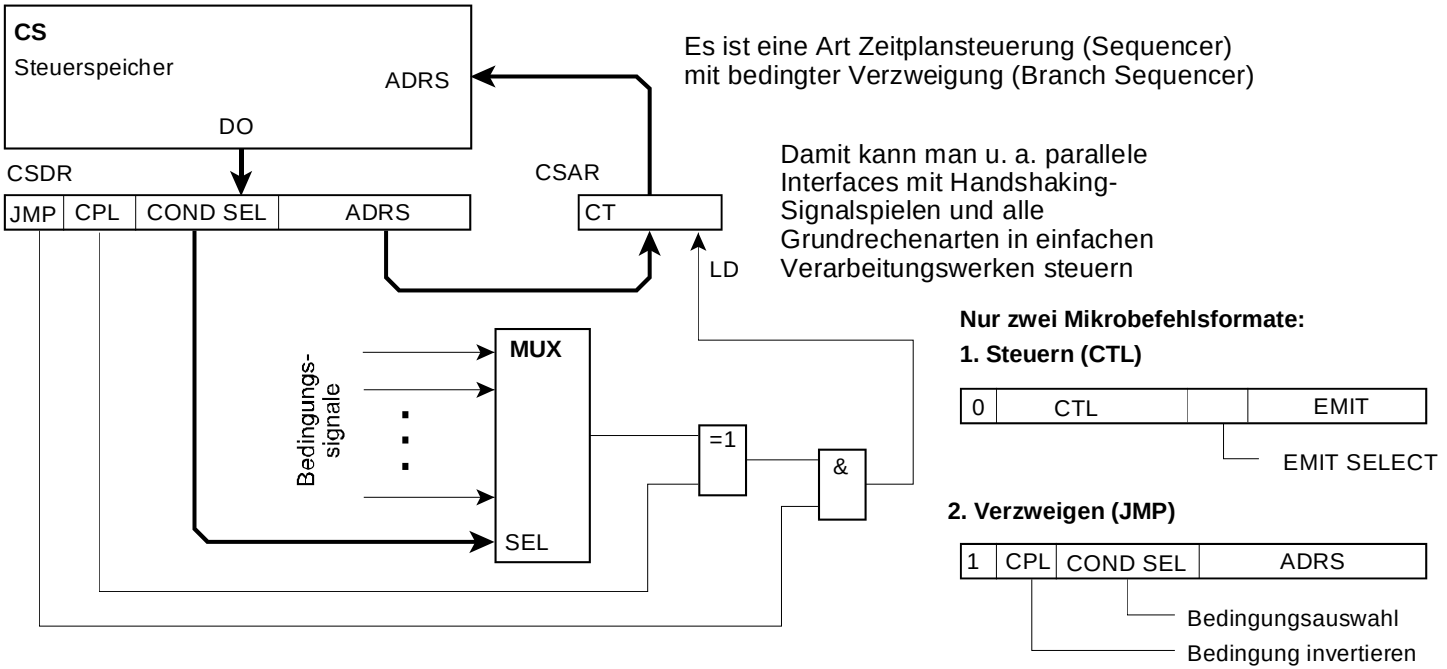
Vertikale Mikrobefehle sind kürzer (Richtwert: 12...32 Bits) und erbringen jeweils nur einige Steuerwirkungen. Die extreme Auslegung: jeder Mikrobefehl hat jeweils nur eine einzige Wirkung (Abfrage, Operandenverknüpfung, Verzweigung usw.). Solche Mikrobefehle ähneln den Maschinenbefehlen der herkömmlichen Mikrocontroller und RISC-Prozessoren.

d) Naheliegende vertikale Mikrobefehlsformate

CTL	Aufschalt-steuerung	Übernahme-steuerung	Quellen-auswahl	Operations-auswahl	Interne Adresse(n)	**Control**	**Steuerwirkungen, Verknüpfungen, Transporte**
ALF	Aufschalt-steuerung	Übernahme-steuerung	Quellen-auswahl	Operations-auswahl	Direktwert (EMIT)	**Arithmetical/ Logical Functions**	**Steuerwirkungen, Verknüpfungen, Transporte mit Direktwert**
JMP	Bedingungs-abfrage	Verzweigungs-steuerung	Mikrobefehls-adresse(n)			**Jump**	**Verzweigung, Unterprogrammruf**

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Für viele Steuerungsaufgaben genügen bereits sehr einfache Mikroprogrammsteuerwerke:

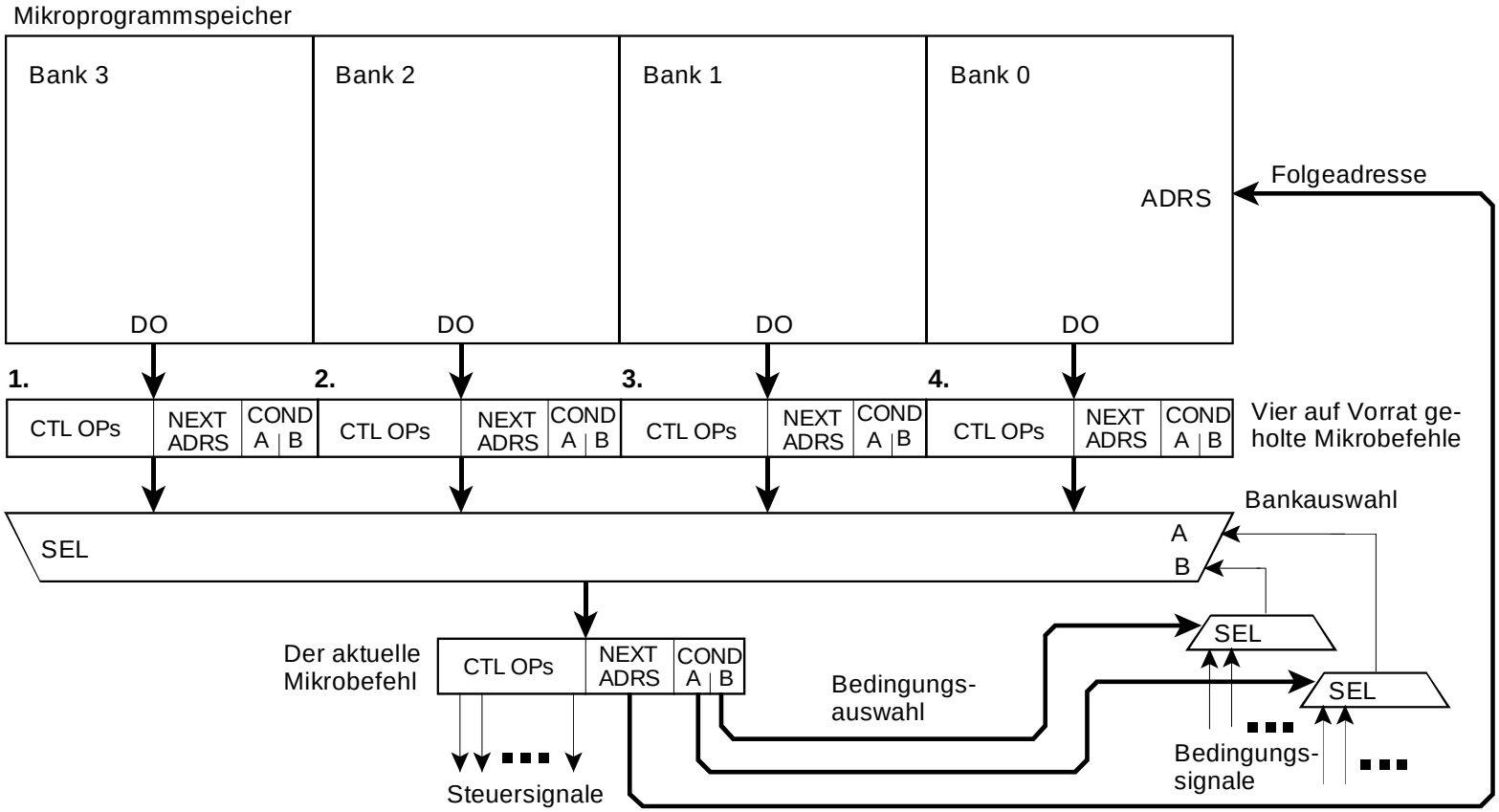


- Das Mikroprogrammsteuerwerk kann in jedem Taktzyklus Signale abfragen und Steuerwirkungen ausüben. Somit ergeben sich sehr geringe Latenz- und Reaktionszeiten.
- Wird das Steuerwerk konsequent auf Leistung ausgelegt, so kann es alle Steuersignale auf einmal erregen, alle Bedingungen gleichzeitig auswerten und in jedem Taktzyklus in mehrere Richtungen verzweigen – wenn es sein muß, aufgrund von Bedingungen, die sich im selben Taktzyklus ergeben haben.
- Mit der Mikroprogrammsteuerung als Architekturprinzip kann man richtige, von Grund auf für ihre Aufgabe optimierte Emulatoren für hardware-unabhängige fiktive Maschinen bauen (JVM, Dalvik usw.).
- Unterprogrammaufrufe und Programmumschaltungen (als Alternative zum üblichen Interrupt) kann man so implementieren, daß sie keine zusätzlichen Maschinenzyklen (Overhead) kosten.

– Es folgen zwei Beispiele höher entwickelter Prinziplösungen –

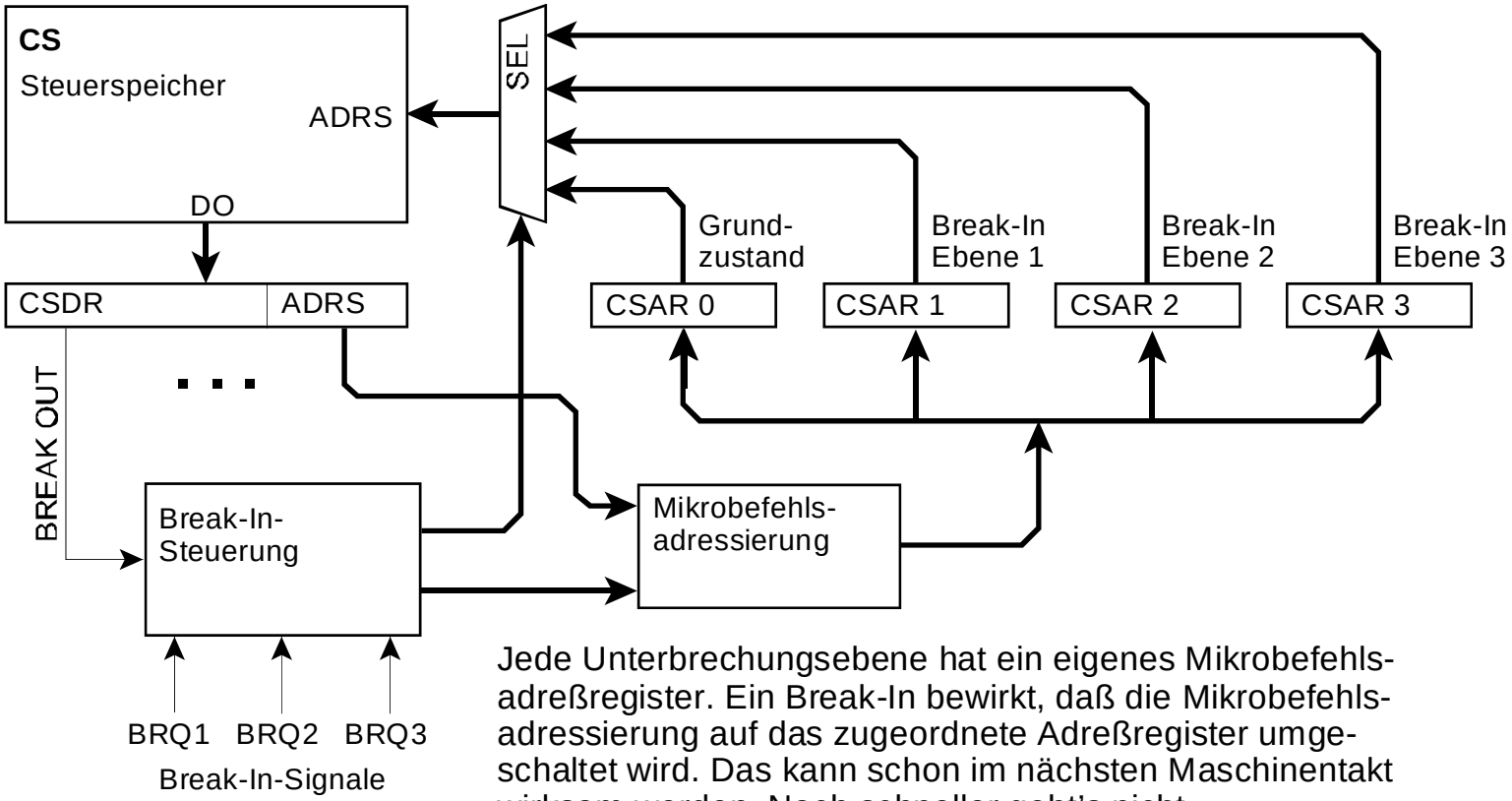
– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Und jetzt das High End: Spätverzweigung in vier Richtungen



Es werden vier Mikrobefehle gleichzeitig gelesen. Zwei Bedingungssignale ergeben eine Zwei-Bit-Adresse, über die einer der vier Mikrobefehle ausgewählt wird. Ein wichtiges Ziel des Schaltungsentwurfs besteht darin, die Bedingungen so spät wie möglich auszuwerten. Ein so entworfenes Mikroprogrammsteuerwerk kann beispielsweise auf die Übertragsbedingung einer im aktuellen Zyklus ablaufenden Addition noch ohne Zeitverlust reagieren.

Break-Ins: Unterbrechungen ohne Overhead

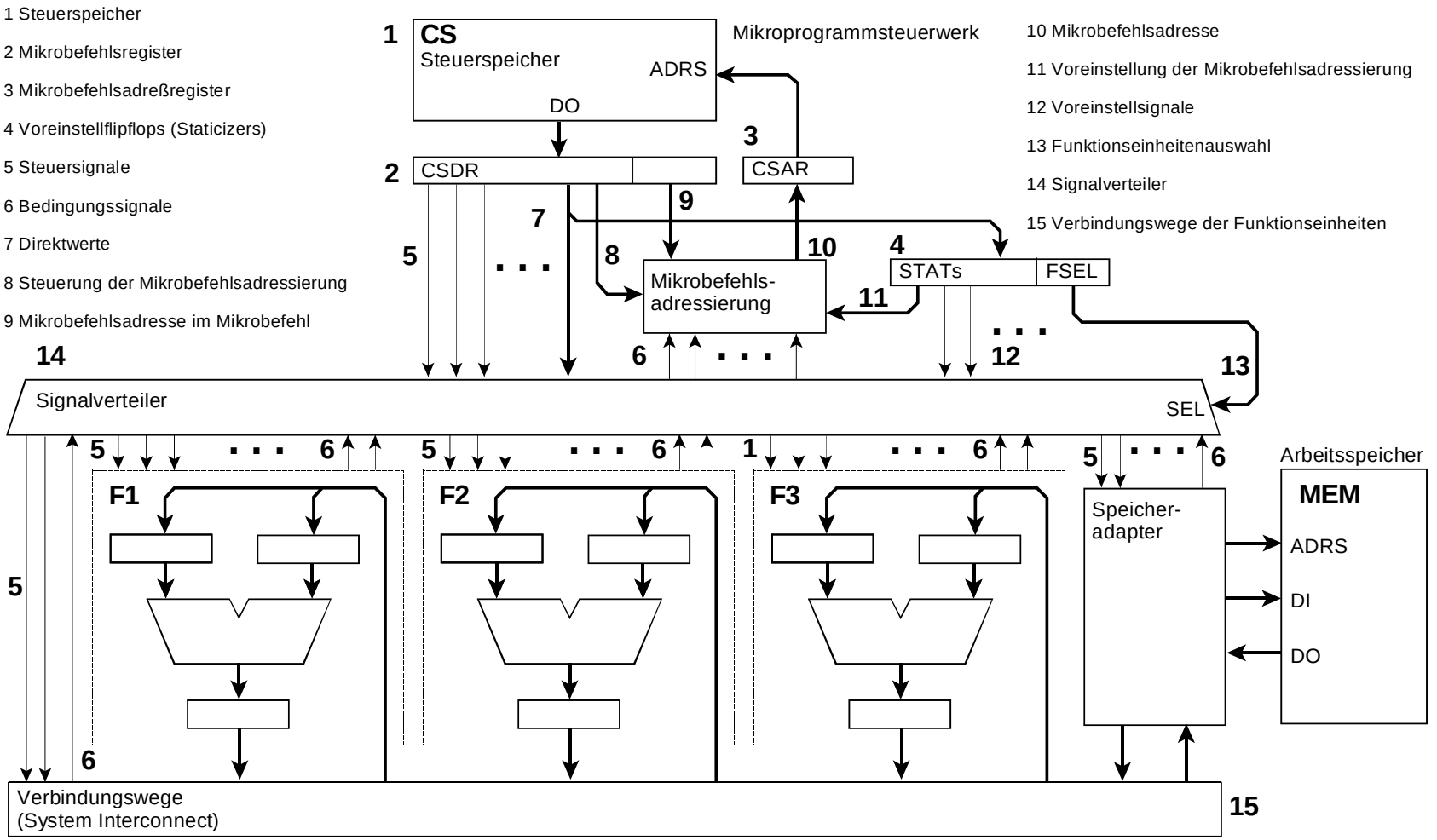


Jede Unterbrechungsebene hat ein eigenes Mikrobefehls-adreßregister. Ein Break-In bewirkt, daß die Mikrobefehls-adressierung auf das zugeordnete Adreßregister umgeschaltet wird. Das kann schon im nächsten Maschinentakt wirksam werden. Noch schneller geht's nicht ...

– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Das Mikroprogrammsteuerwerk im FPGA-Design

Es ist eine Art Dirigent der Anwendungsschaltungen. Es dient dazu, Komplexität beherrschbar zu machen. Die einzelnen Anwendungsschaltungen sind nicht allzu komplex, so daß sie bequem mittelsVerhaltensbeschreibung entworfen werden können. Deren Zusammenspiel wird vom Mikroprogrammsteuerwerk organisiert, das als optimierte Schaltungsstruktur entworfen, ggf. sogar als harter IP Core eingebaut wird.



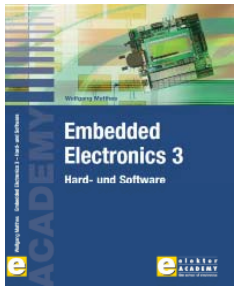
– Mikroprogrammierung und Mikroprogrammsteuerwerke –

Zusammenfassung

- Das übergeordnete Thema: Schaltungsentwurf oder Programmierung?
- Diese Grundsatzfrage ist nicht definitiv zu entscheiden, sondern immer wieder von neuem zu untersuchen.
- Die weiteren Darlegungen waren darauf gerichtet, das Programmieren gegenüber dem Schaltungsentwurf zu bevorzugen. Grundgedanke: alles so soft wie möglich, anwendungsspezifische Schaltungen nur dann, wenn es beim besten Willen nicht anders geht.
- Programmieren ist stets Intuition, Hacken, Schustern und Probieren (mögen die akademischen Lehrwerke doch schreiben, was sie wollen). Die natürliche Intelligenz kann von Anfang an zur Wirkung kommen und somit die Besonderheiten der Anwendungsaufgaben ausnutzen, um sich die Arbeit zu erleichtern.
- Die Schaltungssynthese hingegen beruht stets auf Booleschen Algorithmen mit NP-vollständiger Komplexität, ganz gleich, welchen Komfort die Entwicklungsumgebung anbietet (Verhaltensbeschreibung, Nutzung üblicher Programmiersprachen).
- Deshalb: Lösungen finden, um auch im Grenzbereich programmieren zu können (also wenn man herkömmlicherweise eher zum Schaltungsentwurf hinneigen würde).
- Dafür braucht man aber entsprechende Plattformen. Die herkömmliche Lösung: RISC-Prozessorkerne + Beschleunigungsschaltungen (Akzeleratoren).
- Der grundsätzliche alternative Vorschlag: die altbewährten Prinzipiösungen der Mikroprogrammierung neu beleben.
- Es dürfte alles frei sein (Patente längst abgelaufen)*.

*: Vorsicht. Aussage ohne Gewähr.

In diesen Schriften wurden die Themen bereits angesprochen:



(Monographie zur Mikroprogrammierung in gleicher Ausstattung vorgesehen.)