– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Mikroprogrammsteuerwerke haben ihre Vorteile. Schließlich ist es eine altbewährte Technik. Man müßte sie nur aufs FPGA übertragen. Praktiker erkennen aber sofort das Grundproblem: Wie aber – und vor allem: womit – soll man sie programmieren?

- Heutige Anwendungsaufgaben erfordern soviel Software, daß man sie einfach nicht mehr in Assembler schreiben kann. Compiler für Mikroprogrammsteuerwerke gibt es, es ist aber ein sehr spezielles Thema für sich.
- Deshalb wird man zumeist auf verbreitete Prozessorarchitekturen und Entwicklungsumgebungen zurückgreifen.
- Wenn das Leistungsvermögen des Prozessors nicht reicht und mehrere Prozessoren auch keine brauchbare Lösung sind, fügt man Beschleunigungsschaltungen (Akzeleratoren) hinzu (Hardware-Software Co-Design).
- Auf diese Weise könnte man auch Mikroprogrammsteuerwerke in die Anwendungslösung einbauen.
- Das Zusammenwirken mehrerer programmierbarer Einrichtungen hat aber auch eigene Probleme und Entwurfsschwierigkeiten.

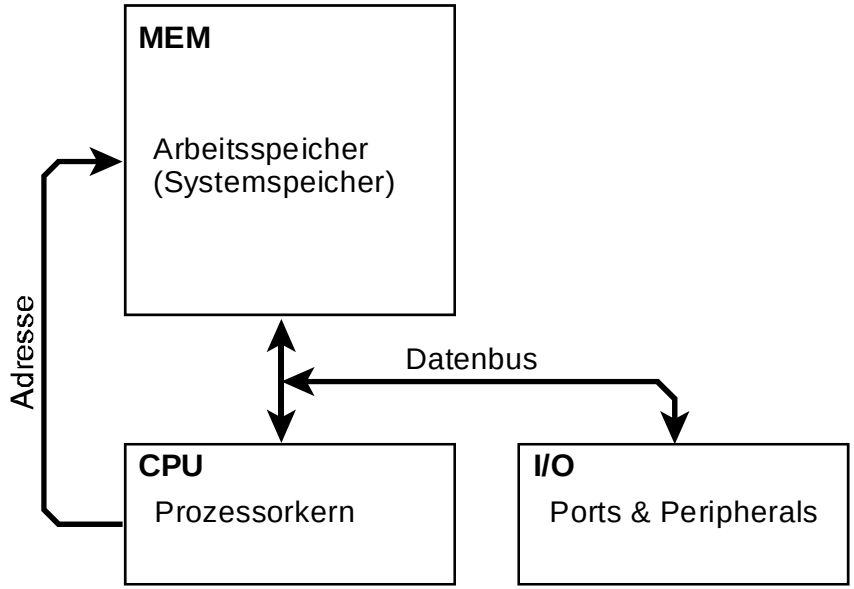
Eine alternative Design-Idee:

- Wir bleiben bei einem bewährten Prozessor und seiner Entwicklungsumgebung. Die weitaus meiste Software läuft auch schnell genug, da muß nichts beschleunigt werden. Um die verbleibenden, wirklich leistungskritischen Programmstücke schneller auszuführen, werden die Befehle außerhalb des Prozessors verlängert.
- Der Prozessor selbst bleibt, wie er ist. Im Innern wird nichts geändert.
- Dem Programmspeicher wird ein Erweiterungs- oder Steuerspeicher hinzugefügt. Die Verlängerungen lösen zusätzliche Steuerwirkungen aus. Manche weisen an, wie der Befehl auf dem Wege vom Speicher zum Prozessorkern verändert oder ersetzt wird.
- Der Prozessor wird zeitweilig zum spezialisierten Mikroprogrammsteuerwerk. Die auf diese Weise ausgeführten Befehle werden außerhalb des Prozessorkerns zu Mikrobefehlen.
- Alle Programme, die mit diesen Funktionen nichts zu tun haben, bleiben, wie sie sind.

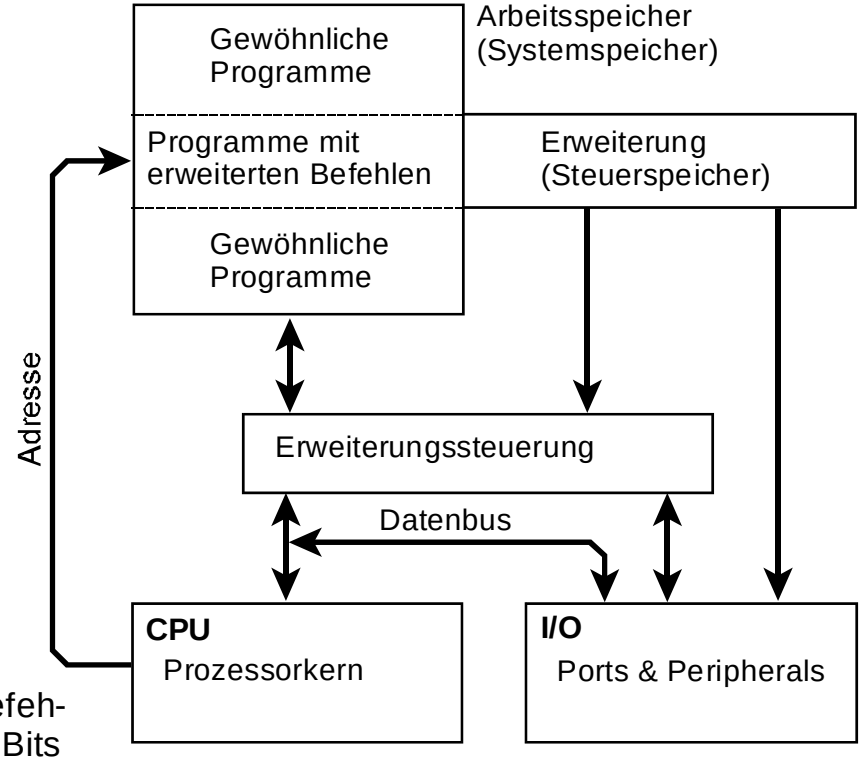
– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

a) Herkömmliches Mikroprozessorsystem



b) Mikroprozessorsystem mit Befehlserweiterung

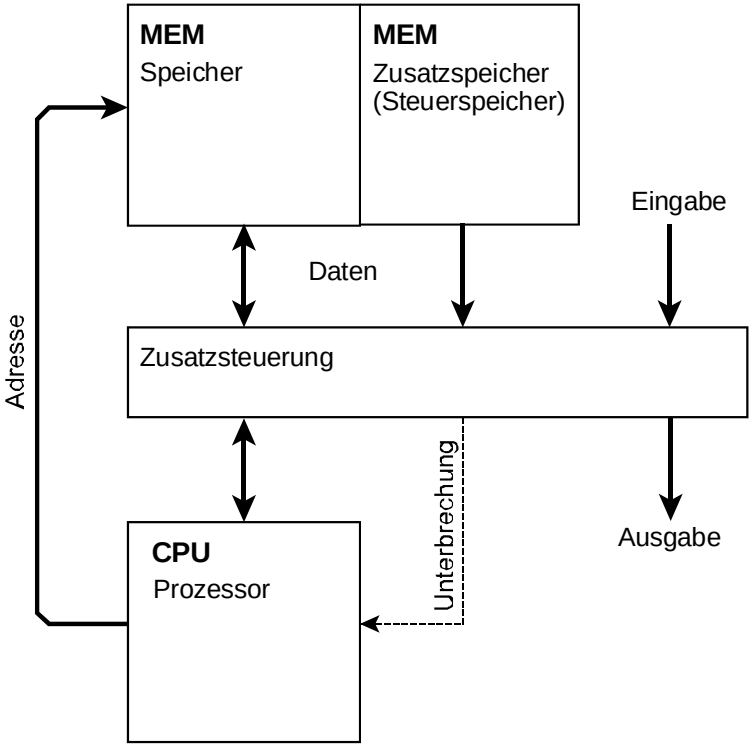


- Die erweiterten Befehle ähneln den horizontalen Mikrobefehlen. Die Befehle des Prozessorkerns können um soviele Bits verlängert werden wie erforderlich.
- Erweiterte und gewöhnliche Befehle können beliebig gemischt werden.
- Im Gegensatz zu einem mikroprogrammgesteuerten Prozessor eigener Architektur verhindert die Erweiterung nicht die Nutzung vorhandener Software und Entwicklungssysteme.

– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Befehlserweiterung mittels Zusatzspeicher (Steuerspeicher)

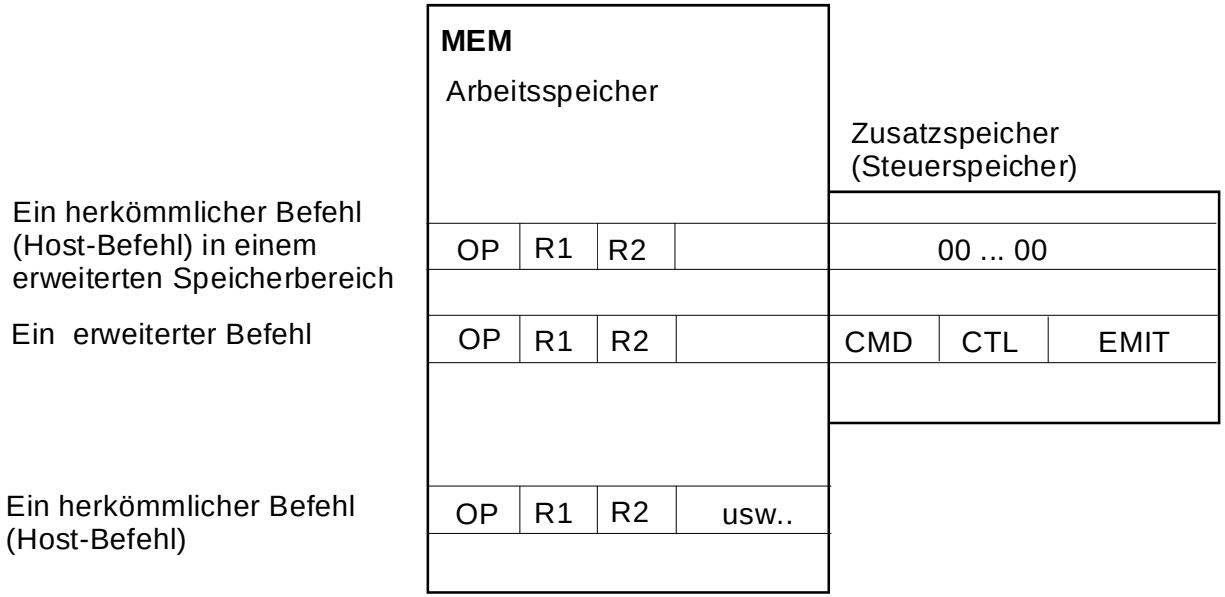


- Der Entwurfsgedanke: den Prozessorkern mit seiner Infrastruktur und seinem Ökosystem beibehalten, aber mit Zusatzschaltungen ergänzen, die auf die Befehlslesewege einwirken. Alle Programme, die solche Beschleunigungsmaßnahmen nicht benötigen, können bleiben, wie sie sind, und auch mit den üblichen Programmiersprachen und Entwicklungswerkzeugen programmiert werden.
- Die Grundsatzlösungen wurden in den 80er Jahren gefunden.
- Das Problem war damas das gleiche: Der Mikroprozessor ist gegeben. Eine große Menge an Software ist gegeben oder neu zu schreiben, der Mikroprozessor ist aber für manche Funktionen nicht leistungsfähig genug.
- Nun könnte man das alles mit speziellen Schaltungen erledigen. Mit TTL-Schaltkreisen hätte sich aber ein geradezu gigantischer Aufwand ergeben. Auch wäre dann jede Änderung eine Schaltungsänderung. Deshalb das Entwicklungsziel: alles so soft wie möglich.
- Es funktioniert mit allen Prozessorkernen, bei denen man an den Befehlslesweg herankommt, die also auf dem weiteren Weg bis ins Befehlsregister nichts mehr zwischenpuffern.
- Mit den herkömmlichen 8-Bit- Mikroprozessoren funktioniert es, mit dem 8086 würden nicht alle Schaltungen funktionieren, weil der einen internen Befehlspuffer hat und immer 6 Befehlsbytes auf Vorrat liest.
- Mit Soft Cores (MicroBlaze, NIOS usw.) sollte es sich einrichten lassen.
- Mit ARM, MIPS usw. dürfte man es hinbekommen, sofern man einen geeigneten Prozessorkern aussucht.
- Prozessorkerne mit tiefen Pipelines und spekulativer Befehlsausführung sind ohnehin nicht besonders geeignet, wenn es darum geht, mit der Außenwelt zusammenzuwirken.

– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Befehlserweiterung mittels Zusatzspeicher (Steuerspeicher)



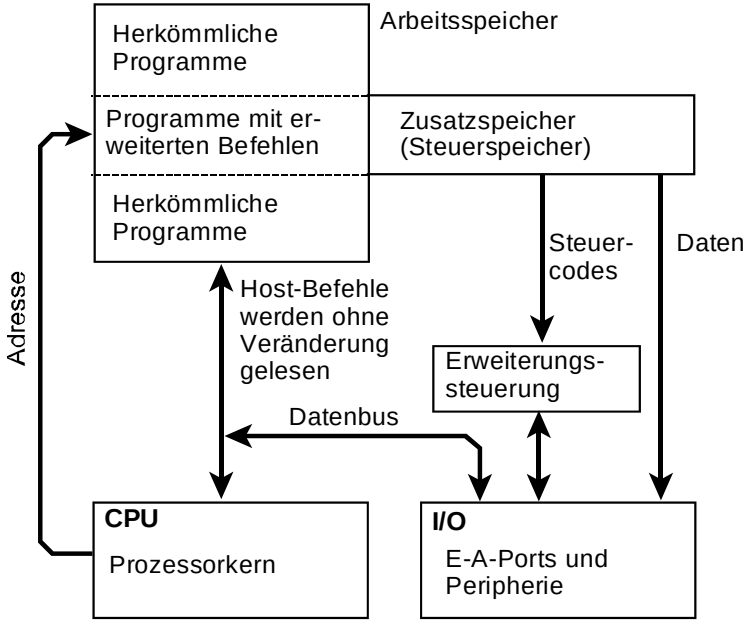
Beispiele erweiterter Befehlswirkungen:

- Vergleichsstop-Funktionen unterstützen (mit unbegrenzt vielen Hardware-Breakpoints)
- Unterbrechungen zeitweilig verhindern
- Daten ausgeben
- Zusatzschaltungen aktivieren, beispielweise Beschleunigungseinheiten (Akzeleratoren) oder anwendungsspezifische Peripherie
- Auf Bits der Ein- und Ausgabe verzweigen
- Befehle in Abhängigkeit von Bedingungen ausführen (Predication)
- Alternative Befehle ausführen
- Befehlsausgabe
- Zusatzeingabe
- Bedingte Befehlsausführung
- Funktionsverzweigung
- EXECUTE-Befehle
- Die vom Prozessor gelieferten Adressen als zusätzliche Ausgabedaten verwenden

– Der Prozessorkern als Mikroprogrammsteuerwerk –

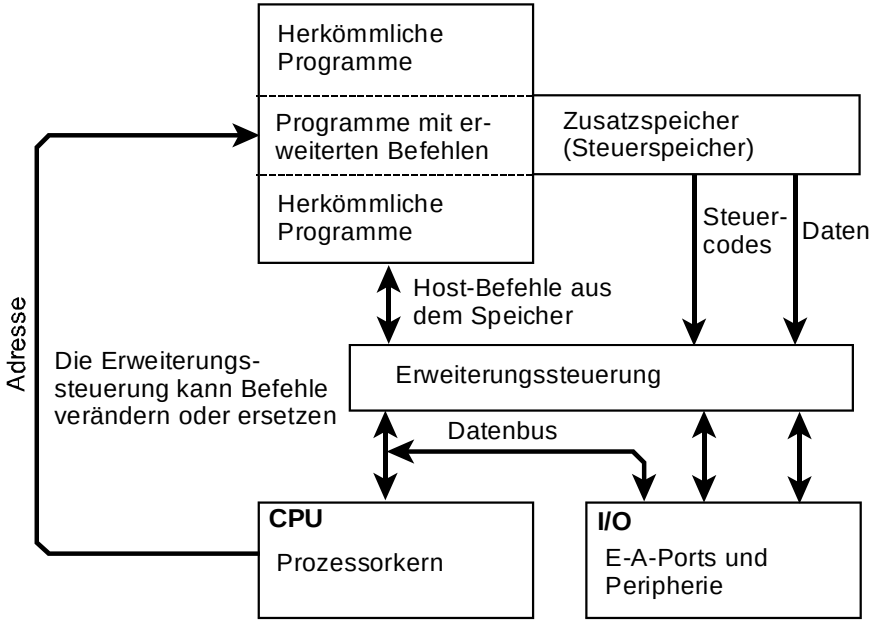
– Eine Design-Idee –

a) Erweiterung ohne Verändern der Host-Befehle



Es ist ein üblicher Datenzugriff zu einer peripheren Einrichtung. Die Zugriffsadresse ergibt sich aber bereits beim Befehlslesen. So können die Datenwege durchgeschaltet werden, ehe der Datenzugriff des Befehls beginnt. Keine Verlangsamung, keine zusätzlichen Wartezustände o. dergl.

b) Erweiterung mit Verändern der Host-Befehle



Die Befehlsveränderung ergibt eine größere Schaltungstiefe beim Befehlslesen. Das kann dazu zwingen, die Taktfrequenz herabzusetzen oder Wartezustände einzufügen.
Die Zusatzschaltungen im Befehlsleseweg sind aber im Grunde nur Auswahlschaltungen. Es muß also nicht so schlimm werden. U. a. könnte man Transfer-Gates einsetzen. Im FPGA (Softcore-Implementierung) kann es auch sein, daß die Auswahlfunktionen noch in den ohnehin vorhandenen Logikzellen des Befehlslesewegs unterzubringen sind.

– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Anwendungsbeispiele:

- Eine Beschleunigungsschaltung (Akzelerator) anschließen
- Vergleichsstop
- Zusatzausgabe
- Nebenläufige Ausgabe
- Adreßausgabe
- Befehlsausgabe
- Zusatzeingabe
- Bedingte Befehlsausführung
- Befehlssubstitution (Funktionsverzweigung, EXECUTE-Befehle)

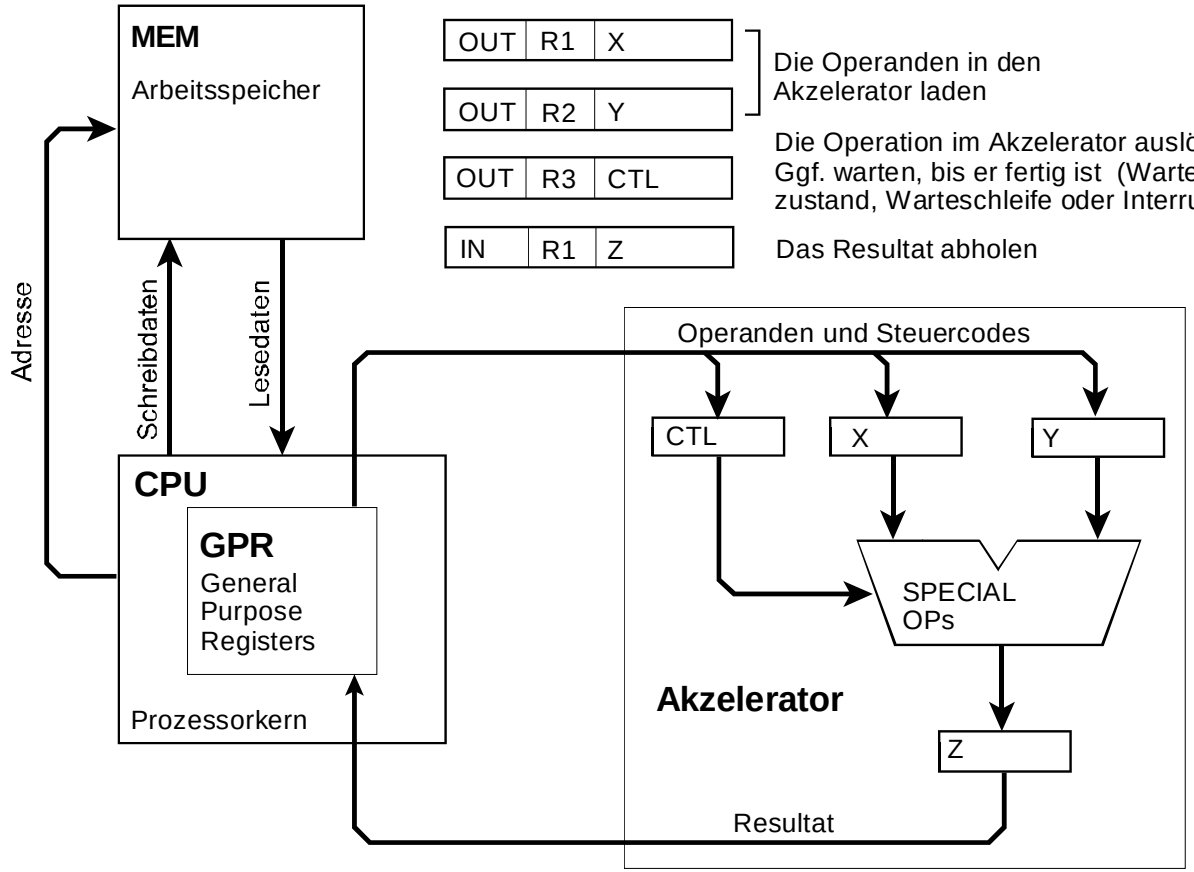
– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

So wirkt ein einfacher Akzelerator (im FPGA) mit dem Host-Prozessor zusammen (vgl. die einschlägigen Applikationsschriften)

Maschinenbefehle

LD	R1	ADRS	Die CPU-Register laden
LD	R2	ADRS	
LDI	R3	LITERAL	
OUT	R1	X	Die Operanden in den Akzelerator laden
OUT	R2	Y	
OUT	R3	CTL	Die Operation im Akzelerator auslösen. Ggf. warten, bis er fertig ist (Wartezustand, Warteschleife oder Interrupt)
IN	R1	Z	Das Resultat abholen



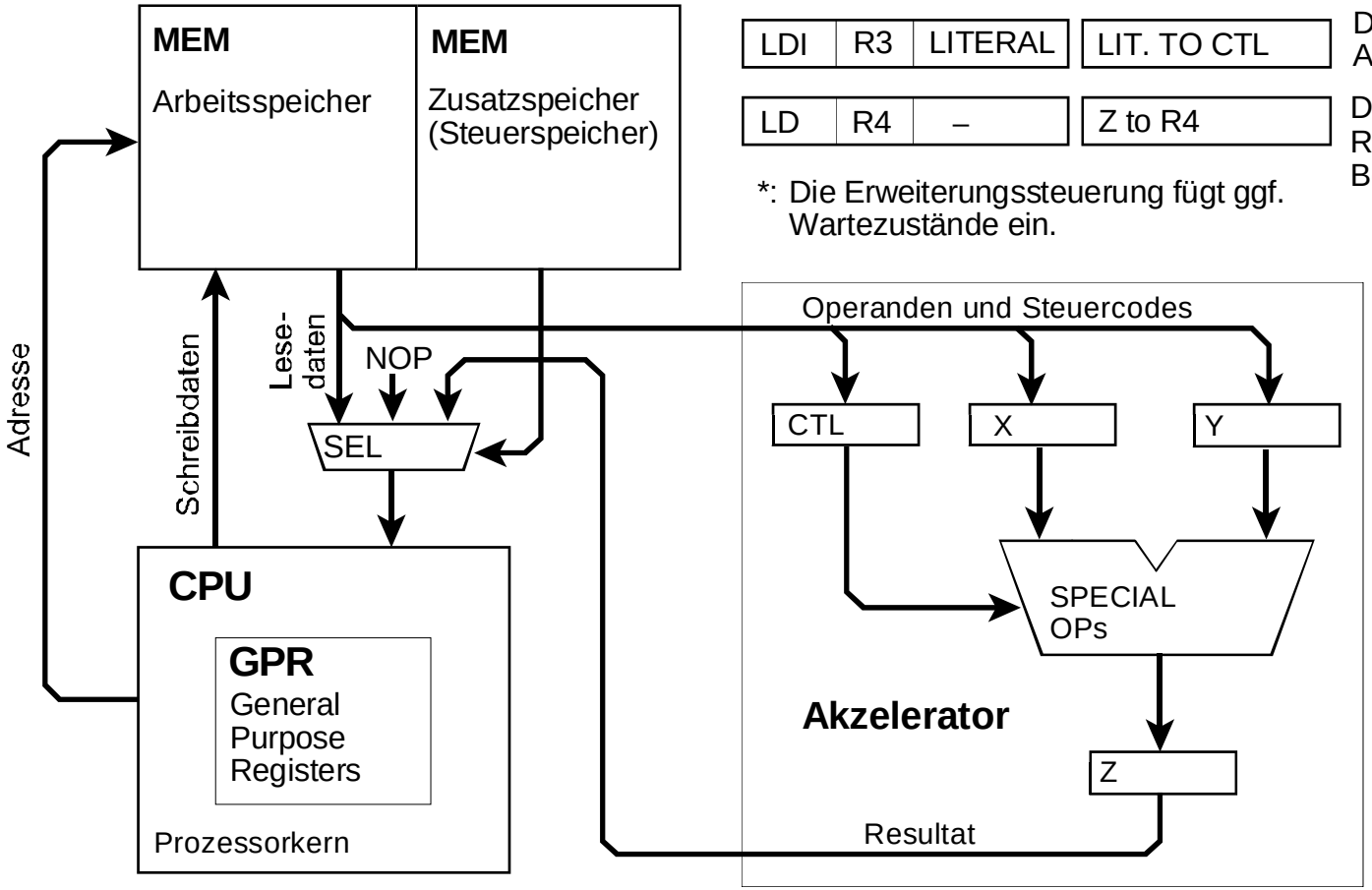
– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

So kann ein einfacher Akzelerator (im FPGA) mit erweiterten Befehlen unterstützt werden

Maschinenbefehle			Erweiterung	
LD	R1	ADRS	LOAD X	Die CPU empfängt Kopien der Daten oder führt NOPs aus
LD	R2	ADRS	LOAD Y	
LDI	R3	LITERAL	LIT. TO CTL	Die Operation im Akzelerator auslösen*
LD	R4	–	Z to R4	Die CPU empfängt das Resultat. Die Adresse im Befehl wird ignoriert.

*: Die Erweiterungssteuerung fügt ggf. Wartezustände ein.

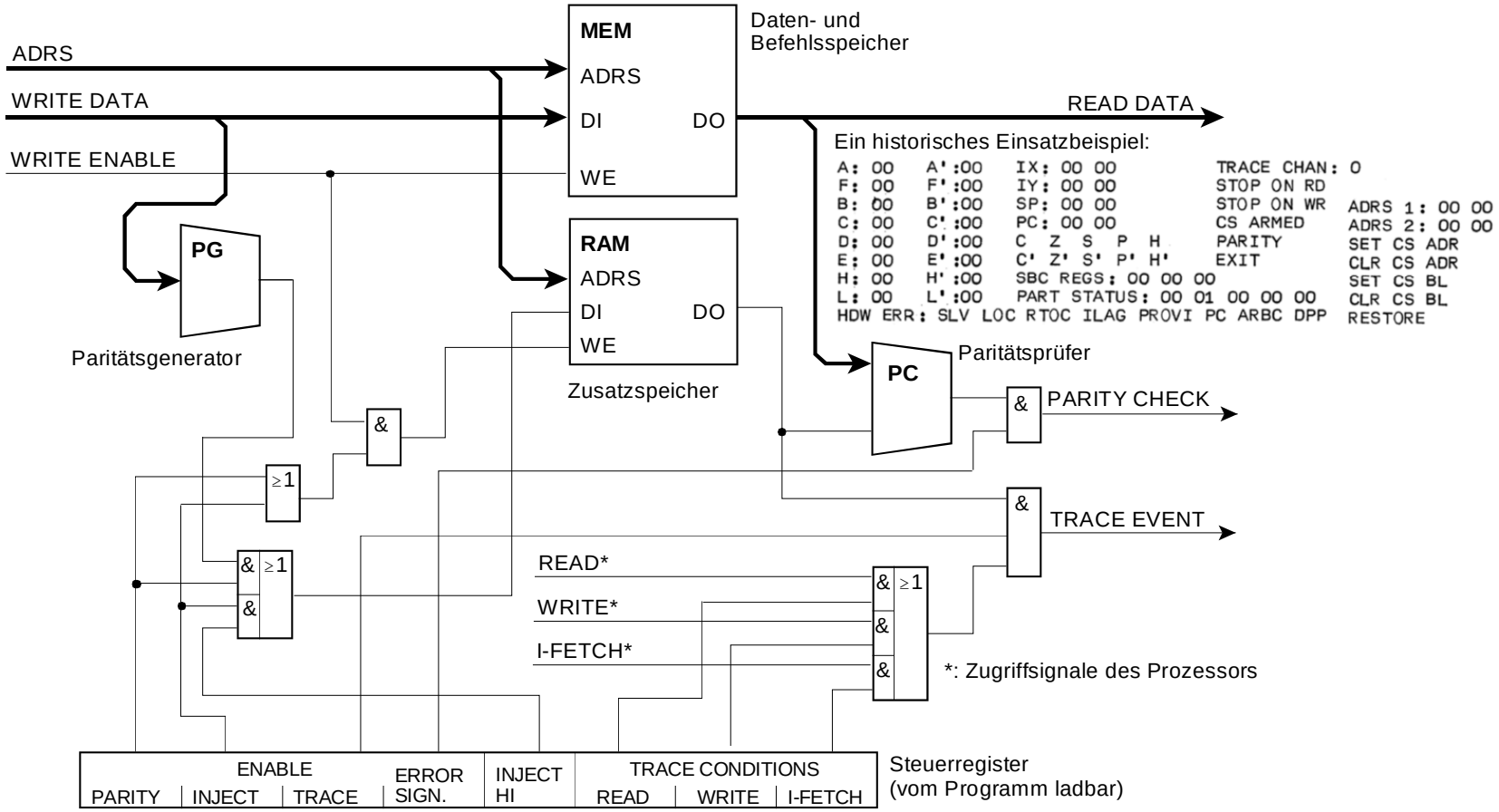


– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Zusatzspeicher unterstützt den Vergleichsstop

Eine Bitposition im Zusatzspeicher bewirkt, daß ein Vergleichsereignis signalisiert wird. Es kann u. a. eine Unterbrechung auslösen. Auf dieser Grundlage kann man eine Vergleichsstop- oder Breakpoint-Funktion implementieren. Wenn man den gesamten Speicher so erweitert, kann man beliebig viele Breakpoints setzen, bis hin zum befehlsweisen Betrieb (Schrittbetrieb). Demgegenüber unterstützen die typischen eingebauten Breakpointvorkehrungen der Mikrocontroller nur wenige Breakpoints (beispielsweise vier). Im Normalbetrieb kann der Zusatzspeicher zur Paritätskontrolle oder Fehlerkorrektur (ECC) verwendet werden.

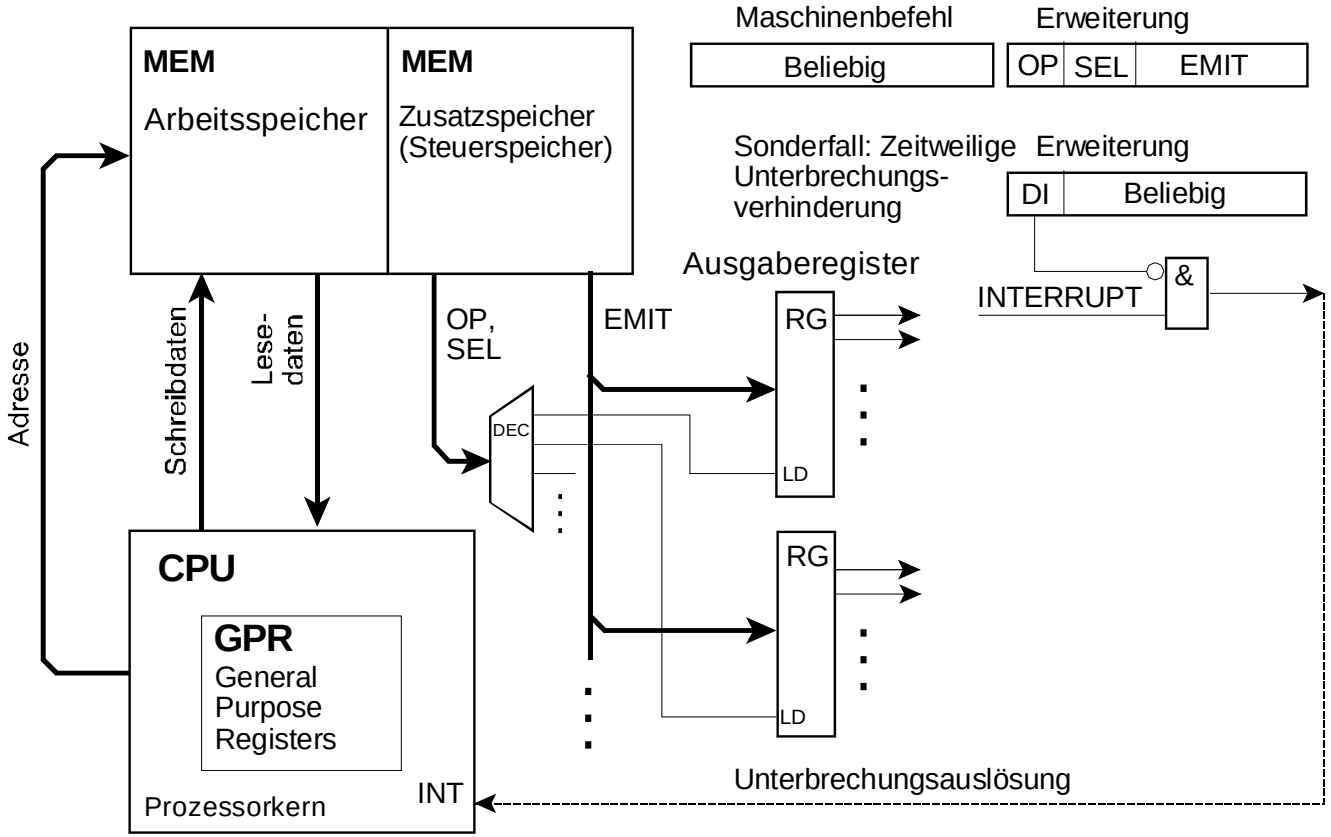


– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Zusatzausgabe

Parallel zur Befehlswirkung im Prozessor wird ein im Zusatzspeicher befindliches Bitmuster ausgegeben. Es versteht sich von selbst, daß man Bitpositionen des Zusatzspeichers verwenden kann, um beliebige weitere Funktionen auszulösen.
Zeitweilige Unterbrechungsverhinderung: Nutzung in Programmabschnitten, die nicht unterbrochen werden dürfen. Der allgemeine Erlaubniszustand des Prozessors, wie er durch Befehle EI, DI (Enable, Disable) eingestellt ist, bleibt dabei erhalten.

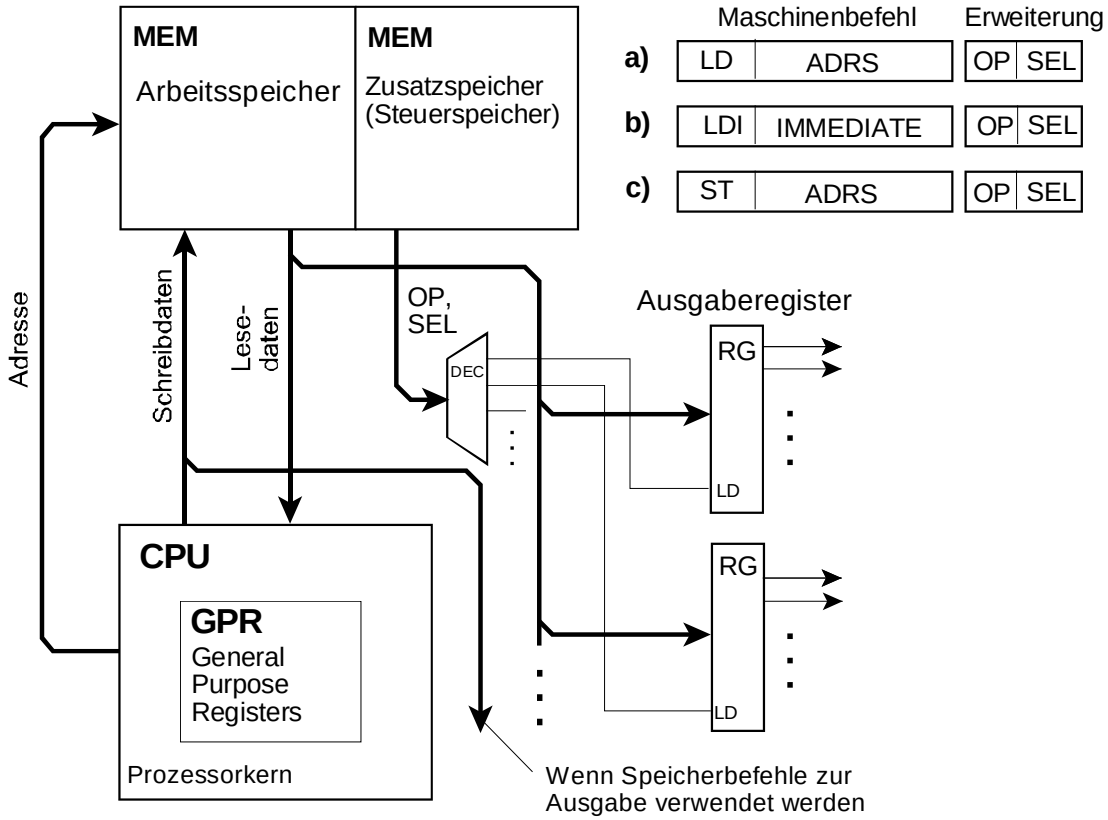


– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Nebenläufige Ausgabe

Lade- oder Speicherbefehle dienen zur Ausgabe. Die gelesenen Daten oder die Schreibdaten werden gleichsam abgezweigt. Eine Folge LOAD Register – OUT E-A- Port, Register verkürzt sich damit auf den Ladebefehl.



a) Laden der Ausgaberegister beim Datenzugriff (Laden mit den gelesenen Daten)
b) Laden der Ausgaberegister beim Befehlslesen (Laden mit dem Direktwert)
c) Laden der Ausgaberegister beim Datenzugriff. Gleichzeitiges Ausgeben und Speichern möglich (Anlegen einer Kopie im Speicher). Ggf. Speichern unterdrücken

– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

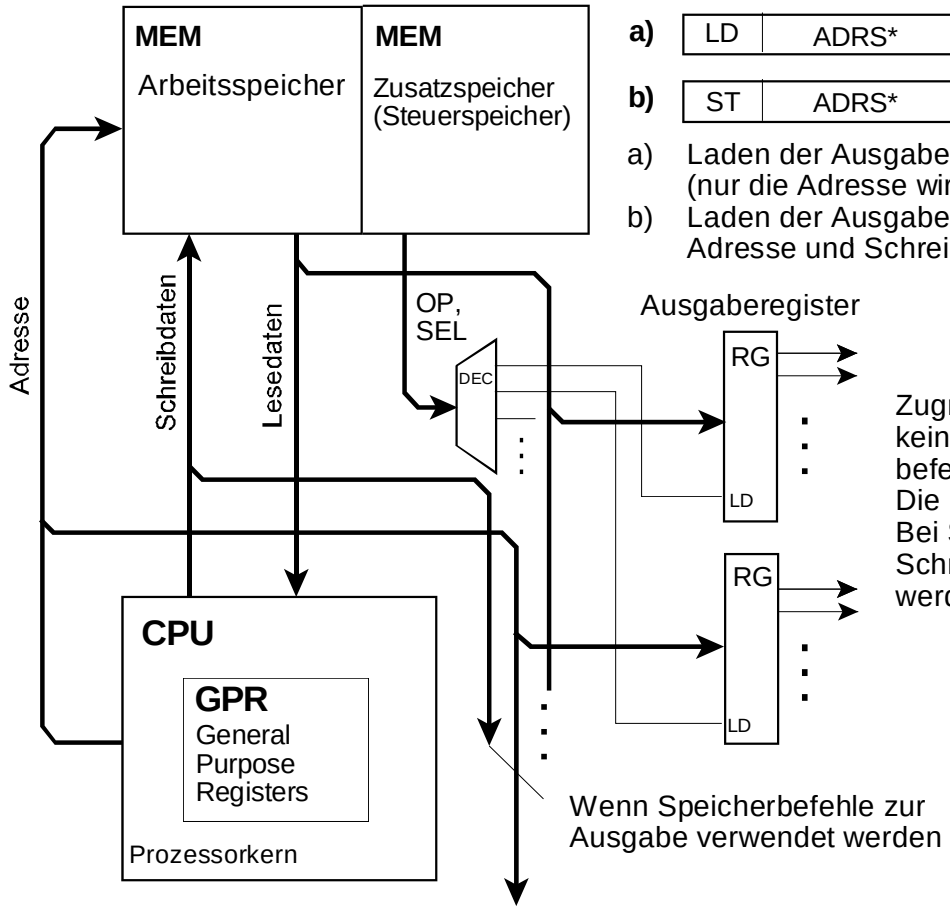
Adreßausgabe

Die während der Befehlsausführung gelieferte Datenadresse wird zur Datenausgabe genutzt. Die Adreßbits werden als Datenbits interpretiert. Auf diese Weise kann eine 8-Bit-Maschine mit einem einzigen Befehl 16 Bits ausgeben, wenn man das Datenbyte hinzunimmt, sogar 24 Bits.
Die ausgegebene Datenadresse ist eine effektive Adresse, also typischerweise nicht der Adreßwert im Befehl, sondern das Ergebnis einer Adreßrechnung (Basis + Displacement oder indirekte Adressierung)

	Maschinenbefehl		Erweiterung	
a)	LD	ADRS*	OP	SEL
b)	ST	ADRS*	OP	SEL

*: Relative Adresse, Adreßregisterauswahl

a) Laden der Ausgaberegister beim Lesezugriff (nur die Adresse wird ausgegeben)
b) Laden der Ausgaberegister beim Schreibzugriff. Adresse und Schreibdaten können ausgegeben werden



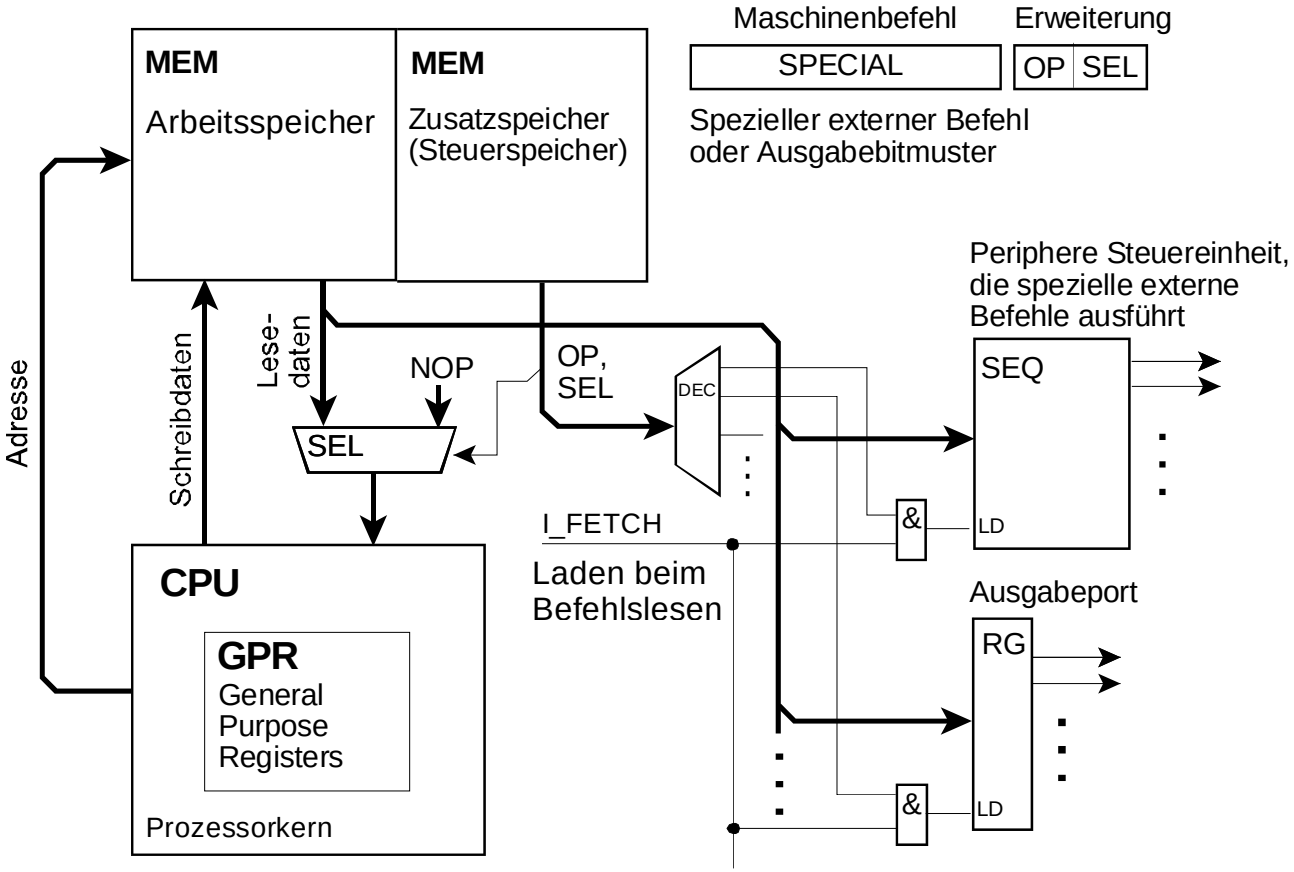
Zugriffe mit beliebigen Adressen dürfen keine Nebenwirkungen haben. Ein Lesebefehl bewirkt nur eine Adreßausgabe. Die gelesenen Daten sind gleichgültig. Bei Schreibzugriffen müssen aber die Schreibwirkungen im Speicher verhindert werden.

– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Befehlsausgabe

Die Bitmuster der Befehle werden ausgegeben. Dem Prozessor werden statt dessen NOP-Bitmuster zugeführt. Diese Funktion ermöglicht es, die Zugriffsbreite des Programmspeichers und den Befehlsleseablauf zu Ausgabezwecken zu nutzen. Das lückenlose Befehlslesen ist oftmals der schnellste Leseablauf des Prozessors.

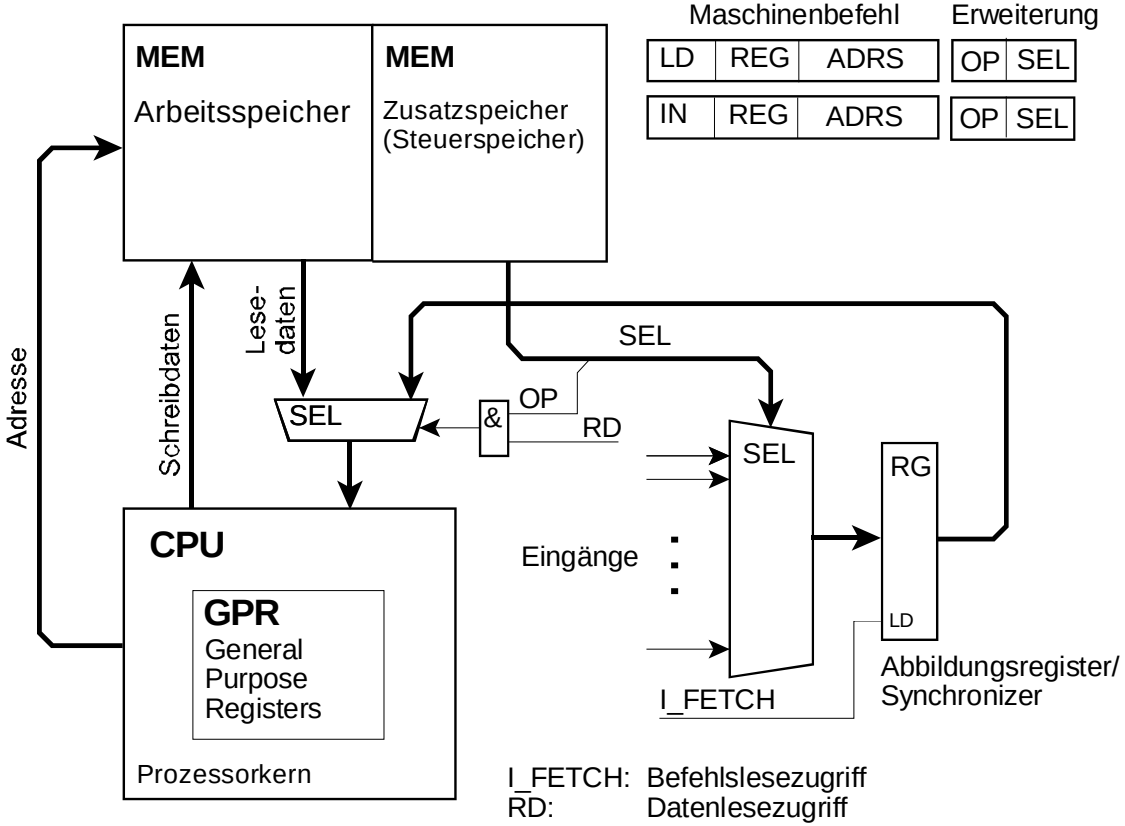


– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Zusatzeingabe (1) – Eingabe in Prozessorregister

In Lade- oder Eingabebefehlen wird anstelle des adressierten Inhalts ein Bitmuster gelesen, dessen Quelle über den Zusatzspeicher ausgewählt wird.
Das Bitmuster wird beim Datenzugriff in den Lesedatenweg zum Prozessor eingespeist.
Die vom Befehl gelieferte Datenadresse kann ignoriert oder zur Auswahl der Lesedaten oder zur Datenausgabe genutzt werden.

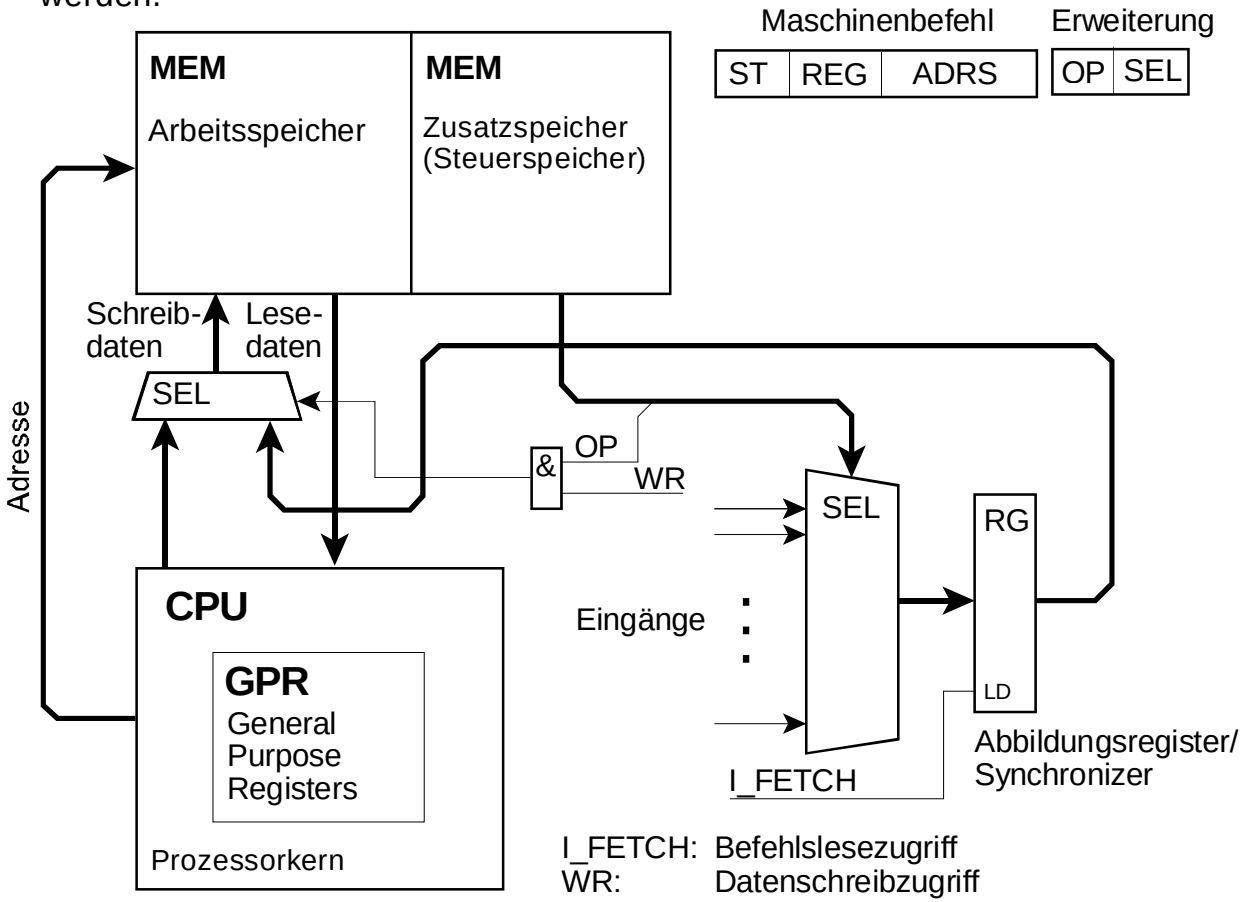


– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Zusatzeingabe (2) – Eingabe in den Speicher

In Speicherbefehlen wird anstelle des Datenworts, das vom Prozessor kommt, ein Bitmuster geschrieben, dessen Quelle über den Zusatzspeicher ausgewählt wird. Das Bitmuster wird beim Datenzugriff in den Schreibdatenweg eingespeist. Die vom Befehl gelieferten Schreibdaten können ignoriert oder zur Datenausgabe genutzt werden.

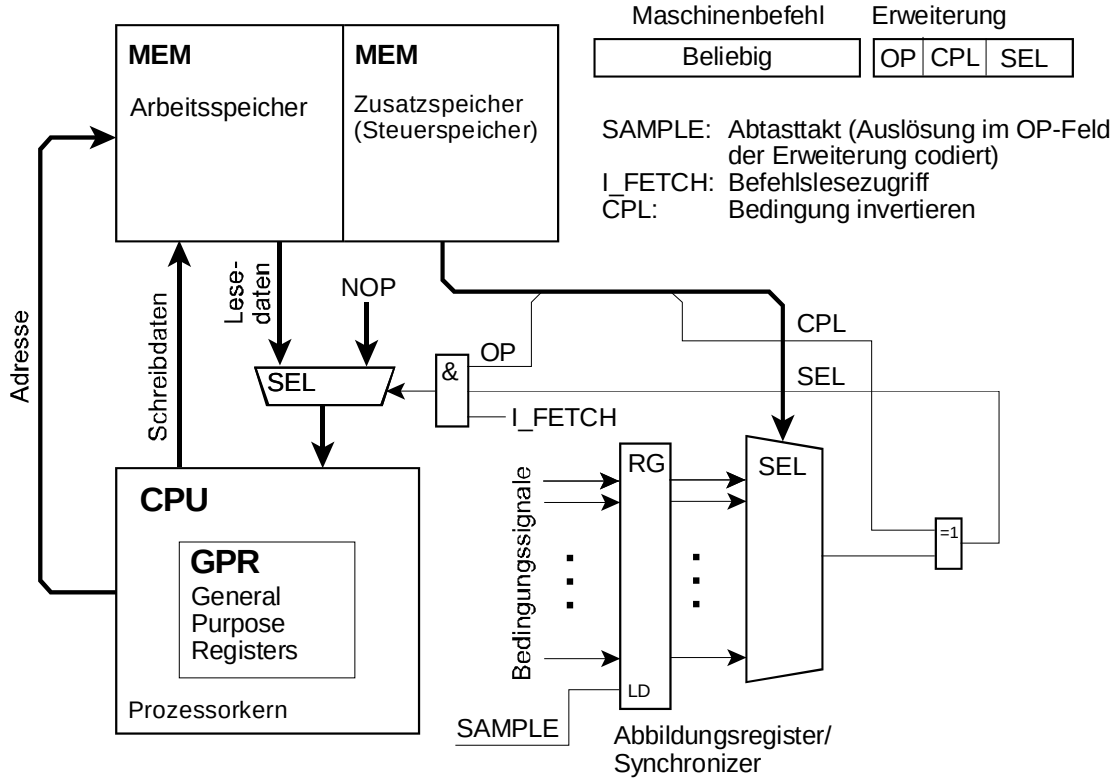


– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Bedingte Befehlsausführung

Die Befehle werden durch Bedingungsauswahlsignale ergänzt. Die Bedingungen werden der Umgebung des Prozessors oder der Außenwelt entnommen (Eingangssignale). Ist die Bedingung erfüllt, wird der aus dem Speicher gelesene Befehl zum Prozessor weitergeleitet. Ist sie nicht erfüllt, wird ein NOP aufgeschaltet.
Soll eine Reihe von Befehlen übergangen werden, ist es wichtig, daß die Bedingung nur am Anfang erfaßt und dann konstant gehalten wird.
Eine bedingte Verzweigung auf eine solche Bedingung wird mit einem unbedingten Verzweigungsbefehl, programmiert, der bei Nichterfüllung übergangen wird. Ein Warten auf eine solche Bedingung ist eine unbedingte Verzweigung auf sich selbst, die bei Erfüllung übergangen und bei Nichterfüllung ausgeführt wird.

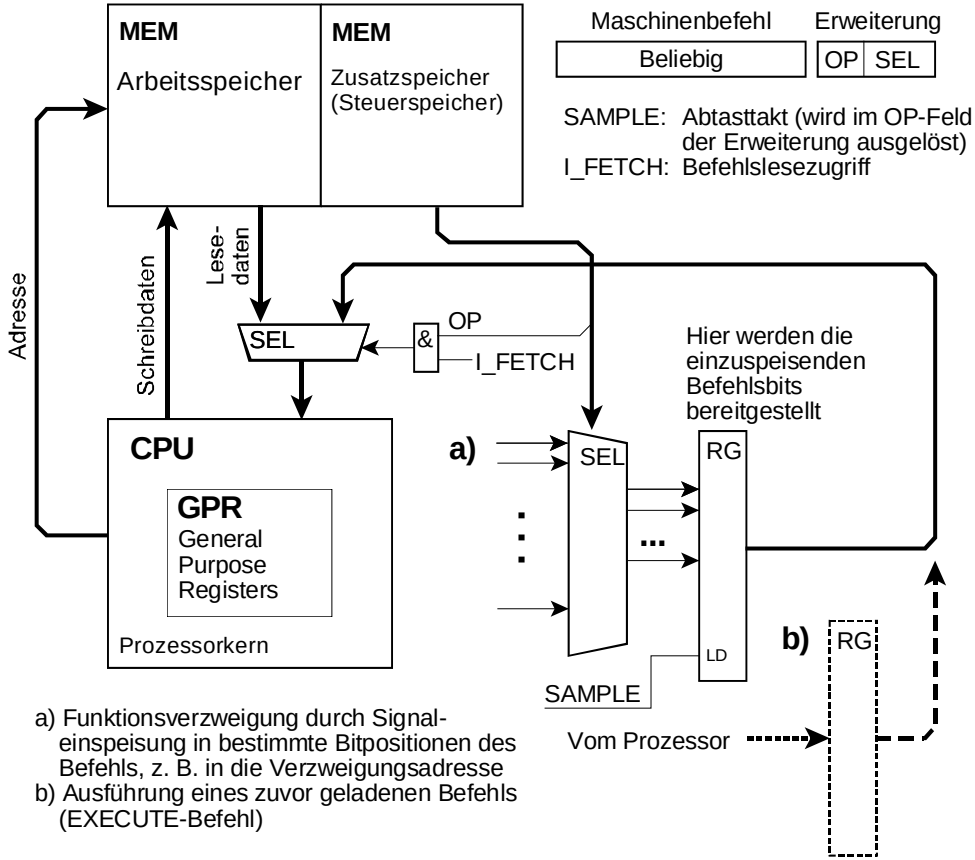


– Der Prozessorkern als Mikroprogrammsteuerwerk –

– Eine Design-Idee –

Befehlssubstitution (Funktionsverzweigung, EXECUTE-Befehl)

Wird die Zusatzeingabe während des Befehlslesens wirksam, kann man beliebige Befehle von außen einspeisen. Auch wäre es möglich, bei der bedingten Befehlsausführung dem Prozessor anstelle des NOP einen beliebigen anderen Befehl anzubieten. Auf diese Weise läßt sich das Prinzip des EXECUTE-Befehls implementieren. Eine weitere Variante ist die Funktionsverzweigung. Hierbei werden beim Befehlslesen bestimmte Bitpositionen des Befehlsformats auf ausgewählte Signale umgeschaltet. Die typische Nutzung besteht darin, die Verzweigungsadresse von Verzweigungs-oder Unterprogrammrufbefehlen abzwandeln. Ein solcher Befehl wird so zu einer Art Mikrobefehl, der zu einem von beispielsweise 8, 16, 32 usw. Nachfolgern verzweigen kann. So wäre es auch möglich, ein ganzes Kommandobyte (von einem Interface oder aus einem Kommunikationspaket) einzuspeisen, um das zugehörige Behandlungsprogramm auf schnellstem Wege zu erreichen.



a) Funktionsverzweigung durch Signaleinspeisung in bestimmte Bitpositionen des Befehls, z. B. in die Verzweigungsadresse
b) Ausführung eines zuvor geladenen Befehls (EXECUTE-Befehl)

– Der Prozessorkern als Mikroprogrammsteuerwerk –

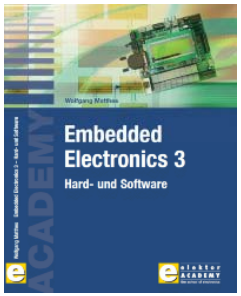
– Eine Design-Idee –

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 Prinziplösungen der Mikroprogrammierung neu beleben.
- Es dürfte alles frei sein (Patente längst abgelaufen)*.
- Das Grundsatzproblem: Womit die Programme schreiben?
- Heutzutage brauchen die Anwendungslösungen viel Software. Aber nur wenige Programme müssen wirklich beschleunigt werden.
- Deshalb wird eine weitere Idee aus der Entwicklungsgeschichte wiederbelebt: beim herkömmlichen Prozessor bleiben und die meisten Programme lassen, wie sie sind, aber außerhalb des Prozessors Erweiterungsschaltungen vorsehen und bedarfsweise in die Speicherzugriffswege eingreifen.
- Auf diese Weise können übliche Maschinenbefehle praktisch zu Mikrobefehlen erweitert werden.
- Hierzu wurden mehrere Schaltungslösungen vorgestellt.

*: Vorsicht. Aussage ohne Gewähr.

In dieser Schrift wurde das Thema bereits angesprochen:



(Eine Monographie zur Mikroprogrammierung ist in Arbeit.)