Halbleitertechnologie von A bis Z

Digitaltechnik

Steuerwerk und Befehlsausführung

Datenpfad und Steuerwerk: wer macht was

Ein Prozessor lässt sich grob in zwei Teile gliedern: den Datenpfad und das Steuerwerk. Der Datenpfad besteht aus genau den Bausteinen, die in den vorherigen Kapiteln behandelt wurden – ALU (Addierer, Multiplizierer), ein Satz von Registern (Flip-Flops, meist zu einer sogenannten Registerdatei mit 16 bis 32 Einträgen zusammengefasst) und Busse, über die Daten zwischen ihnen verschoben werden, mit Multiplexern an jeder Stelle, an der ausgewählt werden muss, welche Datenquelle gerade durchgeschaltet wird. Der Datenpfad selbst trifft dabei keinerlei Entscheidungen; er tut in jedem Taktzyklus exakt das, wozu ihn die an seinen Steuereingängen anliegenden Signale zwingen.

Konkret sind das typischerweise: die Adressleitungen der Registerdatei, die festlegen, welche zwei Register auf die ALU-Eingänge gelegt werden und in welches Register das Ergebnis zurückgeschrieben wird; ein Write-Enable-Signal, das dieses Zurückschreiben überhaupt erst freigibt; ein ALU-Opcode, der bestimmt, ob die ALU addiert, subtrahiert, eine logische Verknüpfung bildet oder schiebt; sowie diverse Mux-Select-Leitungen, die etwa festlegen, ob der zweite ALU-Eingang aus einem Register oder direkt aus einem in der Instruktion enthaltenen Konstantwert (Immediate) stammt. All diese Signale erzeugt das Steuerwerk, und zwar rein aus zwei Eingangsgrößen: dem aktuellen Taktzyklus innerhalb der Befehlsausführung und dem sogenannten Opcode, den ersten paar Bits der gerade auszuführenden Instruktion. Das Steuerwerk "versteht" dabei nichts im menschlichen Sinn – es ist selbst nur eine (unter Umständen sehr komplexe) kombinatorische und sequenzielle Schaltung, die für jede Kombination aus Opcode und Zyklus ein festes, vorher beim Entwurf festgelegtes Muster an Steuersignalen ausgibt.

Opcode und der Fetch-Decode-Execute-Zyklus

Eine Instruktion im Speicher ist nichts als eine feste Bitfolge, typischerweise 16 bis 32 Bit breit, die sich in Opcode (welche Operation) und Operanden (mit welchen Registern oder Werten) aufteilt. Bei einer stark vereinfachten 32-Bit-Additionsinstruktion könnten etwa Bit 26 bis 31 den Opcode für "Addiere" kodieren, Bit 21 bis 25 das erste Quellregister, Bit 16 bis 20 das zweite Quellregister und Bit 11 bis 15 das Zielregister – der Rest bliebe ungenutzt oder würde bei anderen Opcodes für einen Immediate-Wert verwendet. Ein spezielles Register, der Program Counter (PC), enthält jederzeit die Speicheradresse der als Nächstes auszuführenden Instruktion.

Die Ausführung folgt einem sich ständig wiederholenden Muster: In der Fetch-Phase wird die Instruktion an der vom PC angegebenen Adresse aus dem Speicher gelesen und in ein spezielles Instruktionsregister geladen; in der Decode-Phase zerlegt das Steuerwerk den Opcode und leitet daraus die für diese Instruktion nötigen Steuersignale ab, während parallel bereits die im Opcode-Feld benannten Register aus der Registerdatei gelesen werden; in der Execute-Phase durchläuft der Datenpfad, gesteuert von genau diesen Signalen, die eigentliche Operation, etwa eine ALU-Berechnung, gefolgt von einer optionalen Speicherzugriffsphase (bei Lade-/Speicherbefehlen) und einer abschließenden Rückschreibphase ins Zielregister. Am Ende wird der PC um die Breite der aktuellen Instruktion erhöht (bei den meisten Instruktionen also einfach um 4 Byte) – oder bei einem Sprungbefehl direkt auf eine neue, aus der Instruktion berechnete Adresse gesetzt, etwa wenn eine Bedingung wie "Register gleich null" erfüllt ist. Der Zyklus beginnt dann von vorn mit der nächsten Instruktion. Ein Programm ist damit letztlich nichts weiter als eine lange, vom Compiler erzeugte Folge solcher Bitmuster im Speicher, die diesen Zyklus Instruktion für Instruktion mit wechselndem Opcode durchläuft – Verzweigungsbefehle sorgen dabei dafür, dass diese Folge nicht zwingend linear im Speicher abläuft, sondern je nach Zwischenergebnis unterschiedliche Pfade nimmt, was letztlich Schleifen und bedingte Anweisungen auf Hardwareebene ermöglicht.

Hardwired- versus Microcode-Steuerwerk

Für die Realisierung des Steuerwerks selbst haben sich historisch zwei grundsätzlich unterschiedliche Ansätze etabliert. Beim Hardwired-Steuerwerk wird die Zuordnung von Opcode und Zyklus zu Steuersignalen direkt als kombinatorische Logik und ein kleiner Zustandsautomat in Gattern realisiert – vergleichbar mit den in den vorherigen Kapiteln behandelten Schaltungen, nur auf einer höheren Abstraktionsebene, bei der die "Eingänge" nicht einzelne Bits, sondern ganze Opcode-Felder sind. Dieser Ansatz ist sehr schnell, da die Steuersignale ohne zusätzlichen Speicherzugriff direkt aus der Verdrahtung entstehen, wird aber bei einem großen, unregelmäßigen Befehlssatz mit vielen Sonderfällen schnell unübersichtlich, schwer zu verifizieren und bei nachträglich entdeckten Fehlern praktisch nicht mehr korrigierbar, da jede Änderung eine neue Maskensatzrevision erfordern würde.

Beim Microcode-Steuerwerk dagegen ist für jeden Opcode eine eigene, in einem kleinen internen ROM oder PLA hinterlegte Folge von "Mikrobefehlen" gespeichert, die Schritt für Schritt genau die Steuersignale liefern, die das Hardwired-Steuerwerk sonst direkt verdrahtet erzeugen würde; eine komplexe Instruktion wird so intern in eine Sequenz einfacherer Mikrooperationen

zerlegt, ähnlich einem kleinen, fest einprogrammierten Unterprogramm auf einer noch niedrigeren Ebene als die eigentliche Maschinensprache. Dieser Ansatz lässt sich leichter entwerfen und verifizieren, da neue oder korrigierte Instruktionen im Prinzip nur eine neue Mikrocode-Tabelle statt neuer Gatterverdrahtung erfordern, und moderne x86-Prozessoren nutzen diese Eigenschaft bis heute für sogenannte Microcode-Updates, mit denen sich selbst nach der Auslieferung noch Fehler oder Sicherheitslücken in der Befehlsausführung per Firmware beheben lassen. Der Mikrocode-Zugriff kostet dabei allerdings zusätzliche Taktzyklen gegenüber einer rein festverdrahteten Lösung und war einer der Haupttreiber hinter der RISC-Philosophie der 1980er-Jahre, die bewusst auf einen stark vereinfachten, gleichförmigen Befehlssatz mit einheitlicher Instruktionslänge setzte, um wieder rein hardwired und damit schneller steuern zu können – ein Kompromiss, den heutige Prozessoren oft hybrid lösen, indem einfache, häufige Instruktionen hardwired dekodiert werden und nur seltene, komplexe Instruktionen den langsameren Mikrocode-Pfad durchlaufen.