Ablaufsteuerung

Der manuelle Betrieb der ALU/SHIFT-Einheit und des Adressgenerators diente zum Test und zur Vermittlung des Grundverständnisses beider Funktionsblöcke.
Um einen einzigen Teilschritt auszuführen, waren jeweils einige (mindestens einer bis zu 16) der 80 Schalter zu aktivieren. Mehrere Teilschritte bilden wiederum einen Befehl.

Ziel dieses Abschnitts ist es, einen automatischen Schritt-für-Schritt-Ablauf der Schalterstellungen zu realisieren, nicht lineare Befehle wie bedingte Sprünge und die externen Hardware-Signale RST, NMI und INT zu erkennen und zu verarbeiten.

Während für das Design der ALU/SHIFT-Unit und des Adressgenerators dem Entwickler durch den vorgegebenen bzw. gewünschten Befehlssatz (Funktionen und Adressierungsarten) relativ enge Grenzen gesetzt sind, gibt es bei der Realisierung des Steuerwerks große Gestaltungsfreiheiten.

Unterscheidungen von Steuerwerken

In der Praxis werden Mikroprozessoren in unterschiedliche Kategorien eingeteilt.

Zum einem unterscheidet man in Prozessoren mit der „von-Neumann“-Struktur (benannt nach dem gleichnamigen Mathematiker), bei dem Befehlsdaten und Nutzdaten (Operanden) im gleichen Speicher abgelegt sind. Der Befehl ADC #05 setzt sich beim 65C02 aus dem Befehlsbyte 69 für „Addiere das auf das Befehlsbyte folgende Datenbyte mit dem C-Bit in den Akkumulator“ und dem Datenwert 05h zusammen.

Bei Prozessoren mit Harvard-Struktur liegen Befehle und Daten in getrennten Speicher. Mit Transportbefehlen werden die benötigten Daten in einen separaten Arbeitsbereich gebracht und dort mit einem Funktionsbefehl arithmetisch oder logisch verknüpft.

Auf der anderen Seite werden Prozessoren (genauer deren Steuerwerke) in CISC (complex instruction set computer) oder RISC (reduced instruction set computer) unterschieden. Bei CISCs wird das Schalten der Steuerleitungen durch die Ausgabe entsprechender Daten, die in Festwertspeichern abgelegt sind, realisiert. Bei RISCs werden die Zustände der Steuerleitungen direkt aus den Befehlen per Hardware dekodiert. In der heutigen Praxis gibt es eher Mischformen mit unterschiedlicher Gewichtung. Allgemein gilt, dass CISCs einen größeren Befehlsumfang besitzen und durch die Verwendung von Festwertspeichern (Adressierungs- und Ausgabezeitverzögerung) langsamer als RISCs sind. Umgekehrt sind RISCs durch die Hardwaredekodierung schneller als CISCs, besitzen aber in der Regel weniger Befehle als CISCs, da mit jedem zusätzlichen Befehl der Hardwareaufwande für die Dekodierung über-proportional steigt und der Geschwindigkeitsvorteil abnimmt.

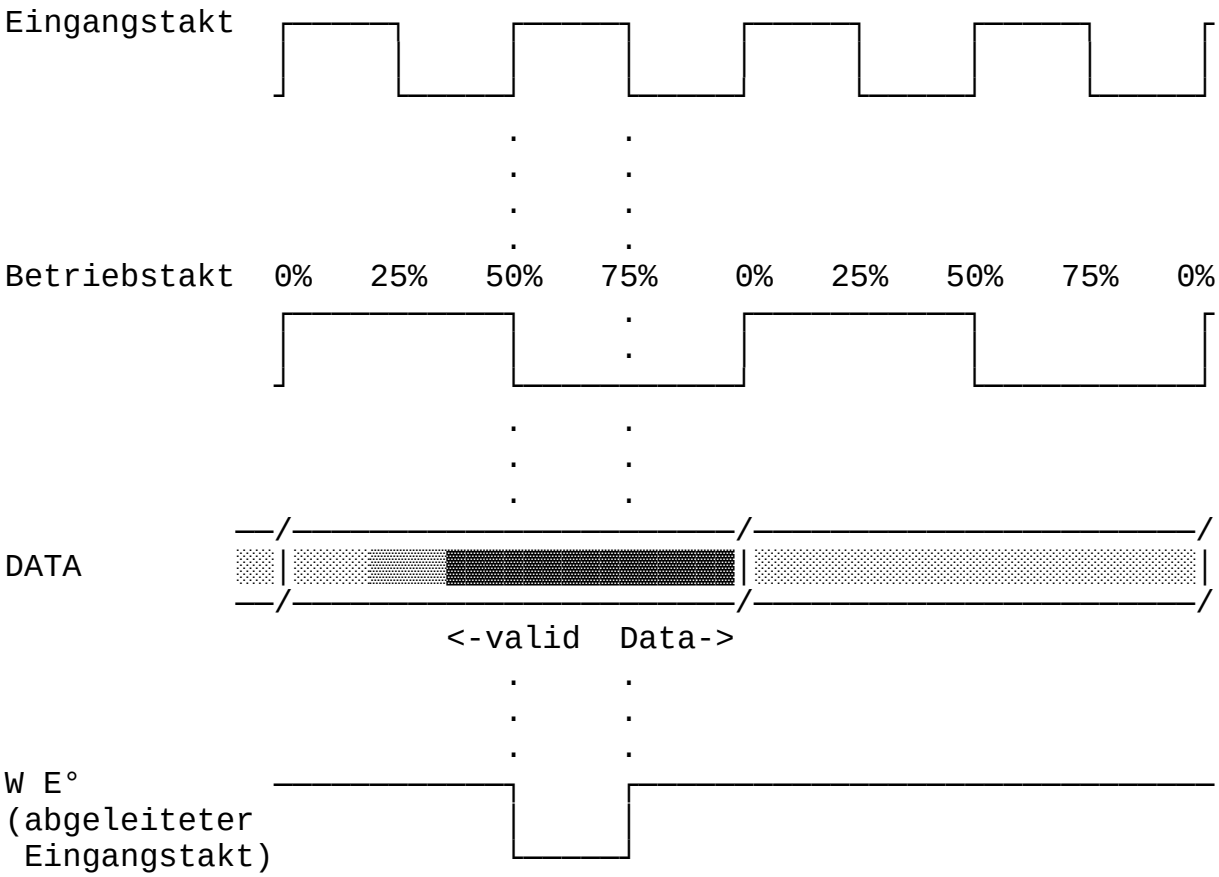
Takt

Im Gegensatz zu Analogrechnern wird bei digitalen Rechnern der Programmablauf schrittweise abgearbeitet. Ein einzelner Befehl besteht aus mehreren Teilschritten, die nacheinander ausgeführt werden. Durch das Anlegen eines Taktsignals am Clock-Eingang des Prozessors wird die Arbeitsgeschwindigkeit und die Arbeitsweise im Inneren festgelegt.

Manche Prozessoren der Anfangszeit (z.B. Motorola 6800) benötigten dabei einen bereits extern aufbereiteten Systemtakt (z.B. zwei phasenverschobene, nicht überlappende Taktsignale), andere erzeugen intern aus einem einfachen externen Systemtaktsignal die benötigten internen Taktsignale.

Prozessoren späterer Generationen werden mit einem einfachen hochfrequenten, symetrischen Rechtecksignal, das ohne großen externen Aufwand realisiert werden kann, betrieben. Bei diesem Eingangssignal handelt es sich aber keineswegs um den Betriebstakt, mit dem die Schaltung angesteuert wird. Vielmehr wird der Eingangstakt um den Faktor 2 (gelegentlich auch mehr) geteilt und der resultierende Takt als Betriebstakt verwendet. Durch diesen Trick ergeben sich zusätzliche Flanken, d.h. Schaltmöglichkeiten, um die einzelnen Funktionen der Schaltung zu bestimmten Zeitpunkten präzise zu triggern.

Das folgende Beispiel zeigt, wie unter Verwendung des Eingangstaktes und des Betriebstaktes das Schreiben in ein statisches RAM realisiert wird. Der Betriebstakt wird durch eine Halbierung des Eingangstaktes erzeugt und bestimmt den Grundtakt der Schaltung (und damit auch die Verarbeitungsgeschwindigkeit). Mit der steigenden Flanke des Betriebstaktes werden die Zieladresse und die Daten auf ihren jeweiligen Bus ausgegeben. Während der ersten Hälfte des Taktes stabilisieren sich die Daten auf den externen Bussen. Nach der Hälfte des Taktes kommt der Eingangstakt zum Einsatz, aus dem das WE°-Signal abgeleitet wird. Das WE°-Signal liegt bis zum Zeitpunkt 75% des Betriebstaktes an, zu dem der Schreibvorgang abgeschlossen wird. Adresse und Daten bleiben aber bis zum Ende des Betriebstaktes stabil auf den Bussen erhalten, so wie es das Timingverhalten eines RAMs erfordert.



Durch weitere Teilung des Eingangstaktes lassen sich zusätzliche Flanken und Zeiträume gewinnen, um die Ablaufsteuerung noch feiner aufzulösen. Allerdings erhöht sich damit die Komplexität für die Ablaufsteuerung erheblich.

Bei einigen Prozessoren sind der Taktgenerator und das Steuerwerk derart gestaltet, das ausschließlich ein dynamischer Betrieb möglich ist, d.h. es ist eine Mindestfrequenz erforderlich, um einen sicheren Betrieb zu gewährleisten.
Andere System können statisch betrieben werden, d.h. das durch das Ausschalten des Taktes der interne Betrieb eingestellt und durch erneutes Takten der Verarbeitungsprozess ordnungsgemäß und unverzüglich wieder aufgenommen wird. Diese Funktion wird in der Regel für den Bereitschaftsmodus oder die Energieeinsparung des Systems (Wait- oder Sleep-Modus) eingesetzt. Der Modus wird per Befehl aktiviert und durch einen Interrupt (extern oder interner Zähler) aufgehoben.

Taktung der Steuersignale

Es gibt zwei Grundprinzipen, mit denen die Ausgabe der Steuersignale aus den Festwertspeichern realisiert werden kann.

Die Steuersignale werden alle (auch die, die erst zu einem späteren Zeitpunkt innerhalb eines Taktes wirksam werden sollen) mit der steigenden Flanke des Betriebstaktes ausgegeben. Die Steuersignale, die zum Eintakten der Daten oder zur Definition von Zeitfenstern während des Betriebstaktes verzögert geschaltet werden sollen, müssen dann jeweils über eine Verknüpfung mit dem abgeleiteten Eingangstakt aktiviert werden (s. Ein-Phasen-Takt). Da dabei jedes verzögerte Signal mindestens ein zusätzliches Gatter benötigt, ist diese Methode nur bei einer geringen Anzahl von verzögerten Signalen sinnvoll.

Ist die Anzahl der verzögerten Signale ausreichend groß (z.B. acht oder ein Mehrfaches bei Verwendung von 8-bit-ROMs) so ist es sinnvoll, diese Signale in ein separates ROM abzulegen, dessen CS-Eingang mit dem erforderlichen Pegel des entsprechenden Takts aktiviert wird.

Ein-Phasen-Takt

Die Auswahl der Taktform hat erheblichen Einfluss auf den Aufbau des Steuerwerkes und dessen Effizienz. Bei dem hier vorgestellten System kommen ein Ein-Phasen-Takt und ein Steuerwerk für einen statischen Betrieb zum Einsatz. Damit ist ein Schritt-für-Schritt-Betrieb möglich, bei dem der automatische Programmablauf wahlweise durch einen Taktgenerator oder durch Betätigung eines Schalters/Tasters schrittweise gesteuert werden kann.

Der Nachteil eines Ein-Phasen-Taktes ist das Vorhandensein von nur zwei Flanken, d.h. nur zwei Zeitpunkten, während denen jeweils Aktionen ausgelöst werden können.

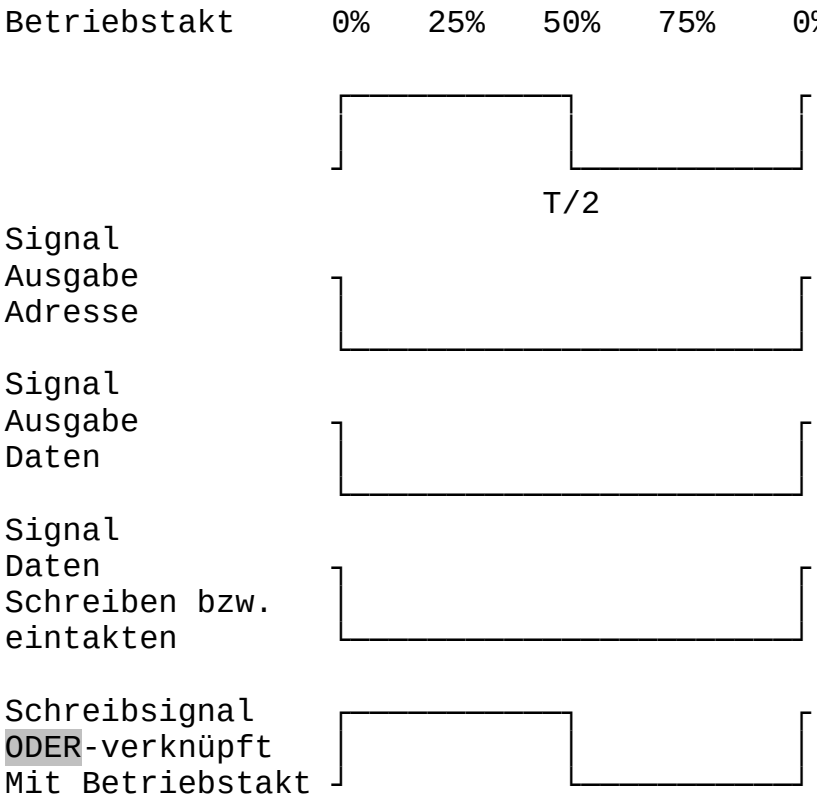
Für dieses Projekt wird folgendes vereinbart:

Der Eingangstakt ist der Betriebstakt.

Mit der steigenden Flanke des Systemtakts wird eine Datenausgabe ausgelöst und ein Zielregisters oder einer Adresse selektiert. Bis zum Zeitpunkt T/2 haben sich die Signale auf dem Daten- und Adressbus stabilisiert und werden mit der fallenden Flanke in das Ziel eingetaktet. Eine Ausnahme von diesem Verfahren wird nur beim X-, Y- und Adressregister sowie Programmzähler gemacht. Bei diesen Registern wird zur Vermeidung von Ringbefehlen jeweils ein weiteres Register nachgeschaltet, das zur zeitlichen Entkoppelung des Lese- und Schreibvorgangs erst mit der steigenden Flanke des nächsten Takts aktualisiert wird. Die Zeit von T/2 bis zum Ende des Systemtakts wird nicht genutzt, ein Luxus, den man sich nur in nicht-professionellen Systemen leisten kann, der aber eine deutliche Vereinfachung für die Realisierung eines Steuerwerks bedeutet. Durch den Umstieg von einem symetrischen zu einem asymetrischen Takt (verkürzte Low-Phase) ließe sich die Verarbeitungsgeschwindigkeit noch steigern.

Im Abschnitt „RAM“ wird gezeigt, dass ein realer Betrieb mit einem statischen RAM bei der Verwendung eines Ein-Phasen-Taktes nur mit zusätzlichen Maßnahmen möglich ist.

Das Signal zum Schreiben bzw. Eintakten der Daten in eine Adresse oder Register wird ebenfalls zu Taktbeginn ausgegeben und mit dem Taktsignal über ein ODER-Gatter verknüpft, so dass das Signal bis zum Zeitpunkt T/2 „high“ bleibt und erst bei T/2 verzögert aktiv („low“) wird.



Eine einzelne Aktion besteht aus einer steigenden Flanke zur Ausgabe von Daten oder Adressen und einer fallenden Flanke zum Schreiben bzw. Eintakten von Daten.

Der Original 65C02 arbeitet intern mit einem Zwei-Phasen-Takt, d.h. mit zwei Flankenpaaren. Im Vergleich zu dem hier vorgestellten Nachbau, der pro Takt eine Aktion durchführt (ein Flankenpaar), ist er somit theoretisch in der Lage, zwei Aktionen innerhalb eines Taktes durchzuführen. Durch das fest vorgegebene Zwei-Phasen-Raster beim Original sind für die Abarbeitung der Befehle jedoch immer 2,4,6,8 usw. Flankenpaare anzusetzen, auch wenn der Befehl u.U. nur eine ungerade Flankenpaaranzahl benötigt (ungenutztes Flankenpaar). Beim Ein-Phasen-Takt werden nur die erforderlichen Flankenpaare (Takte) „verbraucht“.
Aus diesem Grund sind die Zyklusangaben beider Systeme nicht kompatibel. Die Anzahl der effektiven Flanken, d.h. der logischen Einzelschritte ist beim Original-65C02 knapp zweimal höher als die Zyklusangabe für den jeweiligen Befehl.

Symetrische Teilschritte

Normalerweise wird ein Prozessor mit einem symetrischen Betriebstakt angesteuert, d.h. High- und Lowpegel des Taktes sind jeweils 50% aktiv und alle aufeinander folgenden Zyklen sind gleich lang. Die maximale Arbeitsgeschwindigkeit hängt von dem Einzelschritt ab, der die längste Zeit für seine Abwicklung benötigt. Alle Teilschritte, die weniger Zeit benötigen und damit „früher“ abgearbeitet sind, enthalten somit eine gewisse Leerlaufzeit (Lastasymetrie). Die Verarbeitungszeit wird durch die Signallaufzeiten auf den Leitungen, den Anforderungen bezüglich Signalstabilisierung an den Gattereingängen und vor allem durch die Gatterlaufzeiten selbst bestimmt. Bei professionellen Systemen kommt es daher darauf an, die Einzelschritte derart auszulegen, dass ihre Verarbeitungszeiten möglichst gleich lang (kurz) sind, sodass nicht ein einzelner, langer Teilschritt alle anderen ausbremst und den Gesamtdurchsatz des Systems reduziert. Bei diesem Projekt wurde keine zeitliche Teilschrittanalyse durchgeführt, da die Schaltung der visuellen Demonstration der internen Abläufe im unteren Hertz-Bereich und nicht als professioneller Hochleistungsprozessor dienen soll. Bei einer Analyse ist jeweils die Summe der Laufzeiten aller Gatter, die an einem Teilschritt beteiligt sind, zu berücksichtigen.
Untersucht man die Laufzeiten einzelner Gatter so stellt man fest, dass vor allem externe Schreib-/Lesevorgänge (RAM/ROM) und intern die Mikro-Code-Adressierung (ROM) sowie die ALU-Verarbeitung die längsten Verarbeitungszeiten benötigen. Bei diesem Projekt wurde an einer Stelle bewusst zu Gunsten der Vereinfachung auf eine optimierte Lastsymetrie verzichtet. Die Strecke *Ausgabe des ALU- bzw. ALU-IN–Inhalts, ALU-Verarbeitung und das Rückschreiben des ALU-Ergebnisses* wird in einem einzigen Takt durchgeführt. Eine solch lange Verarbeitungskette würde man normalerweise auf jeden Fall unterteilen, in dem das ALU-Ergebnis zunächst zwischengespeichert wird und erst im Folgetakt in den ACCU zurück geschrieben wird. Da das Zwischenspeichern des ALU-Ergebnisses aber ansonsten keinen Erkenntnisgewinn bedeutet, wurde bei dieser Beispielschaltung darauf verzichtet.

Theoretisch lassen sich auch System vorstellen, bei denen dynamisch asymetrische Taktzeiten je nach Befehl zum Einsatz kommen. Durch entsprechend ausreichend hohen und mehrfach geteilten Betriebstakt, den Einsatz eines Zählers und befehls-spezifisch hinterlegte Taktdaten (Anzahl und Dauer) für die Zählerinitialisierung können Befehle mit unterschiedlichen zeitlichen Anforderungen ausgeführt werden.

Befehlsstruktur

Die Hex-Codes der Befehle von 8-Bit-von-Neumann-Prozessoren werden in der Regel in einer Befehlsmatrix angeordnet. Auf der horizontalen bzw. vertikalen Seite findet man die 16 High-Nibbles der Befehle (Most-significant-Bits, MSB), auf der vertikalen bzw. horizontalen die 16 Low-Nibbles (Least-significant-Bits, LSB). Bei vielen Aufstellungen erkennt man auf Anhieb die systematische Anordnung der Befehlsarten (s. 6805). Befehle mit gleicher Adressierungsart sind z.B. in der gleichen Nibble-Spalte und Befehle mit gleicher numerischer oder logischer Funktion in der gleichen Nibble-Reihe. Alle anderen Befehle, die nicht diesem Schema entsprechen, sind ebenfalls in ein oder zwei Reihen bzw. Spalten untergebracht. Durch diese Anordnung ist es bei CISC-Prozessoren möglich, die meisten Befehle mit zwei Nibble-Dekodern nach Adressierungsart und Funktion zu analysieren. Die jeweiligen Teilschritte müssen dann nur einmal im Mikro-Code-ROM abgelegt sein und werden bei jedem entsprechendem Befehl wie ein Unterprogramm abgerufen. Dadurch ist der Speicherbedarf minimal und die Wartung der Teilschritte einfach. Lediglich die übrigen Befehle müssen separat dekodiert werden. Die Befehlsstruktur beim 6502 ist etwas komplexer angelegt, da gleichartige Befehle nicht im Abstand von 16 sondern von 32 Schritten angelegt sind und die im 65C02 nachträglich ergänzten Befehle zwar systematisch strukturiert, aber aus Platzmangel in andere Spalten bzw. Zeilen implementiert wurden (das lässt vermuten, dass der Decoding-Aufwand bei diesem Prozessor nicht unerheblich ist). Auch das Einlesen des nächsten Befehls (Fetch-Cycle) ist als „Unterprogramm“ abgelegt und wird am Ende bzw. am Anfang jedes Befehls aufgerufen.

Bei dem hier vorgestellten System wird ein anderer Weg eingeschlagen. Es wird nicht nach Adressierungsart, Funktion und sonstigen Befehlen unterschieden. Der 8-Bit-breite Binärwert jedes Befehls dient als Teil der Einsprungadresse in das Mikro-Code-ROM. Das hat zur Folge, dass die Adressierungsart und die Funktion explizit für jeden Befehl separat im ROM eingetragen werden muss. Die Anforderungen an den ROM-Speicherplatz steigen damit erheblich an. Für ein Demonstrationssystem ist es aber unerheblich, ob man zehn 2708 oder 2732 einsetzt. Auch die Sequenz für das Einlesen des nächsten Befehls wird jedem Befehle separat angehängt. Der Wartungsaufwand steigt entsprechend, da bei einer Änderung einer Adressierungsart oder Funktion der Code bei allen betroffenen Befehlen modifiziert werden muss. Werden die Sequenzen jedoch in einer Datenbank abgelegt, so lassen sich

Modifikationen relativ leicht bewerkstelligen. All diesen Nachteilen steht jedoch ein großer Vorteil gegenüber: Der Befehlsdecoder ist nahezu matrix- und hardwareunabhängig. Nicht nur selbst erstellte Erweiterungen des Befehlssatzes können an jeder beliebigen freien Stelle implementiert werden sondern alle Befehle können an jeder beliebigen Stelle angeordnet werden, ohne dass aufwendige Modifikationen für das Dekodieren der Befehle erforderlich werden. Da nur die Befehle für bedingte Sprünge und einige 65C02-spezifischen Sonderbefehle über zwei Nibble-Decoder gefiltert werden, kann der Befehlsdekoder zudem nahezu unverändert für andere Prozessortypen verwendet werden.

Die Mikrocode-Speicher-Adressierung

Ein Befehl besteht aus mehreren Teilschritten. Mit jedem Teilschritt werden die zu diesem Zeitpunkt benötigten Signale aktiviert (low). Alle für den ordnungsgemäßen Ablauf notwendigen Teilschritte, d.h. die Zustände der beim manuellen Betrieb zu jedem Zeitpunkt eingestellten Schalter, werden nun in ROMs abgespeichert. Dieser Speicher wird im Folgenden als µCode-Speicher oder Sequenzspeicher bezeichnet und ist nicht mit dem Arbeitsspeicher bzw. Programm-/Datenspeicher des Mikroprozessors zu verwechseln!!! Um die 80 Signale (76 werden bei der vorliegenden Version verwendet) für jeden Teilschritt auszugeben, werden zehn 8-Bit-ROMs parallel geschaltet.

Die erforderlichen Teilschritte für einen Befehl werden als Sequenz bezeichnet!!

Der Befehlsumfang reicht von 00h bis FFh, es sind somit theoretisch 256 Befehle möglich. Bei diesem System werden für die längsten Befehle maximal 10 Teilschritte benötigt (z.B. JSR (ZP,X). Aus diesem Grund werden für jeden Befehl 4-Bit (entsprechend 16 möglichen Teilschritten) reserviert. Die Mikro-Code-Adressen für einen Befehl setzen sich aus dem binären Wert des Befehlsbytes (8 Bit) und den 4 Bit für die einzelnen Teilschritte zusammen.
Als Beispiel dient die Adressierung des Befehls LDA imm. (A9h). Der Wert A9 bleibt für alle Teilschritte konstant, die letzten vier Bit werden, soweit wie für den Befehl erforderlich, mit jedem Takt inkrementiert.

HEX	BINÄR-ADRESSE	KOMMENTAR
A9 0	1010 1001 0000	erste Teilschrittadresse: adressiere Speicher und lies den Wert in ACCU, setze CCR-Bits (Ausführungstakt A1)
A9 1	1010 1001 0001	zweite Teilschrittadresse: hole nächsten Befehl (Fetchcycle F1)
A9 2	1010 1001 0010	dritte Teilschrittadresse: dekodiere neuen Befehl (Fetchcycle F2)

Die Sequenz beinhaltet bei diesem Beispiel die 3 Teilschritte A1, F1 und F2. Die Teilschritte vier bis sechzehn werden in diesem Fall nicht benötigt, die Steuersignale sind alle „high“.

Bei dieser Methode sind somit 12 Adressleitungen erforderlich, um alle theoretisch möglichen 256 Befehle mit jeweils max. 16 Teilschritten abzulegen.

Zusätzlich müssen noch die Sequenzen für RESET, BRK, NMI und IRQ im ROM untergebracht werden. Diese könnten auf freie Befehlsbyte gelegt werden, was aber die Gefahr birgt, dass eine versehentliche Benutzung dieses Befehlsbytes eine unerwünschte Hardware-Reaktion auslöst. Ein nicht verwendetes Befehlsbyte sollte jedoch sicherheitshalber nur den nächsten Befehl laden ohne sonstige Operationen auszulösen.

Aus diesem Grund werden die o.a. Systemsequenzen und, wie später gezeigt wird, einige weitere Teilschritte in einem weiteren ROM-(Bereich) abgelegt. Da es sich nur um wenige Sequenzen handelt und nur ein Teil der 80 Steuerleitungen benötigt werden, könnte man diese Daten in einige wenige, kleine ROMs abspeichern. Dies bedeutet allerdings, dass je nach Befehlsart mit entsprechendem Schaltungsaufwand (s. fehlendes CS° bei ROMs) zwischen den großen und kleinen ROMs gewechselt werden müsste.

Um, wie eingangs gefordert, die Anzahl der Bauteile möglichst gering zu halten, wird hier die Kapazität der ROMs verdoppelt und mit der 13. Adressleitung (A12) zwischen den Standardbefehlen des Befehlssatzes (A12 = low/Page 0) und den Sonderbefehlen (A12 = high/ Page 1) umgeschaltet.

Damit ist eine einheitliche Struktur für alle Befehle mit folgendem Aufbau realisiert:

256 mögliche Befehle in Page 0, davon 244 genutzt, mit jeweils 16 Teilschritten und 80 Bit Datenbreite (Standardbefehle)

256 mögliche Befehle in Page 1, davon 24 genutzt, mit jeweils 16 Teilschritten und 80 Bit Datenbreite (Sonderbefehle)

An dieser Stelle wird noch einmal darauf hingewiesen, dass dieses Verfahren für die Befehlsdekodierung zwar effizient im Hinblick auf die Anzahl der Bauteile aber höchst ineffizient bezüglich der Speichernutzung ist. Obwohl die Adressierungsarten und Funktionen für jeden Befehl explizit im ROM eingetragen werden (was an sich schon eine Verschwendung von Ressourcen bedeutet), finden von 512 * 16 = 8192 Micro-Step-Zeilen nur 1359 (<17%) Verwendung.

Mit jeder steigenden Flanke von CLK wird der Einzelschritt-zähler (Micro-Step-Counter, µStep-Counter) um eins erhöht und damit die nächste Adresse, d.h. der nächste Einzelschritt in den Code-ROMs aktiviert. Durch die systembedingten Gatterlaufzeiten der ROMs werden die Steuersignal erst mit Verzögerung ausgegeben und stehen auch noch einige Nanosekunden zu Beginn des nächsten Taktes an! Diese Verzögerung bildet die Grundlage für das Zeitverhalten der hier vorgestellten Ablaufsteuerung!!!!!

Alle Befehle mit Ausnahme der bedingten Sprungbefehle werden als lineare Befehle bezeichnet. Ihre Teilschritte liegen direkt hintereinander und werden auch in dieser Abfolge aufgerufen.

Bei nicht-linearen Befehlen wird, wie später gezeigt wird, abhängig von bestimmten Zuständen, ggfs. die Abarbeitung der Teilschritte vor dem eigentlichen Ende des Befehls abgebrochen und an einer anderen Stelle mit einer anderen Befehlssequenz fortgeführt.

Fetch-Cycle

Um eine Befehl zu dekodieren, muss dieser zunächst aus dem Arbeitsspeicher in den Prozessor geladen werden. Es ist möglich, den Befehl vor seiner Ausführung oder am Ende jedes vorhergehenden Befehls zu laden. Bei diesem Beispiel kommt die zweite Methode zum Einsatz.

Ein Befehl besteht aus zwei Phasen, der Ausführungsphase (Execution-Cycles) und der Hol-Phase (Fetch-Cycles). Während der Ausführungsphase werden die Teilschritte für die eigentliche Funktion eines Befehls abgearbeitet. Mit dem Abschluss der Ausführungsphase muss der Programmzähler auf die Adresse des nächsten Befehls eingestellt worden sein.

Im Anschluss an die Ausführungsphase muss ein neuer Befehl in die Steuereinheit geladen werden. Diese Phase wird als Hol-Phase (Fetch-Cycle) bezeichnet, die sich wiederum in die eigentliche Hol-Phase d.h. das Lesen des nächsten Befehls (Get) und die Dekodier-Phase (Decode) untergliedert, in der der Befehl und die anstehenden bzw. eingegangenen externen Funktionen NMI und INT ausgewertet werden. Im Folgenden werden die Get-Phase mit F1 und die Decode-Phase mit F2 bezeichnet.

Eine Sequenz setzt sich somit aus mindestens einer und maximal sieben Ausführungsteilschritten (A1 – A7) und zwei Fetch-Schritten (F1 und F2) zusammen

Um den Fetch-Ablauf übersichtlich zu gestalten, wird bei der hier vorgestellten Ablaufsteuerung zunächst der folgende Befehl geladen, dann die Interrupt-Auswertung vorgenommen und abhängig von dem Ergebnis der entsprechende Befehl aktiviert (der ursprüngliche Standardbefehl oder ein Sonderbefehl). Der Ablauf lässt sich optimieren, indem bereits im letzten Ausführungstakt der Interrupt-Status ermittelt wird und ggfs. auf das Laden des dann nicht auszuführenden Standardbefehls verzichtet wird.
Das Holen und Dekodieren eines neuen Befehls ist auch in einem einzigen Takt lösbar. In diesem Zusammenhang wird auch auf den Abschnitt “Symetrische Teilschritte“ verwiesen. Bei dem hier zu Verfügung gestellten Zwei-Schritt-Dekoder handelt es sich um die ursprüngliche Version einer ganzen Entwicklungsreihe.

Zu Beginn des GET-Cycles (F1) wird der Inhalt des Programmzählers auf den Adressbus gegeben um den neuen Befehl zu adressieren. Gleichzeitig wird mit dem FETCH° - Signal = low und PC TO AB° = low der Steuereinheit mitgeteilt, dass es sich bei diesem Byte um einen neuen Befehl handelt. Das Befehlsbyte wird in das Pre-Command-Register eingelesen, da zu diesem Zeitpunkt noch nicht klar ist, ob dieser Standardbefehl oder ein Sonderbefehl (z.B. NMI oder INT) ausgeführt wird). Das Eintakten geschieht mit der fallenden Flanke von CLK, da zu diesem Zeitpunkt alle drei Signale (FETCH°, PC TO AB° und CLK) am Eingang des dreifachen NOR-Gatters auf Low liegen und damit eine steigende Flanke am CLK-Eingang des 74273 auslösen (s. Timing-Diagramm **T1**). Durch die Realisierung dieses CLK-Signals mittels Hardware, erspart man sich eine separate Steuerleitung, die den neuen Befehl in das Pre-Command-Register eintaktet.

Nachdem das neue Befehlsbyte in das Pre-Command-Register geladen wurde, folgt in einem zweiten Takt die Dekodierphase mit folgenden Teilfunktionen:

- Auswertung der externen Signale NMI↓ und INT°
- Filtern des BRK - Befehls
- Filtern der bedingten Sprungbefehle und Auswertung der Status-Bits
- Filtern der Additions- und Subtraktionsbefehle und Auswertung des D-Bits
- Analyse der Adressierungsart
- Analyse der Funktion

Nach der Dekodierung wird am Ende des zweiten FETCH-Cycle-Taktes bzw. zu Beginn des ersten Ausführungstaktes entweder das Befehlsbyte aus dem Pre-Command-Register (Standardbefehl) oder ein Sonderbefehl (NMI, INT usw.) in das Command-Register geladen.

Für das Holen eines neuen Befehls und seine Dekodierung sind somit folgenden Voraussetzungen zu erfüllen:

Vor der Holphase muss der Programmzähler auf die Adresse des neuen Befehls im Arbeitsspeicher eingestellt sein

Hinter jeder µCode-Sequenz für die Ausführung der Funktion eines Befehls schließen sich zwei Schritte für das Holen und Dekodieren des nächsten Befehls (F1 und F2) an.

Den F1-Schritt (Holen) erkennt das System durch die Ausgabe von FETCH° = low und PC TO AB° = low.

Das Eintakten des neuen Befehls in das Pre-Command-Register erfolgt automatisch durch ein hardwaregeneriertes CLK-Signal.

Den F2-Schritt (Dekodieren) erkennt das System durch die Ausgabe von FETCH° = low und PC TO AB° = high. Da sich zwischen dem ersten und dem zweiten Schritt die µCode-Adresse ändert, ist der Pegel von FETCH° nicht zwangsläufig durchgehend auf „low“ (mit „X“ angedeutet)!

Während des zweiten Schritts wird mit der fallenden Flanke von CLK der Programmzähler um eins erhöht (CLK PC°), so dass bei der sich anschließenden Ausführungsphase direkt auf das dem Befehlsbyte folgende Argument oder bei Ein-Byte-Befehlen den nächste Befehl zugegriffen werden kann.

Aktivierung eines neuen Befehls

Die Aktivierung des ersten Teilschritts eines neuen Befehls ist mit die kniffeligste Aufgabe des Projekts und in Verbindung mit dem Ein-Phasen-Takt bzgl. des Timings der kritischste Punkt.

Sowohl das Page-Register als auch das Command-Register und der µStep-Counter müssen am Ende der Dekodier-Phase bzw. zum Anfang der Ausführungs-Phase zeitgleich initialisiert werden.
Die Ansteuerung des Command-Registers wird zunächst an Hand der Standardbefehle erläutert.

Alle Befehle des Befehlsatzes werden im Folgenden als Standard-Befehle bzw. P0-Befehle bezeichnet. Ihre jeweiligen Teil-schritte sind in der unteren Hälfte der ROMs abgelegt (A12 = low).
Steht nach der Dekodierungsphase kein Sonderbefehl zur Ausführung an (am Eingang D des P-Bit-Flip-Flops liegt low), wird das im Pre-Command-Register liegende Befehls-Byte des Standard-Befehls durch den dahinter liegenden und bei Standardbefehlen freigeschalteten Bustreiber geleitet. Das neue Befehlsbyte wird aber nicht, wie bisher üblich, mit der fallenden Flanke von CLK in das Command-Register eingetaktet, da in dem Zeitbereich der fallenden Flanke noch der Programmzähler auf das dem Befehl folgende Argument bzw. bei Ein-Byte-Befehlen auf den nächsten Befehl inkrementiert wird (CLK PC°), sodass zu diesem Zeitpunkt noch kein neuer Befehl aktiviert werden darf. In diesem Zusammenhang wird noch einmal auf die Problematik digitaler Simulationen hingewiesen, bei denen nicht unbedingt 100%ig sichergestellt sein muss, dass zeitlich nah beieinander liegende Ereignisse korrekt aufgelöst, sprich in der richtigen Reihenfolge abgearbeitet werden. Um einen ausreichend sicheren Zeitabstand zum Ereignis „CLK PC°“ zu haben, erfolgt das Eintakten des neuen Befehls in das Command-Register erst mit der steigenden Flanke von CLK zu Beginn des nächsten Taktes (Ausführungsphase). Durch die in der Realität vorhandenen ROM-bedingten Schaltverzögerungen und ein verzögertes Clock-Signal CLK’ (hier jeweils durch zwei Inverter künstlich erzeugt) stehen die entscheidenden Signale (FE°, BI° und CLK’) für die Auswertung des Eintakt-Zeitpunktes ausreichend lange genug (bis in den nächsten Takt) zu Verfügung. Man erkennt an dieser Stelle deutlich die Mängel des Ein-Phasen-Taktes, bei dem zu wenige Flanken für die verschiedenen zeitlichen Erfordernisse für das Eintakten vorhanden sind. Durch die Verzögerung von CLK zu CLK’ wird dieser Mangel über eine Hilfskonstruktion behoben, es handelt

sich aber damit auch nicht mehr um einen reinen Ein-Phasen-Takt!

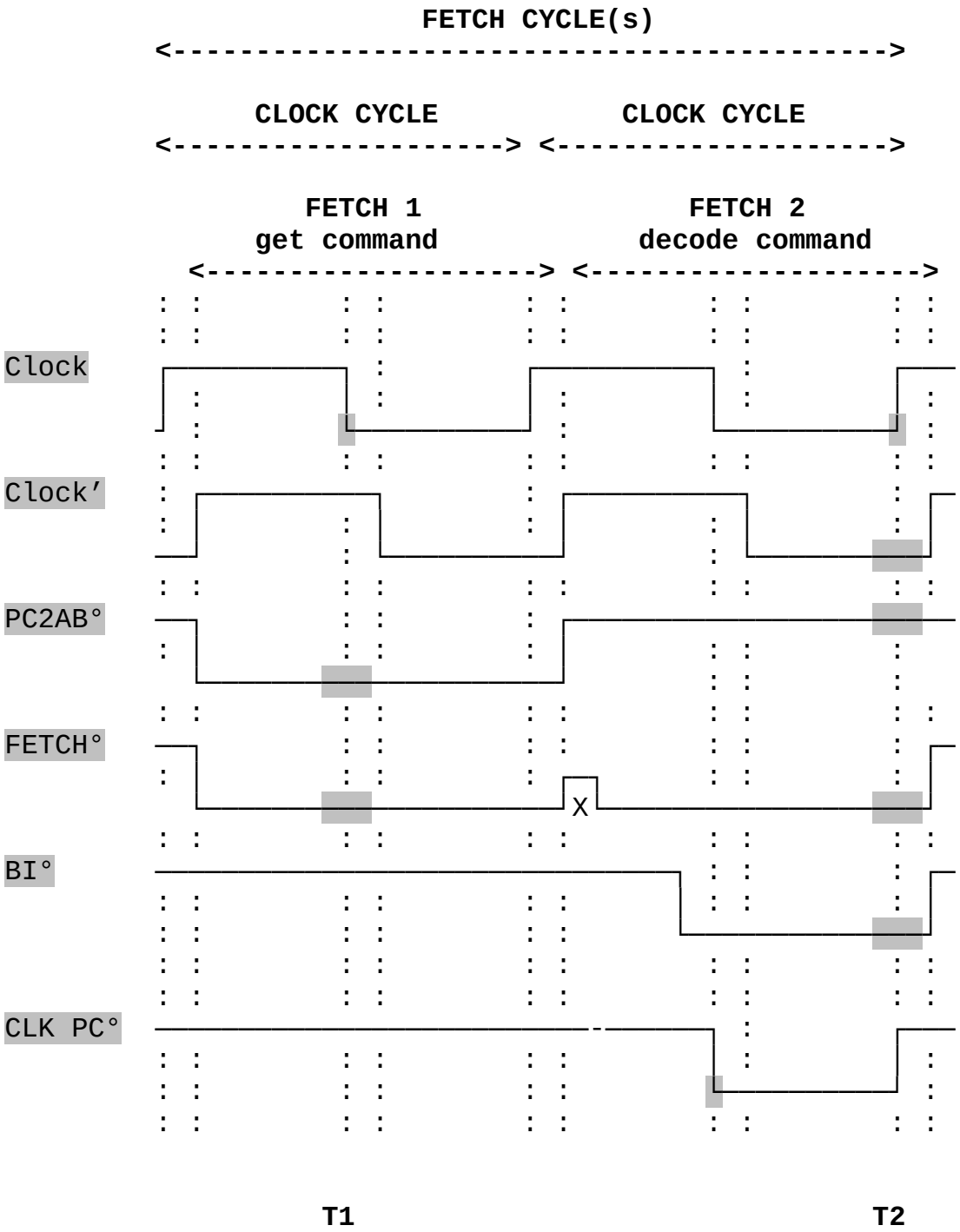


Abb. Timing-Diagram Fetch-Cycle

Unterhalb des Command-Registers liegt das Page-Flip-Flop, das den Zustand der Adressleitung A12 speichert (Standardbefehl: low, Sonderbefehl: high). Der Wert an seinem Dateneingang wird ebenfalls zum Zeitpunkt T2 eingetaktet. Die Erläuterung zur Vorbereitung der Daten für das Page-Flip-Flop erfolgt zu einem späteren Zeitpunkt.
Auch der Einzelschrittzähler (µStep-Counter) wird mit Beginn der Ausführungsphase auf „Null“ gesetzt, sodass zu diesem Zeitpunkt die 13 Adressleitungen den ersten Teilschritt des neuen Befehls adressieren.

Als Beispiel dient noch einmal die Adressierung des Standardbefehls LDA imm. (A9h). Die „0“ für den Standardbefehl und der Wert A9 bleiben für alle Teilschritte konstant, die letzten vier Bit werden, soweit wie für den Befehl erforderlich, mit jedem Takt inkrementiert.

HEX	**BINÄR-ADRESSE**	**KOMMENTAR**
0A90	0 1010 1001 0000	erste Teilschrittadresse: adressiere Speicher und lies den Wert in ACCU, setze CCR-Bits (A1)
0A91	0 1010 1001 0001	zweite Teilschrittadresse: hole nächsten Befehl (F1)
0A92	0 1010 1001 0010	dritte Teilschrittadresse: dekodiere neuen Befehl (F2)

Die Teilschritte vier bis sechzehn werden in diesem Fall nicht benötigt.

CLK DATA IN^- und CLR°- Signal

Zunächst wird analysiert, ob die Bedingung für das Aktualisieren von Page-Bit, Command-Register und µStep-Counter vorliegt. Die Bedingung ist erfüllt, wenn zum Zeitpunkt T2 PC TO AB° = „high“, das verzögerte CLK-Signal CLK’ „low“ und FETCH° oder BI° = „low“ (AND-Gatter) sind. Invertiert man PC TO AB°, so sind bei erfüllter Bedingung alle Signale low und das NOR-Gatter geht auf „high“ (T2-Signal). Mit der steigenden Flanke des (nicht verzögerten) CLK-Signals CLK wird über ein AND-Gatter ein steigende Flanke für CLK DATA IN^ und mit einem NAND-Gatter ein ausreichend langer Low-Pegel für das CLR°-Signal erzeugt.

Page-Bit-Ansteuerung

Das Page-Bit-Flip-Flop, das mit der Adressleitung A12 = low die Standardbefehle und mit A12 = high die Sonderbefehle adressiert muss mit der Aktivierung des Reset-Eingangs RST° des Mikroprozessors auf „Sonderbefehle“ eingestellt werden, da die Resetsequenz bei der Adresse 1 00 0h startet. Aus diesem Grund wird das RST°-Signal direkt auf den Preset°-Eingang des Flip-Flops gelegt, um bei aktivem RST° (Prozessor-Start) das Bit zu setzen.

Im laufenden Betrieb wird der während der Decoding-Phase ermittelte Wert zu Beginn der Ausführungsphase in dass Flip-Flop eingetaktet. Dafür ist das Signal CLK IN DATA^ erforderlich.

Ansteuerung des Command-Registers

Das Command-Register muss mit der Aktivierung des Reset-Eingangs RST° des Mikroprozessors auf 00 zurückgesetzt werden, da die Resetsequenz bei der Adresse 1 00 0h startet. Aus diesem Grund wird das RST°-Signal direkt auf den CLR°-Eingang des Registers gelegt.

Im laufenden Betrieb wird der nach der Decoding-Phase ermittelte Standard- oder Sonderbefehl zu Beginn der Ausführungsphase in das Register eingetaktet. Dafür ist ebenfalls das Signal CLK IN DATA° erforderlich.

Ansteuerung des µStep-Counters

Der µStep-Counter muss mit der Aktivierung des Reset-Eingangs RST° des Mikroprozessors auf 0 zurückgesetzt werden, da die Resetsequenz bei der Adresse 1 00 0h startet. Aber auch beim Laden eines neuen Befehls muss der Zähler zurückgesetzt werden, da alle Befehle bei x xx 0h beginnen. Dafür ist das Signal CLR° erforderlich. RST° und CLR° werden über ein AND-Gatter an den CLR° - Eingang des Zählers geleitet. Ist nur eines der beiden Signale „low“, so wird der Zähler zurückgesetzt.

Im laufenden Betrieb wird der Zähler mit jeder steigenden Flanke von CLK inkrementiert, d.h. er würde nach dem Ende des Fetch-Cycles F2 zu Beginn der Ausführungsphase auf die nächste Adresse (Teilschritt) des alten Befehls erhöht, bevor er kurz darauf durch das CLR°-Signal für den neuen Befehl zurückgesetzt wird. Dies stellt im Prinzip kein Problem dar, da auf der „falschen“ Folgeadresse keine aktiven Signale mehr aus den Micro-Code-ROMs ausgegeben werden (alle Signale sind „high“). Um aber die falsche Adressierung zu vermeiden und unnötige, zumal kurz aufeinander folgende Schaltwechsel zu unterbinden, wird das Clock-Signal für den µStep-Counter zu diesem Zeitpunkt gesperrt, indem es ebenfalls mit der T2-Bedingung verknüpft wird. Nur wenn die Bedingung T2 nicht erfüllt ist, d.h. kein neuer Befehl geladen wird (T2 = 0), soll der Einzelschrittzähler inkrementiert werden. Um eine phasenkorrekte Taktung (steigende Flanke) zu erzielen, wird T2 mit dem invertierten Clock-Signal CLK° über ein NOR-Gatter verknüpft, sodass eine steigende Flanke (Zählimpuls) ausgelöst wird.

Sonderbefehle (P1-Befehle)

Bei Befehlen, die in der oberen Hälfte des ROMs liegen (P-Bit = 1), handelt es sich um Sonderbefehle bzw. P1-Befehle.

Beim 6502 ist die Ausführungspriorität von Hardwarefunktionen und Befehlen wie folgt festgelegt:

```
Höchste Priorität : Hardwarefunktion RESET°
dann :              Hardwarefunktion NMI°
dann :              Hardwarefunktion INT°
dann :              Standardbefehl  BRK
dann :              alle anderen Standardbefehle
```

Beim 65C02 wurde die Reihenfolge der Ausführungspriorität neu festgelegt:

```
Höchste Priorität : Hardwarefunktion RESET°
dann :              Standardbefehl  BRK
dann :              Hardwarefunktion NMI°
dann :              Hardwarefunktion INT°
dann :              alle anderen Standardbefehle
```

Der Unterschied liegt also im BRK-Befehl. Für den 6502-kompatiblen Betrieb wurde BRK (Befehlsbyte 00h) als Standardbefehl implementiert. Bei der aktuellen Schaltung wird der 65C02-Modus benutzt. Um den Standardbefehl BRK zwischen den Hardware-Funktionen RST° und NMI° entsprechend seiner Priorität einzuordnen, wird das Befehlsbyte 00h über eine Hardware dekodiert und künstlich, entsprechend seiner Priorität, zu einem Sonderbefehl (P1-Befehl) ungeformt. Er verhält sich damit quasi wie ein externes Signal (Hardwarefunktion).

Allen Hardwarefunktionen (RESET, BRK (s.o.), NMI und INT) sowie einigen speziellen Befehlsfunktionen müssen ebenfalls eigene Einsprungadressen in das ROM zugewiesen werden. Dazu werden durch die Hardware Pseudo-Befehle generiert, die theoretisch irgendwo im P1-Bereich des ROMs angesiedelt sein können.

Mit RST° auf low muss eine Einsprungadresse für die Initialisierungsroutine des Prozessor geladen werden. Leitet man das Signal auf den Pre-Set°-Eingang des Flip-Flops für das P-Bit, den CLR°-Eingang des Command-Registers und den CLR°-Eingang des µStep-Counters, so ergibt sich der Pseudo-Befehl

(Startadresse) 1000h für die Reset-Routine, ohne das weitere Hardware zur (De)-kodierung des Befehls erforderlich ist.

Um das Dekodieren der vorgeschriebenen Ausführungsprioriäten zu vereinfachen, wurden die Pseudo-Befehle (Adressen) für die in der Prioritätenliste folgenden Funktionen der Art festgelegt, dass, ausgehend von 1000h für Reset, im Command-Byte für jede Prioritätsstufe ein Bit reserviert wurde. Durch diese Anordnung ist die prioritätenabhängige Pseudo-Befehls-Generierung mit wenigen Bausteinen zu realisieren. Es stehen neben der Reset-Funktion bis zu acht weitere Prioritätsebenen zu Verfügung, von denen hier drei (BRK°, NMI°, INT°) Verwendung finden. Bei allen folgenden Pseudo-Adressen handelt es sich um modifizierte oder ergänzende Standardbefehle, die alle die gleiche viertrangige Priorität besitzen.

Befehl/ Funktion	**P-BIT**	**Command-Register**	**µStep-Counter**		**Hardwarefunktionen nach Priorität absteigend ausgewertet**
	<---binäre Adresse--->			**<hex>**	
RESET°	1	0000 0000	0000	1000	external Reset
BRK	1	0000 0001	0000	1010	artificial HW-BRK
NMI°	1	0000 0010	0000	1020	external non-maskable Interrupt
INT°	1	0000 0100	0000	1040	external Interrupt

Darüber hinaus wird zwischen kompletten und inkompletten Pseudo-Befehlen unterschieden. Bei den kompletten Pseudo-Befehlen (RESET°, BRK, NMI°, INT°, B8, B16) wurden alle Bits frei vom Entwickler festgelegt.

Die Additions- und Subtraktionsbefehle ADC bzw. SBC verhalten sich in Abhängigkeit vom D-Bit unterschiedlich. Ist das D-Bit gesetzt, so erfolgt nach der Addition bzw. Subtraktion eine Dezimalkorrektur, ist das D-Bit nicht gesetzt so unterbleibt die Korrektur. Es ist nun möglich, das D-Bit nach der

ursprünglichen Berechnung (innerhalb der Befehlsausführung) auszuwerten und die Entscheidung zu treffen, ob noch eine Korrektur ausgeführt werden soll (post-check). Man kann aber auch vor Beginn des Befehls feststellen, ob das D-Bit gesetzt oder nicht gesetzt ist und abhängig von dem Ergebnis in zwei vollkommen unabhängige Befehle, der eine mit, der andere ohne Korrektur, springen (pre-check). Anhand der beiden Befehlsgruppen ADC/SBC wird das Pre-Check-Verfahren vermittelt. (Wie innerhalb eines Befehls eine Entscheidung über dessen Fortführung getroffen wird, kann an den Beispielen für bedingte Sprünge, speziell bei den BBS/BBR-Befehlen nachgelesen werden).

Da das D-Bit bei 8-Bit-Prozessoren eher selten implementiert ist, wurden die Befehlsgruppen ADC und SBC im Dezimalmodus als Sonderbefehle ausgelegt, sodass auf eine zusätzliche Dekodierung in der ALU-SHIFT-Einheit verzichtet werden konnte und diese unverändert für anderer Prozessoren verwendet werden kann. Die Dekodierung erfolgt in der Steuereinheit, die für den Einsatz für andere Prozessoren ohnehin geringfügig modifiziert werden muss. Doch selbst der Verbleib der D-Bit-Dekodierung in der Steuereinheit wäre bei anderen Prozessoren unschädlich.

Die Additions- und Subtraktionsbefehle bei nicht gesetztem D-Bit liegen entsprechend ihrem Befehlsbyte als Standardbefehle im P0-Bereich. Ist das D-Bit gesetzt, wird das P-Bit auf high gesetzt und damit ein Pseudo-Befehl, basierend auf dem Original-Befehlsbyte, generiert. Diese modifizierten Befehle werden als inkomplette Pseudo-Befehle bezeichnet, da das Original-Befehlsbyte beibehalten bleibt und nur das P-Bit zusätzlich gesetzt wird. Natürlich ist es denkbar auch vollkommen andere Adressen für die beiden Befehlsgruppen zu generieren (komplette Pseudo-Befehle), doch wäre dazu weitere Hardware erforderlich.

In der folgenden Aufstellung sind alle Sonderbefehle aufgeführt, die in der Ausführungsreihenfolge hinter RESET, BRK, NMI und INT stehen und untereinander die gleiche Priorität besitzen.

Befehl/ Funktion	P- BIT	Command- Register	µStep- Counter		Befehlsfunktionen mit gleicher Priorität
	<-binäre Adresse->		<hex>		
B8	1	0000 1000	0000	1080	Pseudo-Command conditional branch with 8-Bit-Offset
B16	1	0001 0000	0000	1100	Pseudo-Command conditional branch with 16-Bit-Offset
ADC mit gesetztem D-Bit					
IMM	1	0110 1001	0000	1690	modified 0690-command
Z	1	0110 1001	0000	1650	modified 0650-command
Z,X	1	0110 1001	0000	1750	modified 0750-command
ABS	1	0110 1001	0000	16D0	modified 06D0-command
ABS,X	1	0110 1001	0000	17D0	modified 07D0-command
ABS,Y	1	0110 1001	0000	1790	modified 0790-command
(IND)	1	0110 1001	0000	1720	modified 0720-command
(IND,X)	1	0110 1001	0000	1610	modified 0610-command
(IND),Y	1	0110 1001	0000	1710	modified 0710-command
SBC mit gesetztem D-Bit					
IMM	1	0110 1001	0000	1690	modified 0690-command
Z	1	0110 1001	0000	1650	modified 0650-command
ABS	1	0110 1001	0000	16D0	modified 06D0-command
ABS,X	1	0110 1001	0000	17D0	modified 07D0-command
ABS,Y	1	0110 1001	0000	1790	modified 0790-command
(IND)	1	0110 1001	0000	1720	modified 0720-command
(IND,X)	1	0110 1001	0000	1610	modified 0610-command
(IND),Y	1	0110 1001	0000	1710	modified 0710-command

NMI - Aufbereitung

Das NMI°-Eingangssignal (aktiv bei negativer Flanke) wird direkt auf den Clock-Eingang eines 7474 geleitet, das als NMI-Auffangregister dient. Der Dateneingang ist permanent auf „high“ geschaltet. Dadurch wird gewährleistet, dass zu jedem Zeitpunkt, unabhängig vom aktuellen Zustand der Befehlsausführung, eine fallende Flanke am NMI-Eingang registriert wird und ein eingegangener NMI in positiver Logik angezeigt wird.

Mit der Aktivierung des Reset-Eingangs RST° des Mikroprozessors werden die NMI-Flip-Flops auf „0“ zurückgesetzt und damit erstmalig für das Empfangen eines NMIs vorbereitet.
Während der Ausführung der NMI-Sequenz wird das Auffangregister im ersten Teilschritt zurückgesetzt (s.u.), um für einen evtl. folgenden NMI vorbereitet zu werden. Während der Zeitdauer des Rücksetzens kann das Flip-Flop den Clock-Eingang nicht auswerten und somit eine weitere NMI-Anforderung nicht erkennen. Deshalb wird ein paralleles NMI-Auffangregister eingesetzt, dass während des Teilschritts 3 zurückgesetzt wird. Die Rücksetzzeitpunkte 0 und 3 wurden gewählt weil diese Werte leicht zu dekodieren sind. Es sind aber auch andere unterschiedliche Zeitpunkte innerhalb der NMI-Sequenz denkbar.
Durch die doppelte Ausführung und das zeitlich getrennte Rücksetzen der beiden Flip-Flops ist eine durchgehende Empfangsbereitschaft garantiert. Die Ausgänge der Flip-Flops werden über ein ODER-Gatter zusammengeführt.

Der aktuelle Zustand der zusammengeführten NMI-Auffang-Flip-Flops wird zu Beginn der zweiten Hälfte des ersten Fetch-Cycles (PC TO AB° und FETCH° = „low“) sowie CLK° = „high“ in das nachgeschaltete NMI-Flip-Flop übertragen.
Von dort wird das Active-high-Signal für einen NMI sowie sein invertierter Zustand zur Auswertung an den Pseudobefehlsgenerator weitergeleitet.

INT - Aufbereitung

Das INT°-Eingangssignal (activ-low-Pegel) wird invertiert (Weiterverarbeitung in positive Logik) auf den Dateneingang eines 7474 geleitet, das als INT-Auffangregister dient. Um einen pegelsensitiven Eingang zu realisieren, wird das Eingangssignal mit jeder steigenden Flanke von CLK in das Auffangregister eingetaktet (latched interrupt), sofern nicht bereits vorher ein INT erkannt wurde (Flip-Flop-Ausgang Q° über AND-Gatter mit CLK verknüpft). Ein externes Interrupt-Signal muss also mindesten eine Taktlänge aktiv („low“) sein, um sicher erkannt zu werden.
Mit der Aktivierung des Reset-Eingangs RST° des Mikroprozessors werden die INT-Flip-Flops auf „0“ zurückgesetzt und damit erstmalig für das Empfangen eines Interrupts vorbereitet.
Während der Ausführung der INT-Initialisierungssequenz werden die Auffangregister im ersten Teilschritt zurückgesetzt (s.u.), um für einen evtl. folgenden Interrupt vorbereitet zu werden.
Der aktuelle Zustand des INT-Auffang-Flip-Flops wird zu Beginn der zweiten Hälfte des ersten Fetch-Cycles (PC TO AB° und FETCH° = „low“) sowie CLK° = „high“ in das nachgeschaltete INT-Flip-Flop zur Auswertung für die folgende Decoding-Phase übertragen. Dazu wird der negierte Ausgang des Flip-Flops mit dem I-Bit aus dem Status-Register über ein NOR-Gatter verknüpft, sodass bei einem erfolgten und zugelassenen Interrupt ein „High“-Signal an den Pseudobefehlsgenerator weitergegeben wird.

0, 3, N und I - Signale

Während der INT- bzw. NMI-Ausführungssequenz soll jeweils beim Teilsschritt 0 und beim NMI noch einmal beim Teilschritt 3 ein CLR-Signal für das zeitgerechte Rücksetzen der jeweiligen Flip-Flops generiert werden. Diese beiden Signale (Zeitpunkte) lassen sich relativ einfach aus dem Zählerstand des µStep-Counters ableiten. Um zu verhindern, dass bei jedem Zählerstand 0 oder 3 ein CLR erfolgt, muss noch sichergestellt werden, dass dies nur während einer NMI- bzw. INT-Sequenz erfolgt. Dieses Ereignis wird jeweils am Q-Ausgang der zweiten NMI/INT-Flip-Flops angezeigt, doch wird das Flip-Flop u.U. durch den Status 0 des µStep-Counters schneller gelöscht als es für die NMI/INT-Auswertung erforderlich ist. Durch die künstlich verzögerte Weiterleitung des µStep-Counter-Zählerstands könnte das Problem umgangen werden. Hier jedoch wird eine andere Signalableitung angewendet, die die systembedingte Verzögerung ausnutzt. Ebenso wie die Signale 0 und 3 wird die Aktivität einer NMI- oder INT-

Initialisierungssequenz aus deren Mikro-Code-Adresse dekodiert. Dazu genügt es, das Page-Bit und die Adressleitung A7 für NMI (µCode-Adresse 1020h) und Page-Bit und A6 für INT (1040h) auszuwerten. Die resultierenden Signale N bzw. I zeigen die entsprechende Aktivität von NMI bzw. INT an.

Nibble Dekoder

Alle weiteren Sonderbefehle werden aus dem im Pre-Command-Register liegenden Befehlsbyte während des zweiten Fetch-Cycles dekodiert. Als Dekoder finden zwei 74154 Verwendung, die jeweils das obere und untere Halbbyte analysieren. Mit dieser Methode lassen sich bei matrix-förmig angeordneten Befehlssätzen alle Adressierung- und Funktionsarten selektieren, doch wird, wie oben beschrieben, bei diesem Befehlsdekoder ein anderes Verfahren verwendet. Die Nibble-Dekoder werden hier nur für die Analyse der Sonderbefehle eingesetzt.

Pseudo-BRK

Um den BRK-Befehl (Standardbefehl) beim 65C02 entsprechend seiner Priorität zwischen RST° und NMI zu platzieren, wird er in einen Sonderbefehl umgewandelt. Wird der Befehl 00h im Pre-Command-Register erkannt, d.h. beide Nibble-Dekoder geben an ihrem 0°-Ausgang ein „low“ aus, werden die beiden Ausgangssignale über ein NOR-Gatter zum active-high BRK-Signal und zeitgleich über ein OR-Gatter zum active-low BRK°-Signal, das für die weitere Auswertung ebenfalls benötigt wird.

Bedingte Sprünge

Es wird zwischen zwei Arten von bedingten Sprüngen unterschieden.

Bei “Branch on Bit Set“ – bzw. “Branch on Bit Reset“ - Befehlen wird die Entscheidung für einen Sprung innerhalb der Ausführungsphase ermittelt. Das resultierende Signal, das die Notwendigkeit eines Sprunges anzeigt, wird deshalb mit Branch internal (BI°) bezeichnet.

Bei den Befehlen BPL, BMI, BVC, BVS, BCC, BCS, BNE und BEQ wird die Sprungentscheidung während der Dekodierungsphase, also außerhalb der Ausführungsphase ermittelt. Zur Unterscheidung von BI-Befehlen werden sie als Branch-external-Befehle (BE) bezeichnet.

Während der 65C02 bei bedingten Sprungbefehlen nur Sprungweiten mit 8-Bit Offset erlauben, wurden bei dem hier vorliegenden Projekt alle Sprungbefehle auch mit 16-Bit Offset implementiert. Zur Unterscheidung der notwendigen Signale wird ihnen die entsprechende Bit-Breite des Offsets angehängt, sodass sich folgende Signal- bzw. Befehlsgruppenbezeichnungen ergeben:

Befehls-gruppe	**Befehlsbyte Low Nibble**	
BE8	0	Original-Befehlssatz
BE16	3	erweiterter Befehlssatz
BI8	F	Original-Befehlssatz
BI16	B	erweiterter Befehlssatz

Bei den neuen Befehlen mit 16-Bit-Offset entspricht das High-Nibble denen der 8-Bit-Befehle (s. Befehlsübersicht).

BE8 - Signal

Zur Analyse der Befehlsgruppe BE8 werden zunächst alle Befehle explizit mit ihren Bedingungen binär aufgeschlüsselt.

Befehl	hex	b 7	b 6	b 5	b 4	b 3	b 2	b 1	b 0	Sprung wenn	kein Sprung wenn
BPL	10	0	0	0	1	0	0	0	0	N-Bit = 0	N-Bit = 1
BMI	30	0	0	1	1	0	0	0	0	N-Bit = 1	N-Bit = 0
BVC	50	0	1	0	1	0	0	0	0	V-Bit = 0	V-Bit = 1
BVS	70	0	1	1	1	0	0	0	0	V-Bit = 1	V-Bit = 0
BCC	90	1	0	0	1	0	0	0	0	C-Bit = 0	C-Bit = 1
BCS	B0	1	0	1	1	0	0	0	0	C-Bit = 1	C-Bit = 0
BNE	D0	1	1	0	1	0	0	0	0	Z-Bit = 0	Z-Bit = 1
BEQ	F0	1	1	1	1	0	0	0	0	Z-Bit = 1	Z-Bit = 0

Ist das untere Nibble (b3 – b0) des Befehlsbytes = “0“ und das Datenbit b4 = “1“ (oberes Nibble = ungerade) handelt es sich um einen bedingten Sprungbefehl.

Man erkennt, dass b7 und b6 des Befehls jeweils ein Bit des Bedingungsregisters repräsentieren:

b 7	b 6	Befehlsgruppe
0	0	-> N-Bit-Befehle
0	1	-> V-Bit-Befehle
1	0	-> C-Bit-Befehle
1	1	-> Z-Bit-Befehle

Darüber hinaus entspricht b5 im Befehl jeweils dem Zustand des zu prüfenden Bits wenn die Sprungbedingung erfüllt ist.

Legt man die Statusregister-Bits an die Dateneingänge eines 74153 und die Daten-Bits b6 und b7 des Befehls im Pre-Command-Register an die Select-Eingänge A und B, so wird das zu dem Befehl gehörende CCR-Bit mit seinem aktuellen Pegel für die folgende Auswertung selektiert. Das gewählte CCR-Bit wird mit b5-Bit des Sprungbefehls verglichen.

Es wird deutlich, dass ein Sprung erfolgen soll, wenn das CCR-Bit gleich dem b5-Bit des Befehls ist und kein Sprung erfolgen soll, wenn die beiden Bits unterschiedlich sind.

Eingänge		**Ausgang**	**Aktion**
CCR Bit	**b 5**		
0	0	1	Sprung
0	1	0	kein Sprung
1	0	0	kein Sprung
1	1	1	Sprung

Bei dieser Funktion handelt sich um ein Exclusiv-NOR, das nur in Open-Drain-Version (74266) oder 7266 erhältlich und wenig verbreitet ist. Um die Schaltung, wie eingangs gefordert, mit verfügbaren Standard-Gattern zu realisieren, wird die Auswertung auf ein Exclusiv-OR umgestellt. Dazu wird das Bit b5 invertiert, sodass sich folgenden Wahrheitstabelle ergibt:

Eingänge		**Ausgang**	**Aktion**
CCR Bit	**b 5°**		
0	1	0	kein Sprung
0	0	1	Sprung
1	1	1	Sprung
1	0	0	kein Sprung

Für die sich anschließende Auswertung der Signale und die Pseudo-Befehlsgenerierung für bedingte Sprünge muss zunächst sichergestellt werden, dass die der Befehlsgruppe zugewiesene Priorität eingehalten wird. Dies trifft zu, wenn kein

ausgewertetes NMI- oder INT-Signal vorliegt (Ausgang OR-Gatter „low“ auf alle folgenden NORs). Wurde durch den Low-Nibble-Decoder eine 0 in der unteren Hälfte des Befehlsbytes erkannt (Ausgang 0° = „low“) und ist das Bit b4 = „high“ und damit invertiert = „low“, so steht am nachfolgenden NOR-Gatter ein „high“ an, das einen bedingten Sprungbefehl anzeigt. Verknüpft man dieses Signal mit dem EXOR-Ausgang (CCR-Bit Auswertung für Sprung, active-high, s.o.) mittels eines AND-Gatters, bedeutet ein „high“ an dessen Ausgang, dass ein BE8-Befehl auszuführen ist.

BE16-Signal

Die Auswertung der BE16-Befehle erfolgt nach dem gleichen Verfahren wie bei den B8-Befehlen. Da diese Befehle aber nicht in der Low-Nibble-Spalte „0“ sondern „3“ platziert wurden, muss dementsprechend der 3°-Ausgang des Low-Nibble-Dekoders ausgewertet werden.

Befehl	hex	b 7	b 6	b 5	b 4	b 3	b 2	b 1	b 0	Sprung wenn	Sprung wenn
BPL	13	0	0	0	1	0	0	1	1	N-Bit = 0	N-Bit = 1
BMI	33	0	0	1	1	0	0	1	1	N-Bit = 1	N-Bit = 0
BVC	53	0	1	0	1	0	0	1	1	V-Bit = 0	V-Bit = 1
BVS	73	0	1	1	1	0	0	1	1	V-Bit = 1	V-Bit = 0
BCC	93	1	0	0	1	0	0	1	1	C-Bit = 0	C-Bit = 1
BCS	B3	1	0	1	1	0	0	1	1	C-Bit = 1	C-Bit = 0
BNE	D3	1	1	0	1	0	0	1	1	Z-Bit = 0	Z-Bit = 1
BEQ	F3	1	1	1	1	0	0	1	1	Z-Bit = 1	Z-Bit = 0

BI8- und BI16-Signal

Auch hier wird das gleiche Verfahren wie oben verwendet. Die Signale BI8 und BI16 werden nur erzeugt, wenn kein NMI oder INT ansteht. Die Originalbefehlsgruppe für BBS und BBR steht innerhalb der Befehlsmatrix in der low-Nibble-Spalte „F“. Wurde in ALU/SHIFT-Einheit das BI°-Signal und durch den Low-Nibble-Dekoder ein F° generiert, so steht am Ausgang des NOR-Gatters ein „high“ an.
Die BI16-Befehle wurden in der Low-Nibble-Spalte „B“ untergebracht, da diese Spalte im Originalbefehlssatz nicht belegt ist und alle 16 BBS/BBR-Befehle bei gleicher High-Nibble-Belegung wie die der BI8-Befehle angeordnet werden können. Für die Auswertung muss dann das B°-Signal des Low-Nibble-Dekoders verwendet werden, um ein BI16-Signal mit „high“ zu erzeugen.

BE8/BI8- und BE16/BI16-Signale

Da die Sequenzen für die bedingten Sprungbefehle sowohl für BE8 und BI8 (relativer Sprung mit 1 Byte Offset)als auch für BE16 und BI16 (relativer Sprung mit 2 Byte Offset) jeweils gleich sind, werden auf der einen Seite BE8 und BI8 und auf der anderen Seite BE16 und BI16 über ODER-Gatter zusammengefasst, sodass die beiden Signale BE8/BI8 und B16/BI16 entstehen.

Generierung der kompletten Pseudo-Befehle

Mit den nun in positiver Logik zu Verfügung stehenden Signalen BRK, NMI, INT, BE8/BI8 und BE16/BI16, die gemäß ihrer erforderlichen Priorität ausgewertet wurden, d.h. nur ein Signal ist nach der Analyse „high“, können die Pseudo-Befehle erzeugt werden. Dazu werden die Signale entsprechend ihrer Priorität auf die Eingänge des Bustreibers gegeben, der die Sonderbefehle an das Command-Register leitet.

Funktion	Treiber Eingang	Priorität
BRK	D0	1
NMI	D1	2
INT	D2	3
BE8/BI8	D3	4
BE16/BI16	D4	4
Keine	D5	-
Keine	D6	-
Keine	D7	-

Da der Pseudo-Befehl 1 00 0 für die RST-Funktion durch direktes Ansteuern der Pre-Set bzw. Clear-Eingänge an P-Bit, Command-Register und µStep-Counter mit höchster Priorität (0) und asynchron erzeugt wird, muss er hier nicht berücksichtigt werden.
Die nicht benutzen Eingänge des Treibers werden auf „0“ (Masse) gesetzt. Es ergeben sich folgende Einsprungadressen in das P1-ROM.

Befehl/ Funktion	P-BIT	Command-Register	µStep-Counter		nach Priorität absteigen
	<---binäre Adresse--->			<hex>	
BRK	1	0000 0001	0000	1010	artificial HW-BRK
NMI°	1	0000 0010	0000	1020	external non-maskable Interrupt
INT°	1	0000 0100	0000	1040	external Interrupt
BE8/BI8	1	0000 1000	0000	1080	conditional branch with 1 Byte Offset
BE8/BI8	1	0001 0000	0000	1100	conditional branch with 2 Byte Offset

Alle fünf Signale werden auch auf den Eingang eines NOR-Gatters geleitet. Besteht eine Anforderung für einen Sonderbefehl (ein Signal auf „high“), so geht der Ausgang des NOR-Gatters auf „low“ und aktiviert den Bustreiber für Sonderbefehle. Mit dem invertierten Ausgang des NORs wird gleichzeitig der Bustreiber für die Standard-Befehle abgeschaltet. Darüber hinaus wird bei einem anstehenden Sonderbefehl der Low-Pegel des NOR-Gatters mit dem DCMD°-Signal (Beschreibung folgt) über ein NAND-Gatter verknüpft, sodass am Eingang des Page-Bit Flip-Flops das erforderliche „high“ ansteht.

Generierung der inkompletten Pseudo-Befehle

Die Sequenzen für die Additions- und Subtraktionsbefehle bei nicht gesetztem D-Bit (Hex-Modus) werden als Standardbefehle im P0-Bereich der µCode-ROMs abgelegt. Ist das D-Bit gesetzt (Dezimal-Modus), so werden die ADC- und SBC-Befehle als inkomplette Pseudo-Befehle ausgelegt, d.h. das Original-Befehlsbyte aus dem Pre-Command-Register wird in das Command-Register übernommen und das P-Bit, dass einen Sonderbefehl anzeigt, wird gesetzt.
Nach dem Laden eines neuen Befehls in das Pre-Command-Register wird über die zwei Nibble-Dekoder festgestellt, ob es sich um einen ADC- bzw. SBC-Befehl handelt. Aus der Anordnung in der Befehlsmatrix erkennt man, dass die High-Nibble 6 und 7 sowie E und F und die Low-Nibble 1, 2, 5, 9 die beiden Befehlsgruppen kodieren.

ADC-Befehle	SBC-Befehle
61	E1
65	E5
69	E9
6D	ED
71	F1
72	F2
75	F5
79	F9
7D	FD

Die Ausgangssignale 6°, 7°, E° und F° des High-Nibble-Dekoders und die Ausgangssignale 1°, 2°, 5°, 9° und D° des Low-Nibble-Dekoders werden jeweils auf ein NAND-Gatter geführt und über ein AND-Gatter zusammengefasst. Handelt es sich um einen ADC-/SBC-Befehl, liegt an dessen Ausgang jetzt ein „high“ an. Mit dem Status des D-Bits über ein NAND-Gatter zusammengeführt, ergibt sich das Signal DCMD°, dass bei einem erkannten ADC-/SBC-Befehl und D-Bit = „high“ einen Low-Pegel annimmt. DCMD° wird anschließend noch über ein NAND-Gatter mit dem P-Bit-Pegel der kompletten Pseudo-Befehls-Gruppe (fünfach-NOR) zusammengefasst. Liegt jetzt ein kompletter oder inkompletter Pseudo-Befehl vor, steht am Dateneingang des P-Flip-Flops ein „high“.

In der Projektdatei 65C02µC.PRJ lassen sich alle Befehle mit ihren Einzelschritten und den dazugehörenden Steuersignale anzeigen.

Start-Logik

Die Schaltung kann wahlweise per Schalter im Einzelschritt-Modus oder mit einem Taktsignal betrieben werden. Der gewünschte Modus wird über den Clock-Mode-Schalter auf der Frontplatte gewählt.

Um den Prozessor, wie in der Realität, erst nach Betätigung des RST°-Schalters anlaufen zu lassen, insbesondere wenn er ausschließlich mit einem Taktgeber betrieben werden sollte (was die Gefahr birgt, dass die Schaltung mit dem Simulationsstart unkontrolliert anläuft und unerwünschte Aktionen auslöst), wird der Systemtakt CLK auf einen Eingang eines AND-Gatters geführt. Am zweiten Eingang des AND-Gatters liegt der Ausgang eines 7474-Flip-Flops, das durch einen Power-On-Impuls auf seinen CLR-Eingang zu Beginn der Simulation zurückgesetzt wird (Ausgang Q = „low“). Dadurch bleibt der Ausgang des AND-Gatters zunächst auf „low“. Erst durch das Aus- und wieder Einschalten des RST-Schalters wird mit dem invertierten RST-Signal eine positive Flanke am Flip-Flop erzeugt, die den High-Pegel am Dateneingang auf den Ausgang Q durchschaltet und das AND-Gatter für das Taktsignal frei schaltet. Das Taktsignal wird für die weitere Auswertung auch invertiert benötigt (CLK°). Um die Flanken von CLK und CLK° möglichst gleichzeitig zu Verfügung zu haben, wird das CLK-Signal mit einem AND-Gatter verzögert, um die Gatterlaufzeit des Inverters für die Erzeugung von CLK° zu kompensieren.

Manueller Betrieb des Steuerwerks

Im manuellen Betrieb der Datei 65C02CU.PRJ kann die Adressgenerierung für alle Befehle überprüft werden. Das schließt selbstverständlich alle Sonderbefehle mit ihren kompletten und inkompletten Pseudo-Befehlen ein. Die Befehlsbytes sind für den Testbetrieb über die HEX-Eingabe einzustellen.

Reset (1000h)

Zum (Neu-)Start muss der RST-Schalter einmal auf OFF und wieder auf ON geschaltet werden. Die Startadresse 1000h für die Reset-Sequenz wird aktiviert.

CLK

Mit den CLK-Schalterstellungen OFF/ON (klick-klick) werden die einzelnen Teilschritt-Adressen ausgegeben. Solange kein neuer Befehl geladen wird (Fetch-Cycle F1 und F2), gibt der µStep-Counter nacheinander alle 16 möglichen µStep-Adressen 0 – Fh aus und beginnt dann wieder bei 0.

Laden eines neuen Befehls:

Neues Befehlsbyte einstellen

CLK auf ON

FE° auf OFF (Fetch-Phase 1, F1)
PC2AB° auf OFF

CLK OFF-ON klick-klick

FE° auf OFF (Fetch-Phase 2, F2)
PC2AB° auf ON

CLK OFF-ON klick-klick

FE° auf ON (Ausführungsphase 1)

BRK (1010h)

Ist der BRK-Befehl 00h eingestellt, so wird dessen Adresse mit der nächsten Fetch-Phase geladen, unabhängig des Status von NMI oder INT.

NMI (1020h)

Der NMI-Eingang ist flanken-sensitiv. Der Schalter muss also zum Auslösen eine NMIs zu einem beliebigen Zeitpunkt auf OFF geschaltet werden und sollte dann wieder auf ON zurückgestellt werden. Mit der nächsten Fetch-Phase wird die Adresse der NMI-Sequenz geladen, sofern kein BRK-Befehl vorliegt.

INT (1040h)

Der INT-Eingang ist pegel-sensitiv. Der Schalter muss also zum Auslösen eine INTs auf OFF geschaltet und bis zur nächsten positiven Flanke von Clock OFF bleiben. Danach sollte der Schalter wieder auf ON zurückgestellt werden, da ansonsten ein INT auf den anderen folgt. Ist das I-Bit auf OFF (enable interrupt) wird mit der nächsten Fetch-Phase die Adresse der INT-Sequenz geladen, sofern kein BRK-Befehl oder NMI vorliegt.

Bedingte Sprungbefehle (Branch on CCR-Bit) mit 8-Bit-Offset (1080h)

Ist die Bedingung des Sprungbefehls erfüllt, so wird die Adresse 1080h geladen, ist die Sprungbedingung nicht erfüllt, wird die Adresse des eingestellten Standardbefehls geladen.

Bedingte Sprungbefehle (Branch on CCR-Bit) mit 16-Bit-Offset (1100h)

Ist die Bedingung des Sprungbefehls erfüllt, so wird die Adresse 1100h geladen, ist die Sprungbedingung nicht erfüllt, wird die Adresse des eingestellten Standardbefehls geladen.

Bedingte Sprungbefehle
(Branch on Bit Set/Reset) mit 8-Bit-Offset (1080h)

Ist die Bedingung des Sprungbefehls erfüllt, so wird die Adresse 1080h geladen, ist die Sprungbedingung nicht erfüllt, wird die Adresse des eingestellten Standardbefehls geladen.

Bedingte Sprungbefehle
(Branch on Bit Set/Reset) mit 16-Bit-Offset (1100h)

Ist die Bedingung des Sprungbefehls erfüllt, so wird die Adresse 1100h geladen, ist die Sprungbedingung nicht erfüllt, wird die Adresse des eingestellten Standardbefehls geladen.

65C02CUmin.PRJ

Verzichtet man auf die NMI- und INT-Eingänge und deren Auswertung, die bedingten Sprünge mit 16-Bit-Offset und den in der Praxis ohnehin eher unbedeutenden Dezimal-Modus für ADC- und SBC-Befehle, so reduziert sich die Ablaufsteuerung auf wenige Bauteile. Sie realisiert aber auch in dieser Form noch die komplette Adressierung der µCode-ROMS für einen 8-Bit Befehlssatz (mit den oben beschrieben ROMS die eines 6502), besitzt jedoch nur noch die zwei Sonderbefehle RST (1000h) und die Sequenz für Sprünge bei erfüllter Bedingung (1080h). Das untere Nibble der Sprungbefehle wird über ein vierfach-NOR-Gatter (Ausgang „high“ wenn Nibble = 0) und das obere Halbbyte mittels des D4-Bits dekodiert.

Man erkennt bei dieser minimalistischen Form, dass die gesamte Ablaufsteuerung nahezu hardwareunhabhängig vom Befehlssatz des Prozessors ist. Lediglich für die Auswertung der bedingten Sprünge sind je nach der Anordnung in der Befehlsmatrix und den Bits im Bedingungsregister ggfs. kleinere Umkodierungs-arbeiten notwendig, wenn man diese Methode der Ablaufsteuerung für andere Prozessoren einsetzen möchte. Unter diesem Gesichts-punkt zeigt sich spätestens jetzt der Vorteil des Konzepts dieser Ablaufsteuerung.

Der Dekodieraufwand für die Ablaufsteuerung ist dermaßen reduziert, dass der Fetch-Cycle auf einen Takt verkürzt werden kann. Das Zwischenspeichern eines neuen Befehls im Pre-Command-Register für den Dekodiervorgang kann entfallen. Dazu ist das Pre-Command-Register zu entfernen und die entstehende Lücke durch acht Leitungsbahnen zu schließen. Ohne Pre-Command-Register entfällt auch der Fetch-Cycle F1, d.h. das Eintakten des neuen Befehlsbytes durch die Schalterstellung FE° „low“ und PC2AB° „low“ bei fallender CLK-Flanke. Das Signal PC2AB° wird somit überhaupt nicht mehr benötigt, die Auswertung eines neu anstehenden Befehlsbyte wird nur noch durch FE° = „low“ ausgelöst. Mit BI° = „low“ für den bedingten Sprung bei BBS/BBR- Befehlen wird weiterhin auf die µCode-Adresse 1080h verzweigt.

Nicht nur für andere 8-Bit-Mikroprozessortypen lässt sich diese Ablaufsteuerung durch geringe Modifikationen einsetzen, sondern generell für Programmsteuerungen, die Befehle mit unterschiedlicher Schrittlänge und Bedingungs-Bits mit den ihnen zugeordneten Sprungbefehle benötigen. Die erforderlichen Befehle bzw. Befehlssätze lassen sich nach individuellen

Erfordernissen gestalten und die dazu gehörenden Teilschritte in den µCode-ROMs mit beliebiger Bit-Breite ablegen.

Makro-Assembler

"HAND-ASSEMBLER"

Kleinere (Demo-)programme lassen sich direkt durch Eingabe des HEX-Codes in das RAM bzw. ROM erstellen. Spätestens beim nachträglichen Einfügen von Befehlen (Verschieben des restlichen Codes und Neuberechnung der Sprung-Offsets) hört hier aber der Spaß auf.

MAKRO-ASSEMBLER

Für das komfortable Erstellen von Programmen ist der Einsatz eines (Makro)-assemblers unerlässlich.

Das Problem ist, dass die von Assemblern produzierten Ausgabedateien nicht so ohne weiteres als Eingabedatei für ein Digital-ProfiLab-RAM bzw. ROM verwendet werden können, da hierfür eine plain-Byte-by-Byte-Textfile erforderlich ist.

Für die .LST-Datei des CA65-Assemblers liegt ein Q&D-Konverter bei. Er ist als DOS - Anwendung realisiert, da es sich beim CA65 ebenfalls um ein DOS-Programm handelt und man am besten die ganze Entwicklungsprozedur in der DOS-Box von Windows durchführt.

Diesen Makro-Assembler erhalten Sie kostenlos und lizenzfrei von der Internet-Seite

www.cc65.org

in der Sektion Download unter dem Punkt available for download.

Laden Sie sich die Datei

cc65-win32-2.13.1-1.zip oder
cc65-win32-2.13.1-1.exe (inkl. Beschreibung)

auf den Rechner in ein separates Verzeichnis, um sie dort zu extrahieren. Kopieren Sie die Datei CA65.EXE aus dem Verzeichnis in Ihr Entwicklungsverzeichnis, das nun folgende Dateien enthalten sollte:

CA65.EXE	Makroassembler, er benötigt für den Aufruf Parameter, deshalb ist er über die Batch-Datei A.BAT (s.u.) aufzurufen
A. BAT	assembliert eine Source-Datei <dateiname>.ASM Beispiel: A 6502DEMO (Aufruf ohne Datei-erweiterung .ASM) und erstellt neben der Object-Code-Datei eine .LST- Datei als Eingabedatei für den Konverter (s. LST2TXT.EXE)
C.BAT	konvertiert eine Datei <dateiname>.LST in <dateiname>.TXT, die in Digital-Profi-Lab direkt in das RAM/ROM geladen werden kann (Kurzform von LST2TXT.EXE) Beispiel: C 6502DEMO (Aufruf ohne Dateierweiterung .LST)
AC.BAT	führt A.BAT und C.BAT in einem Aufwasch durch. Beispiel: AC 6502DEMO (Aufruf ohne Dateierweiterung)
LST2TXT.EXE	konvertiert die .LST–Ausgabe des CA65- Makro-Assemblers in das für ProfiLab erforderliche .TXT-Format, die in Digital-Profi-Lab direkt in das RAM/ROM geladen werden kann (Aufruf ohne Dateierweiterung .LST.) Der Aufruf erfolgt über C.BAT bzw. AC.BAT (s. dort). Da der 65C02 die Zero-Page-Adressierung unter-stützt, sollten Programme nicht im Adressbereich 0000h - 00ffh liegen, um den Zero-Page-Adress-bereich für den schnellen Datenzugriff frei zu halten. Der Stackpointer-Bereich liegt im Bereich 0100h - 01ffh. Um Konflikte zu vermeiden, dürfen Programme hier nicht residieren!!

Wird der SWI-Befehl verwendet, muss der Bereich 0200h – 02FFH für die SWI-Vektoren freigehalten werden!!
Um diese Voraussetzungen zu erfüllen, schreiben Sie in Ihren Quellcode als erste Programmzeile den Pseudo-Befehl .org $0300h (oder höher). Fügen Sie hinter dem Text ".org" genau ein Leerzeichen und ein Dollarzeichen ein, gefolgt von der vier-stelligen Startadresse! Der Konverter analysiert die .ORG - Anweisung und fügt die entsprechende Anzahl Nullen in die TXT - Datei vor dem Code ein. Die konvertierte TXT-Datei wird dann in 65C02.PRJ aktiviert, indem der RAM-Baustein (Makro) geöffnet und die TXT-Datei geladen wird.

Achtung:
Digital-ProfiLab überschreibt beim Laden von Programmcode in das RAM/ROM den kompletten Bereich zwischen Code-Ende und Speicherende mit Nullen. Ein schrittweises Laden von Programmabschnitten an bestimmte Adressen ist nicht möglich.

Will man, wie im Beispiel 6502DEMO.ASM, mehrere Programmabschnitte (Hauptprogramm, INT-Rroutine, NMI-Routine) auf bestimmte Adressen in den Speicher laden, so kann durch mehrere .ORG-Anweisungen der jeweils gewünschte Abstand erreicht werden. Dadurch erspart man sich das ständige Anpassen der Interruptvektoren im ROM. LST2TXT berechnet die erforderlichen Abstände und füllt die Lücken mit Nullen.
Dies gilt nur für den Gebrauch mit CA65.EXE in der Version 2.13, LST2TXT.EXE und Digital-ProfiLab.

Achtung:
absolute Sprünge von einem .org-Bereich in einen anderen werden bei dieser Version noch nicht umgesetzt und müssen ggfs. per Hand angepasst werden.

Vor der Verwendung des Codes für reguläre Zwecke (z.B. Linken) sind die zusätzlichen ORGs ggfs. zu entfernen und der Code neu zu assemblieren.

```
6502DEMO.ASM Ein kleines Demonstrationsprogramm inkl. NMI-,
             INT- und Subroutinen. Die Vektoren liegen für:

             RST  auf $FFFCh  ROM-Adresse 03FCh (LB) Inhalt 00h
                      $FFFDh  ROM-Adresse 03FDh (HB) Inhalt 03h
                              = 0300h

             NMI  auf $FFFAh  ROM-Adresse 03FAh (LB) Inhalt 00h
                      $FFFBh  ROM-Adresse 03FBh (HB) Inhalt 05h
                              = 0500h

             INT  auf $FFFEh  ROM-Adresse 03FEh (HB) Inhalt 00h
                      $FFFFh  ROM-Adresse 03FFh (LB) Inhalt 04h
                              = 0400h
```