Technische Universität Darmstadt
Institut für Datentechnik
Fachgebiet Rechnersysteme
Prof. Dr.-Ing.H.Eveking

Studienarbeit

von
Joachim Horch

Entwurf eines RISC-Prozessors in der Hardwarebeschreibungssprache VHDL

Betreuer: Prof. Dr.-Ing.H.Eveking

Juni 1997

Inhalt

Abbildungen

1 Einleitung

In der Entwicklung von technischen Produkten wird immer häufiger auf den Einsatz von Prozessoren zurückgegriffen. Das Angebot auf dem Markt reicht von einfachen Mikrocontrollern bis zu modernen superskalaren Hochleistungsprozessoren. Daraus folgt, daß der Entwurf von digitalen Schaltungen ständig an Bedeutung gewinnt. Dabei sind der korrekte Entwurf und die rasche Markteinführung des Produktes gleichermaßen wichtig.

Aus diesen Gründen werden für einen Ingenieur ausreichende Erfahrung in der Anwendung einer Hardwarebeschreibungssprache und die Kenntnis der Abläufe in einem Prozessor erforderlich. Dies ist für den Entwurf von neuen Prozessoren und eingebetteten Systemen ebenso wesentlich wie für die Entwicklung der zugehörigen Software.

Der fehlerfreie Entwurf von Hardwaresystemen ist ein aktives Forschungsgebiet. Dabei sollen Verfahren entwickelt werden, die den Nachweis erlauben, daß beispielsweise ein superskalarer, pipeline-implementierter Prozessor frei von Entwurfsfehlern ist.

Die grundlegenden Prinzipien und Methoden für den Entwurf von Rechnerarchitekturen werden besonders anschaulich in dem Buch 'Computer Architecture: A Quantitative Approach' von John L. Hennessy und David A. Patterson vermittelt. Dort wird unter anderem das Modell eines RISC-Prozessors mit dem Namen 'DLX' [1; S.96 ff.] beschrieben.

Ziel dieser Arbeit ist es, eine ausführlich dokumentierte und damit verständliche VHDL-Beschreibung einer pipeline-implementierten DLX-Maschine anzufertigen. Der Prozessor soll einen Daten-Cache und einen Befehls-Cache besitzen und die Implementierung von virtuellem Speicher und effektiver Adressierung unterstützen. Hierzu gehört der Aufbau einer Adreßumsetzung und die präzise Behandlung von Ausnahmesituationen der CPU und der externen Interrupts. In dem technologieunabhängigen Entwurf sind keine Verzögerungen durch Signallaufzeiten zu beachten. Das Taktsignal der CPU kann daher eine beliebige Frequenz aufweisen. Freigabesignale für alle Register erlauben die Übernahme der anliegenden Eingangsdaten an der positiven Taktflanke.

Diese Ausarbeitung kann an der Hochschule in weiterführenden praktischen Veranstaltungen genutzt und dabei möglicherweise ausgebaut werden. Studenten können so die bisher erworbenen theoretischen Grundlagen zur Arbeitsweise von pipeline-implementierten RISC-Maschinen an einem praktischen Prozessormodell nachvollziehen.

Von den Beschreibungsmöglichkeiten, die VHDL bietet, wird hier nur ein kleiner Teil benutzt. Dies soll das Verwenden von Verifikationswerkzeugen zum Überprüfen der entworfenen Hardware erlauben. Damit können dann Fehler bei der Ausarbeitung der Steuerlogik, die durch nicht bedachte Zustände des Prozessors entstehen, erkannt werden.

2 Festlegen der Architektur

In diesem Kapitel wird der Entwicklungsverlauf der Architektur kurz dargestellt. Ausgangspunkt ist die im Buch von Hennessy und Patterson beschriebene einfache DLX-Pipeline [1; S.132 ff.]. Diese wird Schritt für Schritt erweitert.

2.1 Grundlagen für eine einfache pipeline-implementierte Maschine

Die folgende Einführung in den Entwurf von pipeline-implementierten Maschinen faßt die wesentlichen Merkmale und Schwierigkeiten einer solchen Architektur zusammen. Dieser Überblick kann jedoch kein Lehrbuch ersetzen.

Eine Pipeline ist wie ein Fließband: Jeder Schritt in der Pipeline vollendet einen Teil des Befehls. Wie bei einem Produktionsfließband ist die bei einem Befehl auszuführende Arbeit in kleine Abschnitte zerlegt, von denen jeder einen Teil der Zeit zur Bearbeitung des ganzen Befehls benötigt. Jeder dieser Schritte wird Pipeline-Stufe (pipe stage) genannt. Ziel des Pipelinings ist die Reduzierung der mittleren Ausführungszeit pro Befehl. Dies wird erreicht, indem man die Parallelität zwischen den Befehlen in einem sequentiellen Befehlsstrom ausnutzt. Pipelining ist eine Technik, die für den Programmierer nicht sichtbar ist. Dies bedeutet, daß sich die pipeline-implementierte Maschine wie eine sequentiell arbeitende Maschine verhalten muß, die einen Befehl nach dem anderen in der durch das Programm vorgegebenen Reihenfolge ausführt. Man kann dies als die Pflicht zum In-Reihe-Beenden von Befehlen bezeichnen.

Die Pipeline der DLX besteht aus den folgenden fünf Befehlsverarbeitungsphasen.

2.1.1 Befehl holen (instruction fetch)

In dieser Phase holt der Prozessor den vom Befehlszähler angezeigten Befehl aus dem Cache bzw. dem Speicher. Bei sequentieller Bearbeitung des Programms wird anschließend der Befehlszähler um vier Bytes erhöht und zeigt damit auf den nächsten Befehl. Die Ausführung eines Sprungbefehls setzt den Befehlszähler auf die Adresse des Sprungziels. Das Holen der Befehle wird im nächsten Takt an der Adresse des neuen Befehlszählers fortgesetzt.

Falls eine Ausnahmebehandlung gestartet wird, muß nicht nur der Befehlszähler auf die Anfangsadresse der Ausnahmeroutine gesetzt, sondern auch der externe Interrupt gesperrt werden. Nur so ist eine sichere Ausführung der Ausnahmebehandlung möglich. Da dieser Vorgang den Status der Maschine verändert, dürfen Ausnahmebehandlungen, die durch eine Befehl verursacht wurden, nur dann gestartet werden, wenn alle vorangehenden Befehle die Pipeline verlassen haben [1; S.179 ff.]. Dies erzwingt auch die Pflicht zum In-Reihe-Beenden von Befehlen. Wenn zum Beispiel in dieser Befehlsverarbeitungsphase ein Transferfehler des Speichers (transfer-error, bus-error) auftritt, dann muß dieses Ereignis zunächst in die Pipeline-Register geschrieben und Schritt für Schritt zum Ende der Pipeline geschoben werden. Erst dann ist sichergestellt, daß alle vorangegangenen Befehle die Pipeline verlassen haben. Der Prozessor startet jetzt die Ausnahmebehandlung und kann dabei den Maschinenstatus verändern.

2.1.2 Befehlsdecodierung und Register holen (instruction decode and register fetch)

In diesem Bearbeitungsschritt decodiert der Prozessor den vorher geholten Befehl und übergibt ihn gegebenenfalls zusammen mit seinen Operanden der Ausführungs- und Adreßberechnungsphase.

Gewöhnlich stellt der Registersatz die Operanden oder auch Daten eines Befehls zur Verfügung. In einer pipeline-implementierten Maschine ist es jedoch möglich, daß der Registersatz zum Zeitpunkt des Lesens nicht den Wert enthält, der sich in einer sequentiell arbeitenden Maschine am Start des gleichen Befehls ergeben würde. Diese Situation entsteht zum Beispiel dann, wenn ein vorangehender Befehl in ein Quellenregister des aktuellen Befehls schreibt, aber dieser Schreibvorgang noch nicht von der Pipeline ausgeführt wurde. Diesen Konflikt ordnet man der Gruppe der Daten-Hazards zu und nennt ihn Read-After-Write-Hazard [1; S.151 ff.]. Es sind nun zwei Fälle zu unterscheiden. Im ersten sind die benötigten Daten schon in einem Pipeline-Register enthalten. Dann ist es möglich, diese Daten schon vor dem Schreiben in das Zielregister zu verwenden und direkt an die Ausführungs- und Adreßberechnungsphase weiterzuleiten (forwarding). Die Befehlsausführung kann so ungehindert fortfahren. Im zweiten Fall sind die Operanden wegen einer Ausführungsverzögerung in keinem Register der Pipeline enthalten und können so auch nicht weitergeleitet werden. Aus diesem Grund kann der Prozessor die Bearbeitung nicht fortsetzen und fügt deswegen so lange Wartezyklen (stalls) in die Pipeline ein, bis die Daten zur Verfügung stehen.

Während der Bearbeitung von Sprungbefehlen wird der Befehlszähler mit der Adresse des Sprungziels überschrieben. Die Durchführung bedingter Verzweigungen setzt voraus, daß die Daten zur Auswertung der Bedingung bereitstehen und die Verzweigung als ausgeführt (taken branch) erkannt wird.

Ein undefinierter Befehl oder der Aufruf eines Betriebssystemdienstes lösen am Ende der Pipeline eine Ausnahmebehandlung aus.

2.1.3 Ausführung und Adreßberechnung (execution and effective address calculation)

In dieser Bearbeitungsphase führt der Prozessor alle arithmetisch-logischen Operationen aus. Dies umfaßt die Adreßberechnung für die Lade- bzw. Speicherbefehle und die arithmetisch-logischen Befehle. Deren Ergebnis steht nach dieser Phase für die folgenden Befehle zur Verfügung.

Eine undefinierte arithmetisch-logische Funktion löst am Ende der Pipeline eine Ausnahmebehandlung aus.

2.1.4 Speicherzugriff (memory access)

In dieser Pipeline-Stufe wird der Speicherzugriff von Lade- bzw. Speicherbefehlen durchgeführt. Arithmetisch-logische Befehle passieren diese Phase ohne eine Operation.

Das Ergebnis von Lesezugriffen steht nach dieser Phase für die folgenden Befehle zur Verfügung.

Die Pflicht zum In-Reihe-Beenden von Befehlen erfordert es, daß das Schreiben in den Speicher erst dann erfolgen darf, wenn alle vorangegangenen Befehle die Pipeline verlassen haben.

Ausrichtungs- und Transferfehler lösen am Ende der Pipeline eine Ausnahmebehandlung aus. Da nach dem Auftreten von Ausrichtungsfehlern Speicherzugriffe unterdrückt werden, haben Ausrichtungsfehler eine höhere Priorität als Transferfehler.

2.1.5 Rückschreiben zum Zielregister (write back)

Im letzten Schritt der Pipeline werden die Resultate der arithmetisch-logischen Befehle und der Ladebefehle in die Zielregister geschrieben. Diese Befehle werden damit in der Reihenfolge abgeschlossen, in der sie auch im Programmfluß auftreten.

Während des Starts einer Ausnahmebehandlung kann der Maschinenstatus sicher gerettet und später wiederhergestellt werden.

Die folgende Skizze zeigt die Ausführungsphasen der nun entstandenen einfachen DLX-Pipeline.

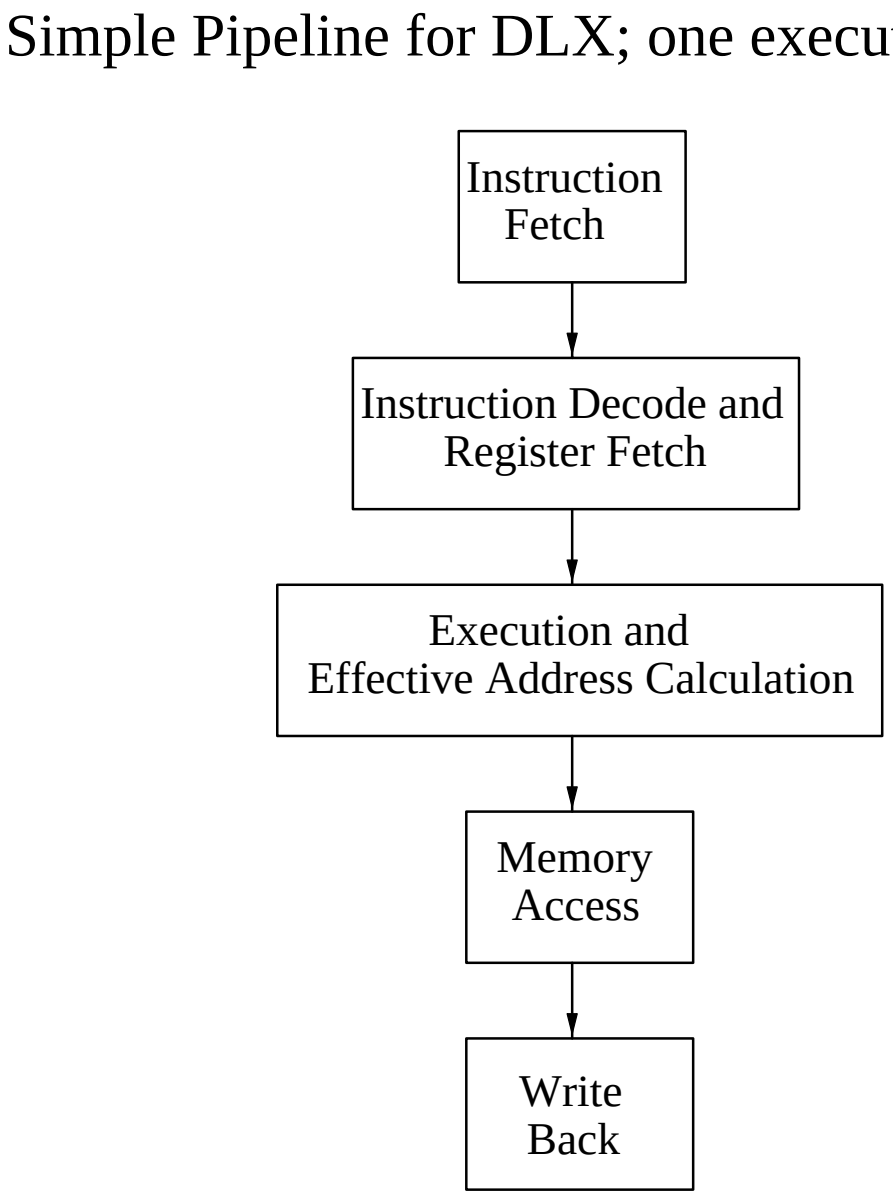

Abbildung 2-1: Einfache DLX-Pipeline

2.2 Interne Sonderregister

Der Prozessor benötigt spezielle Register zur Steuerung und Verwaltung von Ausnahmebehandlungen. Während dem Start einer Ausnahmebehandlung muß die Annahme von externen Interrupts gesperrt und der aktuelle Maschinenstatus gerettet werden. Hierzu ist ein Flag notwendig (Interrupt-Enable-Flag), das anzeigt, ob ein externer Interrupt zugelassen ist. In ein weiteres Register schreibt der Prozessor das aktuelle Interrupt-Enable-Flag und die Wort-Adresse, an der das unterbrochene Programm nach der Ausnahmebehandlung fortgesetzt werden soll. Beide Register müssen auch vom Betriebssystem, also durch Software, lesbar und schreibbar sein. Hierzu gibt es prinzipiell zwei Möglichkeiten. Die erste besteht darin, den Sonderregistern feste Adressen im Speicherraum zuzuordnen und sie so mit Hilfe der Lade- bzw. Speicherbefehle anzusprechen. Alternativ hierzu können besondere Befehle zum Lesen und Schreiben der Register eingeführt werden. Dies erfordert zusätzlichen Aufwand. Der Vorteil liegt jedoch darin, daß der Speicherraum vollständig zur freien Verfügung bleibt und auf Befehlsebene leicht eine Trennung zwischen den Zugriffen auf die Sonderregister und den Hauptspeicher möglich ist. Dies ist besonders dann erwünscht, wenn eine große Zahl von Sonderregistern mit komplexen Funktionen in einem Prozessor vorhanden sind. Die Pflicht zum In-Reihe-Beenden von Befehlen erfordert, daß diese Register erst dann geschrieben werden dürfen, wenn alle vorausgehenden Befehle die Pipeline verlassen haben. In der Architektur des PowerPc™ von IBM/Motorola wird auf die Sonderregister in einer eigenen Ausführungseinheit durch spezielle Befehle zugegriffen [2; S.1-10]. Bei der einfachen DLX-Maschine hat sich gezeigt, daß hier eine strenge Trennung zwischen den Zugriffen auf die Sonderregister und den Speicher nicht notwendig ist. Aus diesem Grund werden den Sonderregister in dieser Implementierung feste Adressen am oberen Ende des Speicherraums zugeordnet. Bei der Einführung von virtuellem Speicher und der dazu erforderlichen Adreßumsetzung ergibt sich für das Betriebssystem noch die Möglichkeit, durch Belegung der Seitentabellen zu entscheiden, ob der Prozeß eines Benutzers auf die Sonderregister zugreifen darf.

2.3 Mehrzyklusoperationen

Bisher kann die DLX nur Einzyklusbefehle wie arithmetisch-logische Befehle und Zweizyklusbefehle wie Lade- bzw. Speicherbefehle durchführen. Zusätzlich soll der Prozessor auch Multiplizier- und Dividierbefehle für ganze Zahlen bearbeiten können. Diese sind rechenintensiv und zählen so zu den Mehrzyklusoperationen (multicycle operations) [1; S.185 ff.]. Die arithmetisch-logische Recheneinheit ist zur Ausführung dieser Befehle ungeeignet. Deswegen wird die Multiplizier-Dividier-Einheit (Multiply-Divide-Unit) in die DLX aufgenommen. In der Pipeline befindet sich die neue Einheit parallel zur Ausführung und Adreßberechnung sowie dem Speicherzugriff der bisher implementierten Befehle.

Abbildung 2-2 zeigt in einer Skizze die Ausführungseinheiten der jetzt entstandenen erweiterten DLX-Pipeline.

Es liegen nun zwei parallele Ausführungspfade vor. Hierdurch erschwert sich die Einhaltung der Pflicht zum In-Reihe-Beenden von Befehlen [1; S.278 ff.]. Ein Divisionsbefehl wird zum Beispiel während mehrerer Takte innerhalb einer Ausführungsphase bearbeitet. Ein nachfolgender arithmetisch-logischer Befehl kann auf seinem Ausführungspfad leicht vor dem Divisionsbefehl vollständig bearbeitet sein und die Pipeline verlassen. Die Befehle werden so nicht in der Reihenfolge beendet, in der sie im Programmfluß auftreten. Eine präzise Ausnahmebehandlung ist damit nicht mehr möglich oder wird zumindest deutlich erschwert. Dies wird anschaulich, wenn man sich vorstellt, daß der arithmetisch-logische Befehl einen Arithmetikfehler erzeugt und eine Ausnahmebehandlung auslöst. Es ist dann nicht möglich, nur eine Adresse während dem Start der Ausnahmebehandlung zu retten, die gleichzeitig den nicht ausführbaren Befehl und den Befehl, mit dem das unterbrochene Programm später fortgesetzt werden soll, beschreibt. Am Ende der Pipeline ist es ohne zusätzliche Hardware auch nicht möglich zu entscheiden, in welcher Reihenfolge der Divisionsbefehl und der arithmetisch-logische Befehl im Programm vorkommen. Dies ist aber notwendig, um die Resultate in der richtigen Folge in den allgemeinen Registersatz zu schreiben und so die Bearbeitung der Befehle abzuschließen.

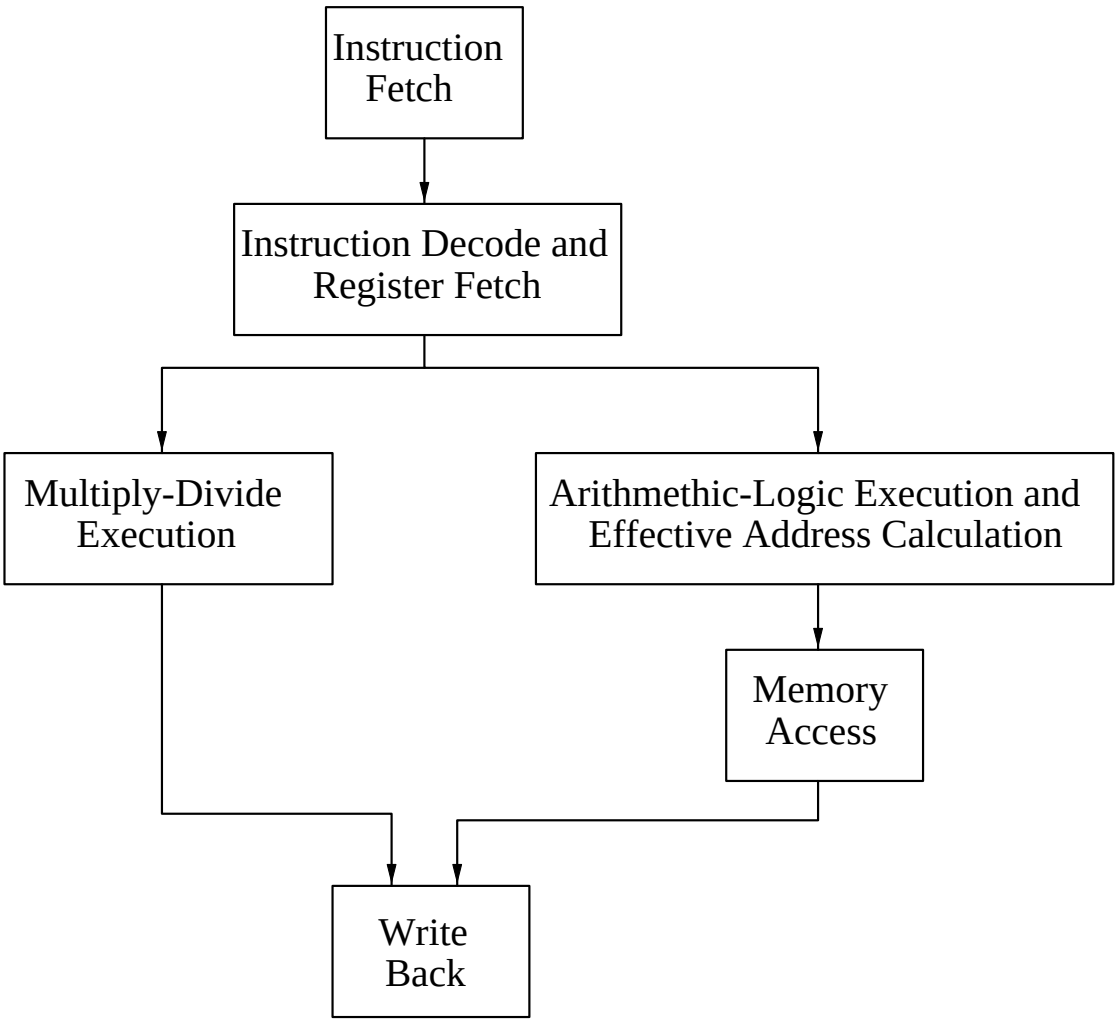

Abbildung 2-2: Erweiterte DLX-Pipeline

Zusätzlich tritt noch das Problem auf, daß der arithmetisch-logische Befehl bei einem Außer-Reihe-Beenden sein Resultat schon in den Registersatz geschrieben hat. Damit wurde der Maschinenstatus so verändert, daß während oder nach einer Ausnahmebehandlung nicht der Zustand vorliegt, der bei einem In-Reihe-Beenden der Befehle erwartet wird.

Die Pipeline muß also in der Lage sein, die Befehle in der Reihenfolge zu beenden, in der sie auch im Programm erscheinen.

2.4 In-Reihe-Beenden von Befehlen bei zwei Ausführungspfaden

Hierzu wurde im Entwicklungsverlauf dieser Arbeit zunächst an die Einführung eines Zählers in der Decodierungsphase gedacht. Dieser kann die Befehle zählen, die an die Ausführungseinheiten übergeben werden. Wird zusätzlich auch der Wert des Zählers an die Ausführungseinheiten weitergeleitet und dieser durch die Pipeline-Register bis zur Rückschreibephase durchgereicht, so kann dort bei ausreichender Breite des Zählers und Beachtung des Überlaufs die ursprüngliche Reihenfolge der Befehle wieder erkannt werden. Die Pflicht zum In-Reihe-Beenden von Befehlen ist so erfüllbar.

In der verwendeten Literatur finden sich zwei weitere Lösungsansätze: Ein Reorder-Buffer und eine Completion-Queue.

2.4.1 Reorder-Buffer

Der Reorder-Buffer [1; S.309 ff.] ist ein Ringpuffer. Jeder Befehl wird während seinem Start an einen Eintrag im Reorder-Buffer weitergeleitet. Diese Position bleibt dem Befehl dann bis zum Verlassen der Pipeline fest zugeordnet. Ein Eintrag besteht aus je einem Feld für die Befehlsparameter und die errechneten Daten für das Zielregister sowie einem Valid-Flag. Ein Zeiger in den Reorder-Buffer bestimmt den nächsten Befehl, der in der Programmreihenfolge zu beenden ist. Während der Prozessor eine Instruktion an eine Ausführungseinheit übergibt, schreibt er die Befehlsparameter in den vom Zeiger aus betrachtet nächsten freien Eintrag des Reorder-Buffer. Ein Befehl kann nur dann gestartet werden, wenn ein freier Eintrag im Reorder-Buffer zur Verfügung steht. Gleichzeitig erhält die Ausführungseinheit einen Zeiger auf den ihr jetzt zugeordneten Eintrag. Innerhalb des Ringpuffers haben die Befehle nun die gleiche Reihenfolge wie im Programmfluß. Durch die feste Zuordnung einer Instruktion zu einem Eintrag kann die Ausführungseinheit ihre errechneten Daten für das Zielregister in das entsprechende Datenfeld des Reorder-Buffer schreiben.

Im Anhang befinden sich bildlichen Darstellungen von dem Befehlsfluß durch die Pipeline. Die Funktiosweise des Reorder-Buffer läßt sich daran am einfachsten nachvollziehen.

Der Prozessor darf einen Befehl nur dann beenden, wenn dessen Ausführung abgeschlossen ist und der Zeiger auf ihn deutet. Dabei wird das Zielregister mit den neuen Daten beschrieben und der Zeiger auf den folgenden Eintrag gesetzt.

Falls nun eine Ausnahmebehandlung gestartet wird, so ist bekannt, welcher Befehl als nächstes die Pipeline in Programmfolge verlassen würde. Dessen Adresse findet der Prozessor im zugeordneten Parameterfeld und rettet sie. Wird gleichzeitig noch der gesamte Reorder-Buffer gelöscht, so ist der Maschinenstatus nur bis zu dem zuletzt beendeten Befehl verändert und das unterbrochene Programm kann im Anschluß an die Ausnahmebehandlung nach der Pflicht zum In-Reihe-Beenden von Befehlen präzise fortgesetzt werden.

Der Reorder-Buffer ist dann leer, wenn der Zeiger auf eine ungültige Position verweist. In diesem Fall befindet sich der nächste auszuführende Befehl entweder im Pipeline-Register zwischen der Holphase und der Decodierungsphase oder noch nicht in der Pipeline.

Die Ausführungseinheiten schreiben ihr Resultat in das zugeordnete Datenfeld des Reorder-Buffer. Es kann nun vorkommen, daß sich die Ausführung eines zuerst gestarteten Befehls verzögert und die Bearbeitung einer späteren Instruktion schon abgeschlossen ist. Der fertig ausgeführte Befehl kann jetzt die Pipeline nicht verlassen, da der Zeiger auf den vorigen Befehl deutet und dessen Resultat noch nicht zur Verfügung steht. Die schon vorliegenden Resultate können jedoch bei einem Read-After-Write-Hazard an die neu zu startende Instruktion weitergeleitet werden. Bei mehreren Registerübereinstimmungen im Reorder-Buffer muß immer der Eintrag mit dem am spätesten gestarteten Befehl ausgewählt werden. Sind dessen Daten noch nicht verfügbar, fügt der Prozessor Wartezyklen in die Pipeline ein.

Wenn man zuläßt, daß je zwei Befehle gleichzeitig in der richtigen Reihenfolge in den Reorder-Buffer geschrieben und von dort beendet werden können, dann kann so auch ein superskalarer Prozessor gesteuert werden.
Abbildung 2-3 zeigt den Kontrollfluß für das Weiterleiten und Beenden von Befehlen.

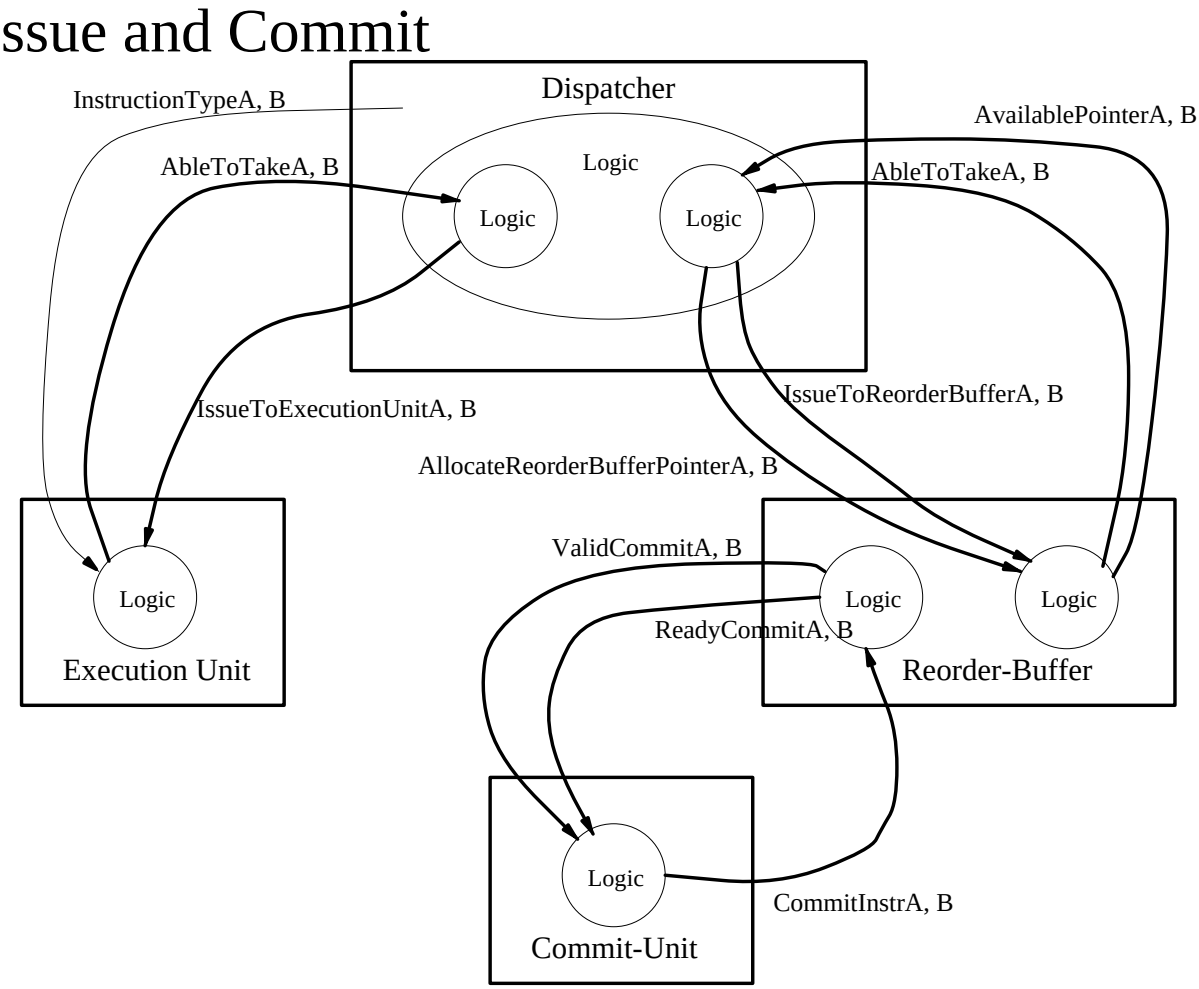

Abbildung 2-3: Kontrollfluß für das Weiterleiten und Beenden von Befehlen

2.4.2 Completion-Queue

Im PowerPc™ MPC603 ist zum In-Reihe-Beenden von Befehlen eine Completion-Queue [2; S.6-5 ff.] implementiert. Diese arbeitet nach dem Prinzip eines Schieberegisters. Während der Prozessor einen Befehl an eine Ausführungseinheit übergibt, schreibt er die Befehlsparameter in das Schieberegister. In dessen Einträgen gibt es kein Datenfeld. Der MPC verfügt über einen erweiterten Registersatz mit einem sogenannten Future-File aus Rename-Registern. Jedem gestarteten Befehl ist ein Rename-Register fest zugewiesen, in das die Ausführungseinheiten dann ihr Resultat schreiben. Die Befehlsparameter bewegen sich während der Ausführung zum Ausgang des Schieberegisters. Dort beendet der Prozessor fertig ausgeführte Befehle, indem er deren Resultat aus dem Rename-Register in den allgemeinen Registersatz schreibt. Das In-Reihe-Beenden ist so gewährleistet und eine Ausnahmebehandlung kann präzise vorgenommen werden.

Die getrennte Speicherung von Befehlsparametern und Daten nach dem Ansatz der Completion-Queue führt dazu, daß die Steuerlogik die Wechselwirkungen zwischen dem Schieberegister, den Ausführungseinheiten und dem Future-File kontrollieren muß. Der Reorder-Buffer enthält dagegen die Befehlsparameter und die Daten für die Zielregister. Die Entscheidung zwischen beiden Ansätzen entfiel zugunsten des kompakten Reorder-Buffer. Der Aufbau eines Ringpuffers ist zwar komplizierter als der eines Schieberegisters, es kann aber die notwendige Logik zum In-Reihe-Beenden der Befehle und Weiterleiten von vorliegenden Resultaten an nachfolgende Instruktionen in einer Implementierungseinheit aufgebaut werden.

2.5 Superskalare DLX-Implementierung

Die Beschreibungen in der Literatur zum In-Reihe-Beenden von Befehlen bei mehreren Ausführungspfaden haben zu einer superskalaren DLX-Implementierung motiviert [1; S.278 ff.].

In dieser Architektur ist es möglich, zwei Befehle in einem Takt aus dem Cache zu lesen und zu starten. Es wurde schon beschrieben, daß eine erweiterte Implementierung des Reorder-Buffer in der Lage ist zwei Befehle gleichzeitig in der Programmreihenfolge zu starten und zu beenden.

Der bisher gemeinsame Ausführungspfad für die arithmetisch-logischen Befehle und die Lade- bzw. Speicherbefehle wird in zwei getrennte Ausführungspfade aufgespalten. Durch die nun vorhandenen drei unabhängigen Ausführungseinheiten erhöht sich der Befehlsdurchsatz zusätzlich.

Abbildung 2-4 zeigt in einer Skizze den Datenpfad für die superskalare DLX-Pipeline.

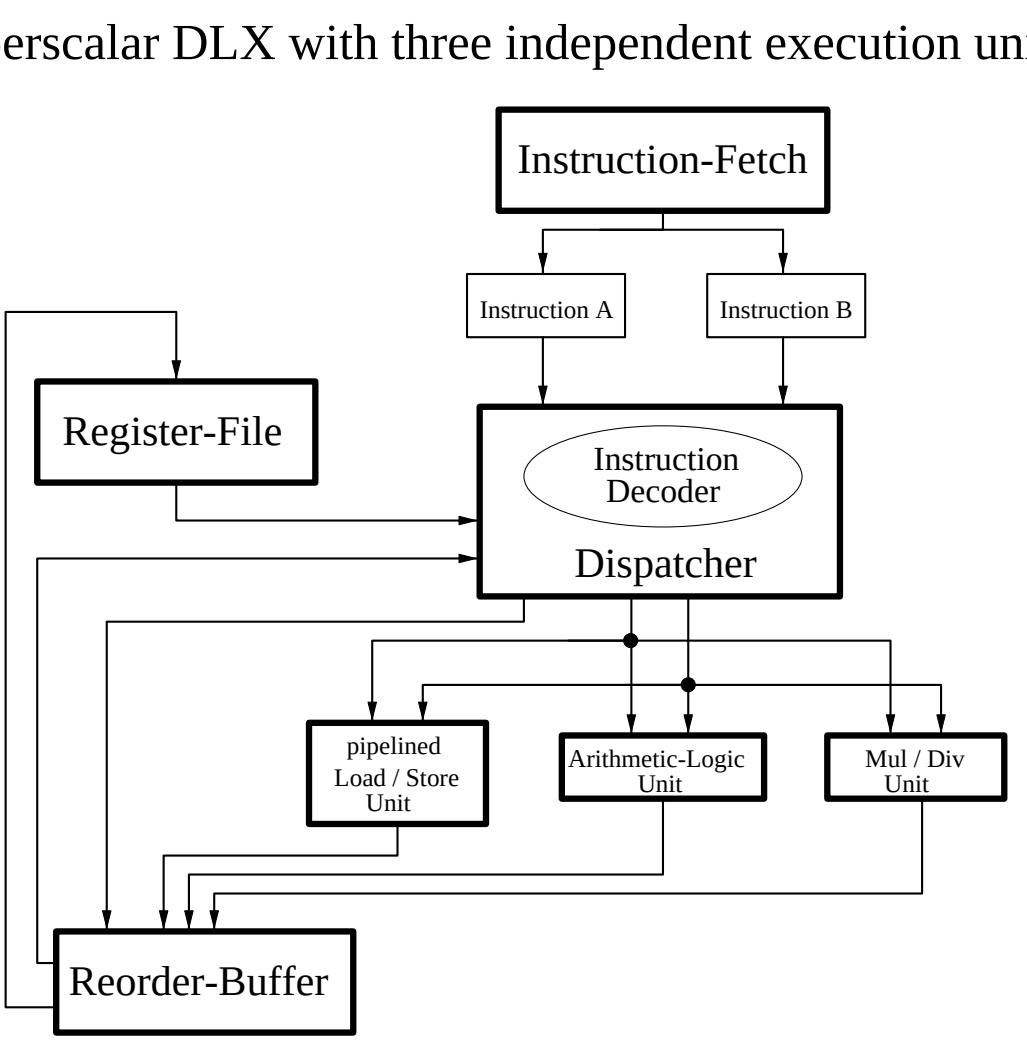

Abbildung 2-4: Superskalare DLX-Pipeline

2.6 Ausbauen der Pipeline-Register zwischen der Decodierungs- und Ausführungsphase

Der MPC603 verfügt über einen Reservierungsplatz (reservation station) am Eingang jeder Ausführungseinheit [2; S.6-2, S.6-10]. Ein Reservierungsplatz ist eine Erweiterung des Pipelineregisters zwischen der Decodierungs- und der Ausführungsphase. Er ermöglicht es, daß der Dispatcher einen Befehl auch dann an die entsprechende Ausführungseinheit weiterleiten kann, wenn die Daten für diesen Befehl noch nicht verfügbar sind. Durch diesen Puffer kann das Pipelineregister zwischen der Holphase und der Decodierungsphase frühzeitig entleert und so für nachfolgende Befehle bereitgestellt werden. Wenn die benötigten Daten in einem der folgenden Takten bereitstehen, werden diese von dem Reservierungsplatz übernommen und die Befehlsausführung im nächsten Takt gestartet.

In dieser Implementierung des DLX-Prozessor wird je ein Reservierungsplatz für einen 32-Bit Quelloperanden benutzt.

Abbildung 2-5 zeigt in einer Skizze den Datenpfad zum Laden der Operanden in einen Reservierungsplatz.

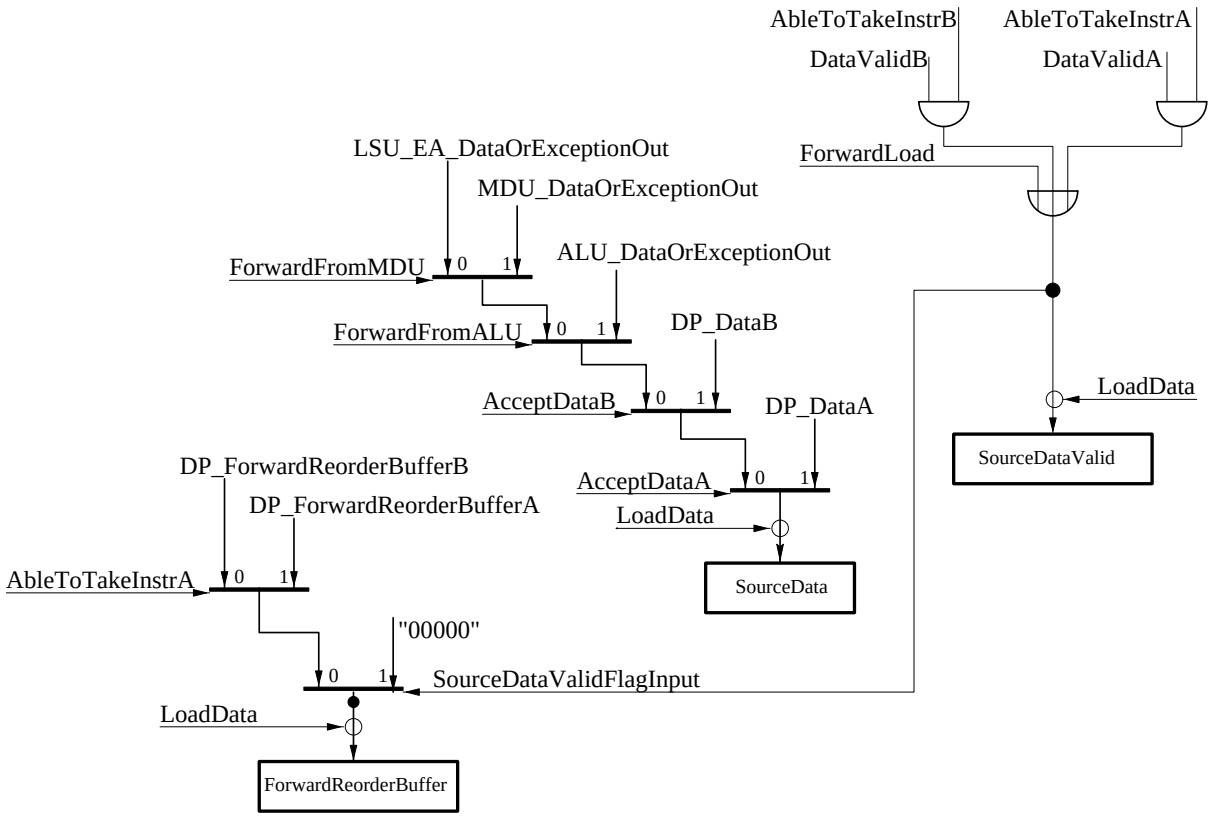

Abbildung 2-5: Datenpfad zum Laden der Operanden in einen Reservierungsplatz

2.7 Die Behandlung von Verzweigebefehlen

Bisher wurde noch nicht auf die besondere Bedeutung von Sprungbefehlen in pipeline-implementierten Maschinen eingegangen [1; S.161 ff.].

Zum Zeitpunkt der Ausführung einer Verzweigung können sich schon nachfolgende Befehle in der Pipeline befinden, die jetzt nicht mehr bearbeitet und deswegen gelöscht werden. In der DLX-Pipeline bedeutet dies, daß für jeden durchgeführten Sprung ein zusätzlicher Schritt benötigt wird, um den Befehl vom Sprungziel zu holen. In diesem Takt muß dann ein Wartezyklus in die Pipeline eingefügt werden. Dadurch verringert sich die Leistung des Prozessors, da Verzweigungen in Programmen häufig vorkommen [1; S.166 ff.].

Ein Lösungsansatz besteht hierzu in der Einführung von verzögerten Verzweigungen (delayed branch) [1; S.168 ff.]. Hierbei führt der Prozessor nach der Ausführung eines Sprungbefehls die sequentiell schon in die Pipeline eingelesenen Nachfolgebefehle des Sprungs noch aus. Diese werden also nicht gelöscht. Erst dann beginnt, für das Programm als Verzögerung sichtbar, die Ausführung der Befehle am Verzweigeziel. Bei dieser Technik ergeben sich im Programm sogenannte Verzweigeverzögerungsplätze (branch-delay slots) nach den Verzweigebefehlen, die der Compiler mit sinnvollen Befehlen belegen muß. Diese werden unabhängig von dem Sprung ausgeführt.

Die bisher festgelegte DLX-Architektur kann zwei Befehle pro Takt aus dem Cache bzw. dem Speicher holen. Falls nun der erste Befehle eine bedingte Verzweigung darstellt, dann führt der Prozessor bei Anwendung der Verzögerungstechnik den zweiten Befehl und die beiden nachfolgenden Befehle immer aus. Daraus ergeben sich drei Verzweigeverzögerungsplätze. Da es für einen Compiler schwierig oder nicht möglich ist diese Plätze sinnvoll zu belegen [1; S.172 ff.], muß eine andere Technik zur Reduzierung der Verzweigeverluste eingesetzt werden.

Hierzu bietet sich eine dynamische Hardware-Voraussage mit Hilfe eines Verzweigezielpuffers an.

2.7.1 Verzweigezielpuffer (branch-target buffer)

Der Verzweigezielpuffer [1; S.271 ff.] ist ein Cache, in dem jeder Eintrag aus zwei Adressen besteht. Die erste ist die Adresse von bekannten Verzweigebefehlen, die beim letzten Vorkommen im Programmfluß ausgeführt wurden. Während der Prozessor einen neuen Befehl aus dem Befehls-Cache oder dem Speicher holt, wird dessen Adresse, also der Wert des Befehlszählers, mit den Befehlsadressen im Verzweigezielpuffer verglichen. Wird hierbei ein Treffer festgestellt, so bedeutet dies, daß der gerade geholte Befehl eine Verzweigung darstellt, die beim letzten Auftreten im Programmfluß ausgeführt und in den Verzweigezielpuffer eingetragen wurde. Die zweite Adresse eines Eintrags im Verzweigezielpuffer stellt die Zieladresse des Sprunges dar. Diese wurde bei der letzten Ausführung zusammen mit der Befehlsadresse in den Cache geschrieben und kann bei einem Treffer direkt in den Befehlszähler kopiert werden. Im nächsten Takt setzt der Prozessor das Holen der Befehle unverzögert am Sprungziel fort.

Falls die so vorausgesagte Verzweigung in der Decodierungsphase als nicht ausgeführt erkannt wird, dann muß der Befehlszähler jetzt auf den der Verzweigung folgenden Befehl gesetzt werden, um so die Ausführung des Programms korrekt fortzusetzen. Bei Verwendung eines Ein-Bit-Voraussageschemas löscht der Verzweigezielpuffer gleichzeitig den Eintrag mit der falsch vorausgesagten Verzweigung.

In dieser Technik wird eine Verzweigung ausgeführt, bevor der Befehl decodiert ist. Dies bedeutet, daß die Befehlsadressen im Verzweigezielpuffer nur Adressen von Verzweigebefehlen darstellen dürfen. Ein neuer Eintrag wird also nur dann in den Verzweigezielpuffer geschrieben, wenn in der Decodierungsphase ein Verzweigebefehl erkannt wird und dieser nicht vorausgesagt wurde.

In einer superskalaren DLX-Architektur können zwei Befehle gleichzeitig in die Pipeline geholt werden. Für sie gibt es im Verzweigezielpuffer nur einen gemeinsamen Eintrag. Bei der Implementierung muß also mit zusätzlicher Hardware immer zwischen beiden Befehlen unterschieden werden.

2.8 Ungewisse Verzweigungen und Befehlsausführungen

Die Reservierungsplätze vor den Ausführungseinheiten ermöglichen es, daß Befehle auch dann die Decodierungsphase verlassen können, wenn deren Operanden wegen einer Datenabhängigkeit noch nicht zur Verfügung stehen. Bei Verzweigebefehlen kann ebenfalls eine noch nicht auflösbare Datenabhängigkeit bestehen. Der DLX-Prozessor erhält nun eine weitere Ausführungseinheit zum Auflösen von Verzweigungen (Branch-Resolve-Unit). Tritt eine nicht auflösbare Verzweigebedingung in der Decodierungsphase auf, so wird der Verzweigebefehl an die Branch-Resolve-Unit weitergeleitet. Wenn durch den Verzweigezielpuffer eine Verzweigung vorausgesagt wurde, setzt der Prozessor die Ausführung am Verzweigeziel fort, sonst sequentiell bei dem nächsten Befehl. In beiden Fällen ist die weitere Bearbeitung der Instruktionen jedoch ungewiß (speculative), da der ausgeführte Pfad noch nicht als korrekt erkannt ist [1; S.308 ff.]. Während eine Verzweigung in der Branch-Resolve-Unit auf ihre Daten zur Auflösung der Bedingung wartet, erhalten alle neu gestarteten Befehle ein Speculative-Flag. Dieses wird als zusätzlicher Befehlsparameter an die Ausführungseinheit und den Eintrag im Reorder-Buffer übergeben. Die Branch-Resolve-Unit übernimmt, ähnlich wie ein Reservierungsplatz, die erwarteten Daten, sobald diese bereitstehen. Falls der Prozessor nun erkennt, daß der falsche Pfad ausgeführt wurde, löscht er alle Befehle mit gesetztem Speculative-Flag aus den Ausführungseinheiten und dem Reorder-Buffer und führt die Bearbeitung des Programms mit dem richtigen Pfad fort. War der ungewisse Pfad korrekt, so müssen nur die Speculative-Flags gelöscht werden.

Abbildung 2-6 zeigt den Daten- und Kontrollfluß in der Arithmetic-Logic-Unit. Dabei ist auch die notwendige Logik zur Behandlung ungewisser Befehlsausführungen dargestellt.

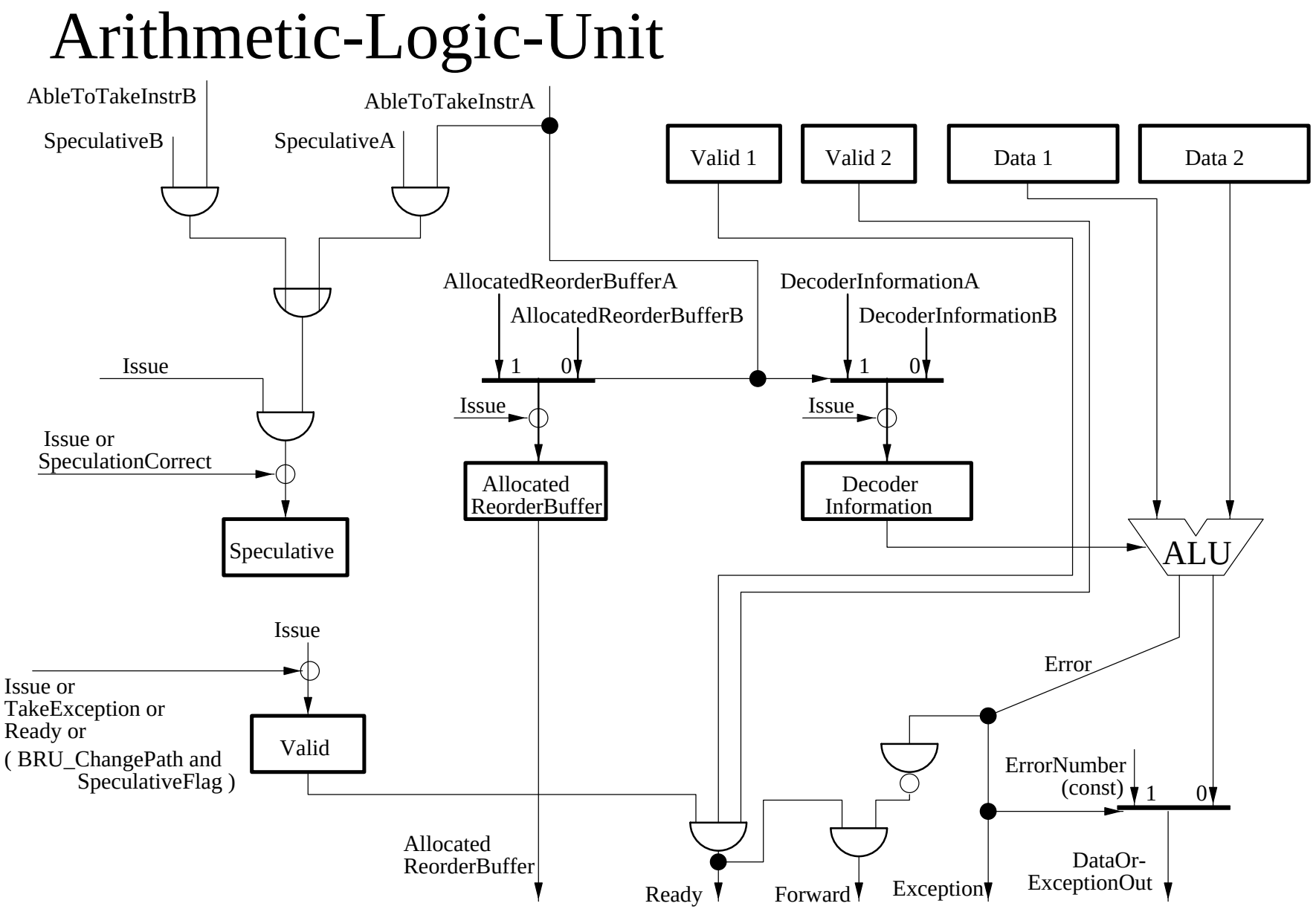

Abbildung 2-6: Daten- und Kontrollfluß in der Arithmetic-Logic-Unit

2.9 Befehls-Cache und Daten-Cache

Der Befehls-Cache und der Daten-Cache werden sehr einfach aufgebaut. Beide Caches sind direkt abbildend und die Blockgröße beträgt 64 Bit. Dies entspricht zwei Befehlen, die ja gleichzeitig in die Pipeline geholt werden können. Da in dieser superskalaren DLX-Architektur auch der externe Datenbus 64 Bit breit sein wird, empfiehlt sich für beide Caches diese Blockgröße.

Schreibzugriffe auf den Daten-Cache werden an den Speicher weitergeleitet (write-through). Wird die zu schreibende Speicheradresse nicht im Cache gefunden, dann werden die Daten nur in den Hauptspeicher geschrieben (no-write-allocate).

2.10 Schreibpuffer (write buffer)

Muß der Prozessor auf die Fertigstellung eines Schreibvorgangs warten, ist es notwendig, daß Wartezyklen in die Pipeline eingefügt werden. Deren Zahl kann man mit dem Einsatz eines Schreibpuffers [1; S.411 ff.] reduzieren, da der Prozessor weiter arbeiten kann, während der Speicher aktualisiert wird. Werden Daten vom Speicher in den Daten-Cache gelesen, so muß sichergestellt werden, daß es sich hierbei nicht um veraltete Daten handelt, die der Schreibpuffer noch nicht auf den aktuellen Stand gebracht hat. Dies kann zum Beispiel dadurch geschehen, daß die Daten auf dem Weg vom Speicher in den Cache den Schreibpuffer durchlaufen und dort mit den möglicherweise neuen Bytes verschmolzen (merged) werden. Bei dieser Art der Implementierung liest der Cache die Daten durch den Schreibpuffer.

Die erste Stufe des Schreibpuffer hat eine besondere Funktion. Die Load-Store-Unit schreibt während der Ausführung die Daten immer in diese Eingangsstufe. Durch die ungewisse Bearbeitung von Befehlen und das Starten von Ausnahmebehandlungen ist es möglich, daß der ausgeführte Schreibzugriff nicht im Programmfluß liegt und deswegen nicht beendet werden darf. Die Daten der ersten Stufe dürfen also in diesem Fall nicht in den Cache oder den Speicher geschrieben werden. Die Pflicht zum In-Reihe-Beenden von Befehlen realisiert der Schreibpuffer durch ein Commit-Flag in der Eingangsstufe. Erst wenn dieses durch das Beenden des Schreibbefehls gesetzt ist, dürfen die Daten an die weiteren Stufen und damit an den Speicher übergeben werden. Wird eine Ausnahmebehandlung gestartet während das Commit-Flag nicht gesetzt ist, löscht der Schreibpuffer die Eingangsstufe. Dies ist vergleichbar mit dem Löschen von ungewiß ausgeführten Befehlen auf einem falschen Programmpfad.

Abbildung 2-7 zeigt den Datenpfad des Write-Buffer.

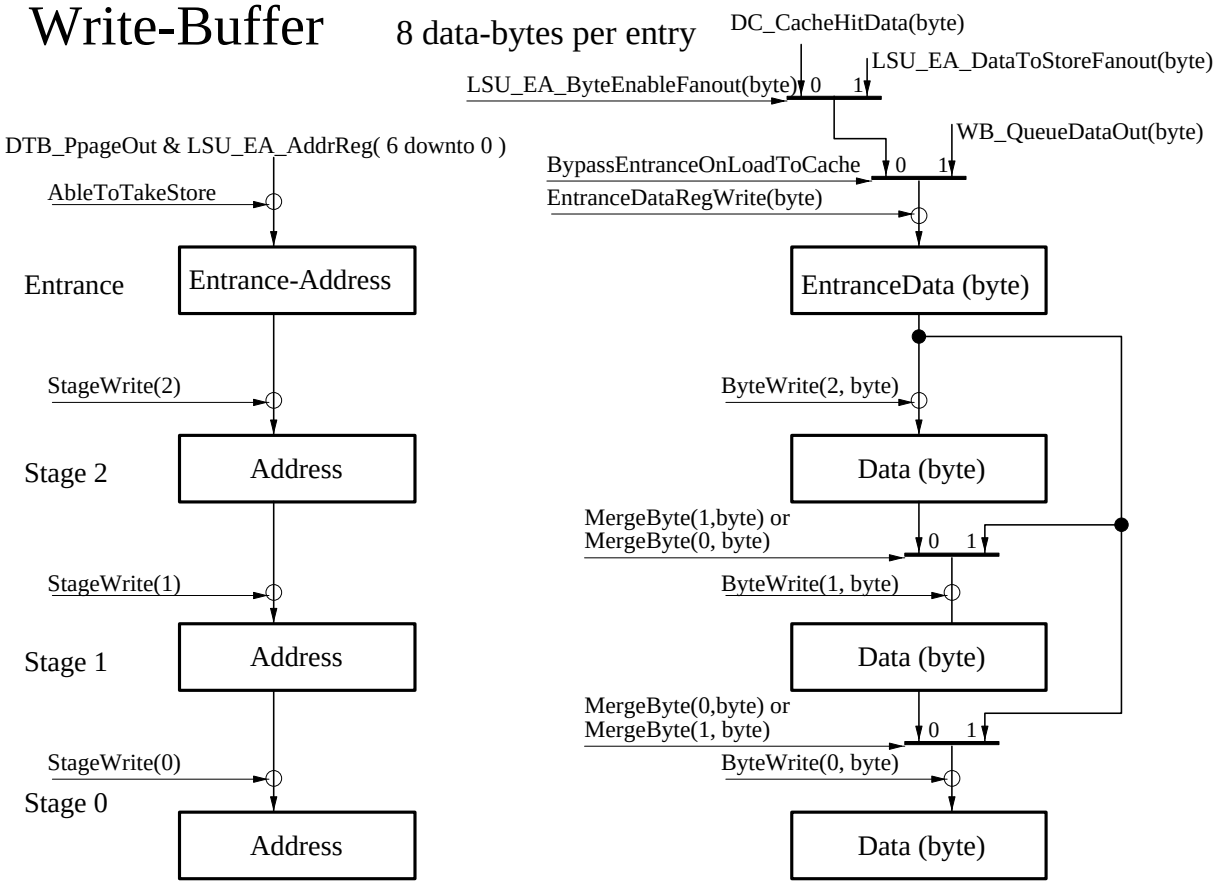

Abbildung 2-7: Datenpfad des Write-Buffer

2.11 Unterstützung von Mehrprogrammbetrieb und virtuellem Speicher

Zur Unterstützung von Mehrprogrammbetrieb wird zunächst ein weiteres Sonderregister implementiert. Dieses enthält die Prozeßidentifikationsnummer des laufenden Prozeß.

Die Einführung von virtuellem Speicher [1; S.439 ff.] erfordert eine Adreßumsetzung der effektiven Adressen des laufenden Prozeß in reale Adressen des Hauptspeichers. Hierzu wird je ein Adreß-umsetzungspuffer [1; S.445 ff.] parallel zum Befehls- und Daten-Cache aufgebaut.

In diesem Entwurf eines Modellprozessors beträgt die Seitengröße 128 Bytes und die Caches haben jeweils eine Kapazität von 64 Bytes. Diese Werte entsprechen nicht denen von realen Prozessoren. Sie ermöglichen es aber schon kleinen Testprogrammen die Funktionen dieser DLX-Implementierung zu demonstrieren.

Das folgende Bild zeigt den Aufbau einer effektiven Adresse und den Ablauf der Adreßumsetzung. Die virtuelle Adressierung des Cache liefert die oberen Bits (tag) der realen Speicheradresse. Gleichzeitig wandelt der Adreßumsetzungspuffer die virtuelle Seitennummer in eine physikalische Seitennummer um. Sind die daraus resultierenden oberen Bits der realen Speicheradresse identisch mit dem Tag, so liegt ein Cache-Hit vor und die Daten sind gültig.

Wenn der Adreßumsetzungspuffer nicht in der Lage ist die virtuelle Seitennummer zu übersetzten, dann fügt er die Anmeldung für eine Ausnahmebehandlung in die Pipeline ein. Das Starten der Routine zur Ausnahmebehandlung erfolgt nach der Pflicht zum In-Reihe-Beenden von Befehlen. Das Betriebssystem muß dann den fehlenden Eintrag aus der Seitentabelle lesen und in den Adreßumsetzungspuffer laden. Falls sich die angesprochene Seite nicht im Hauptspeicher befindet, ist der Aufruf einer Prozedur zum Einlesen der Daten von der Festplatte notwendig.

Abbildung 2-8 skizziert die Adreßumsetzung und den Zugriff auf den Cache.

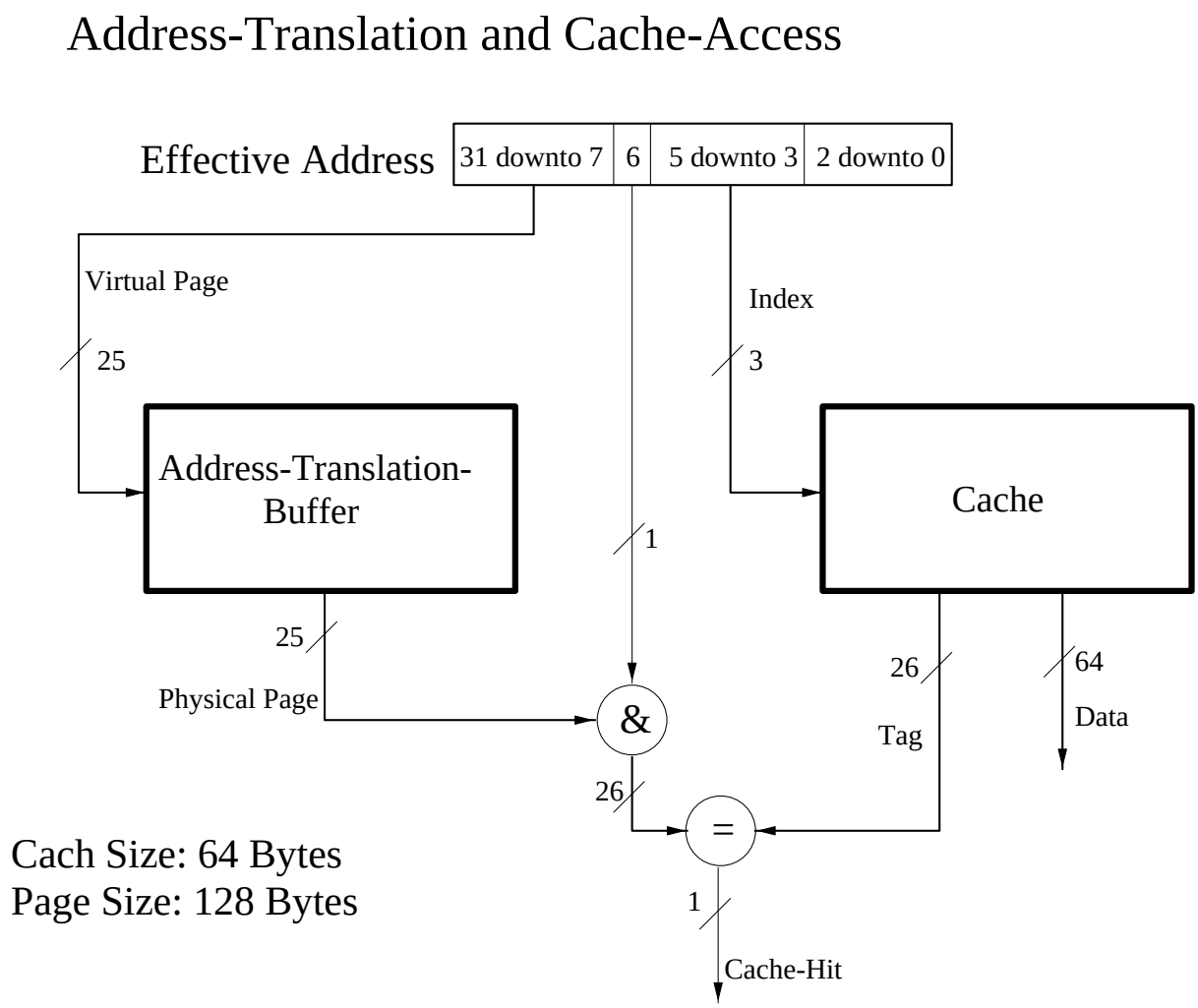

Abbildung 2-8: Adreßumsetzung und Zugriff auf den Cache

Der Cache ist hier nur halb so groß wie eine Seite des Speichers. Daraus ergibt sich, daß ein Bit aus dem Offset der angesprochenen Seite nicht zur Adressierung des Cache dient und deswegen zusammen mit der physikalischen Seitennummer zum Adreßvergleich benutzt werden muß.
In realen Prozessoren beträgt die Größe des Cache gewöhnlich ein ganzzahliges Vielfaches der Seitengröße. Der Cache muß dann nur die physikalische Seitennummer als Tag speichern.
Da bei der parallelen Arbeitsweise von Adreßumsetzungspuffer und Cache nur der Offset der Seite zur Addressierung des Cache benutzt werden darf, kann ein einfach assoziativer Cache nicht größer sein als eine Seite der Adreßumsetzung. Wenn die Kapazität erhöht werden soll, dann muß der Aufbau des Cache mehrfach assoziativ organisiert sein.

2.12 Protokoll für den externen Bus

Der DLX-Prozessor erhält ein einfaches synchrones Bus-Interface. Externe Systemregister müssen auf feste Adressen im Speicherraum abgebildet werden.

Abbildung 2-9 zeigt den Datenpfad der Bus-Interface-Unit.

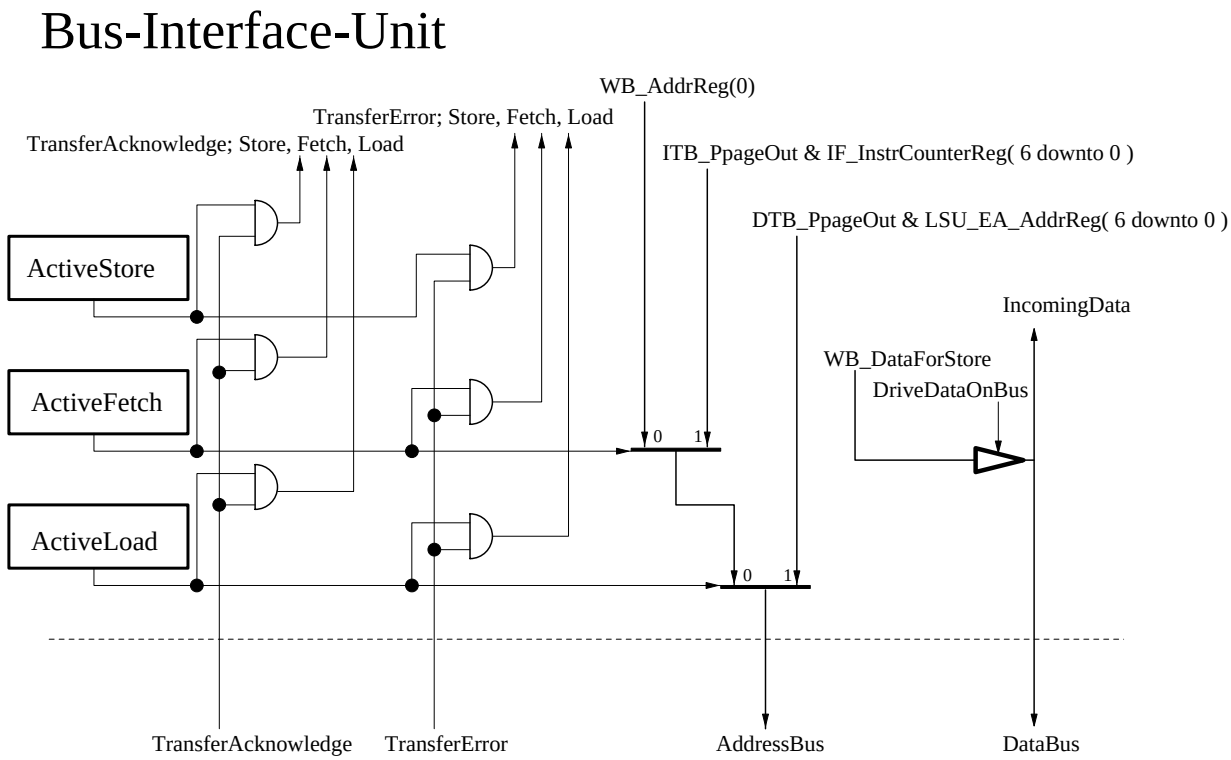

Abbildung 2-9: Datenpfad der Bus-Interface-Unit

Abbildung 2-10 zeigt das Blockdiagramm des implementierten DLX-Prozessors.

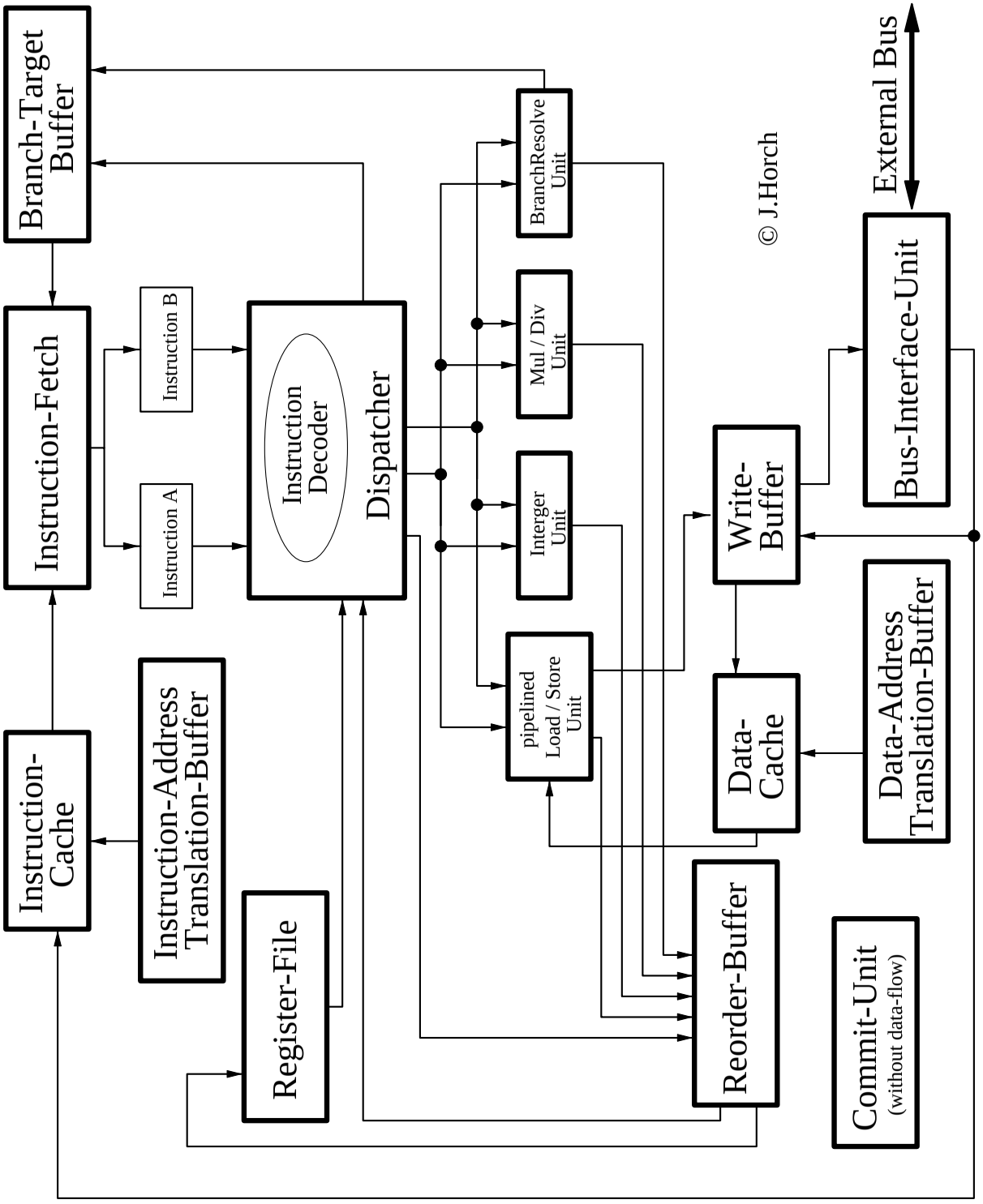

Abbildung 2-10: Blockdiagramm der DLX

3 Die entstandene DLX-Implementierung im Überblick

In diesem Kapitel werden die Besonderheiten dieser DLX-Implementierung beschrieben. Die Grundarchitektur und der Befehlssatz sind im Buch von Hennessy und Patterson ausführlich dargestellt und sollten dort nachgelesen werden. Zusammen mit den folgenden Ausführungen ist die Programmierung der DLX möglich.

In der hier vorliegenden Arbeit wird keine Gleitpunktarithmetik unterstützt. Die folgenden Abbildungen zeigen alle implementierten Befehle.

lb	lh	lw	lbu	lhu	sb	sh	sw

Abbildung 3-1: Befehle für den Datentransport

add	addi	addu	addui	and	andi	div	divu
lhi	mult	multu	or	ori	seq	seqi	sequ
sequi	sge	sgei	sgeu	sgeui	sgt	sgti	sgtu
sgtui	sle	sll	slei	sleu	sleui	slli	slt
slti	sltu	sltui	sne	snei	sneu	sneui	sra
srai	srl	srli	sub	subi	subu	subui	xor
xori							*nop*

Abbildung 3-2: Befehle für arithmetisch-logische Operationen

beqz	bnez	j	jal	jalr	jr	rfe	trap

Abbildung 3-3: Befehle für die Programmsteuerung

3.1 Spezielle Merkmale

- Superskalare, pipeline-implementierte Maschine
  - Bis zu 2 Befehle können in einem Takt aus dem Befehls-Cache geholt und gestartet werden.
  - Bis zu 3 Befehle können in einem Takt abgeschlossen werden. Diese Zahl ergibt sich aus einer Sprunganweisung im vorderen Teil der Pipeline und zwei Datentransportbefehlen bzw. arithmetisch-logischen Befehlen am Ende der Pipeline.
  - Ein Reorder-Buffer mit 5 Einträgen ermöglicht das In-Reihe-Beenden von Befehlen.
  - Bis zu 5 Befehle können gleichzeitig in Ausführung sein.
  - Ein Branch-Target Buffer mit 4 Einträgen und einem Ein-Bit-Voraussageschema erhöht den Befehlsdurchsatz der Pipeline.
  - Eine präzise Ausnahmebehandlung (precise exception processing) wird durch den Reorder-Buffer unterstützt.
- Vier unabhängige Ausführungseinheiten mit Reservierungsplatz (reservation station) für die Quelloperanden
  - Branch-Resolve-Unit
  - Arithmetic-Logic-Unit
  - Multiply-Divide-Unit
  - Load-Store-Unit
- Schreibpuffer mit 4 Stufen zu je 64 Bit
  - Die Eingangsstufe dient auch zum Puffern der Daten von Schreibbefehlen, die schon ausgeführt sind, aber die Pipeline noch nicht verlassen haben.
- 64 Byte Befehls-Cache, direkt abbildend
- 64 Byte Daten-Cache, direkt abbildend, Schreibstrategien: write-through, no-write-allocate
- Befehls-Adreßumsetzungspuffer mit 4 Einträgen, direkt abbildend, 128 Byte Seitengröße
- Daten-Adreßumsetzungspuffer mit 4 Einträgen, direkt abbildend, 128 Byte Seitengröße

Anmerkungen:

- Bei realen Prozessoren beträgt die Größe eines Caches einige Kilobyte (16kB im PowerPc™ MPC603, 8kB im Alpha AXP 21064). Die deutlich kleineren Caches in dieser Implementierung ermöglichen es schon bei kurzen Testprogrammen Kapazitätsfehlzugriffe und Kollisionsfehlzugriffe zu beobachten. Außerdem verringert sich die Simulationszeit.
- Bei realen Prozessoren beträgt die Seitengröße einige Kilobyte (4kB in der PowerPc™ Architektur, 8kB im Alpha AXP 21064). Die deutlich kleineren Seiten in dieser Implementierung erlauben es schon kurzen Testprogrammen Seitenbegrenzungen zu überschreiten und so eine Adreßumsetzung-Ausnahmebehandlung auszulösen. Zusätzlich kann der gesamte Adreßumsetzungspuffer schnell gefüllt werden.

3.2 Anmerkungen zum Befehlssatz

3.2.1 Ganzzahl-Multiplikation: MULT- und MULTU-Befehl

- Es werden vier Taktzyklen zur Ausführung benötigt.
- Die 32 Bit breiten Quelloperanden werden zunächst zu einem 64 Bit breiten Produkt multipliziert. Wenn ein Überlauf auftritt, also das Resultat nicht in das 32 Bit breite Zielregister paßt, wird eine Multiply-Divide-Overflow Ausnahmebehandlung festgestellt. Im anderen Fall repräsentieren die unteren 32 Bit des berechneten Produkts das erwartete Resultat.

3.2.2 Ganzzahl-Division: DIV- und DIVU-Befehl

- Es werden sechzehn Taktzyklen zur Ausführung benötigt.
- Der 32 Bit breite Dividend wird durch den 32 Bit breiten Divisor dividiert. Wenn der Divisor gleich Null ist, wird eine Divide-By-Zero Ausnahmebehandlung erkannt. Falls ein Überlauf auftritt (vorzeichenrichtige Division: 0x80000000 / 0xFFFFFFFF), wird eine Multiply-Divide-Overflow Ausnahmebehandlung festgestellt.

3.2.3 RFE-Befehl (return from exception)

- Es wird kein Parameter benötigt.
- Der RFE-Befehl wird erst dann ausgeführt, wenn der Reorder-Buffer leer ist. Dies wird wegen der Pflicht zum In-Reihe-Beenden von Befehlen notwendig. Der Befehl kann den Maschinenstatus sofort ändern und darf deswegen nicht ungewiß ausgeführt werden.
- Die oberen 30 Bit des Return-From-Exception Register bestimmen die Adresse, die in das Befehlszähler-Register geladen wird.
- Das Bit 0 des Return-From-Exception Register enthält den Wert, der in das Interrupt-Enable-Flag geladen wird.

TRAP-Befehle

- Eine Gruppe von speziellen Befehlen ist als 'TRAP #number' implementiert.
- Der TRAP-Befehl wird erst dann ausgeführt, wenn der Reorder-Buffer leer ist. Dies wird wegen der Pflicht zum In-Reihe-Beenden von Befehlen notwendig. Der Befehl kann den Maschinenstatus sofort ändern und darf deswegen nicht ungewiß ausgeführt werden.
- Es wird ein Parameter benötigt, der die Aktion des Befehls bestimmt.

Folgende TRAP-Anweisungen sind implementiert:

3.2.4 TRAP 0x000

Der Prozessor wird angehalten.

3.2.5 TRAP 0x001 - TRAP 0x100

Diese Befehle führen in diesem Entwurf zu keiner Aktion. Andere DLX-Implementierungen benutzen TRAP 0x001 bis TRAP 0x100 zum Aufruf von Systemprogrammen.

3.2.6 TRAP 0x101

Aufruf eines Betriebssystemdienstes in einem Nutzerprogramm. Es wird die System-Call Ausnahmebehandlungsroutine aufgerufen.

3.2.7 TRAP 0x102

Keine Aktion. Es werden nur solange Wartezyklen in die Pipeline eingeschoben, bis der Reorder-Buffer leer ist. Dies ist mit dem 'sync' Befehl des PowerPc™ Prozessor [2; S.6-12, und 3; S.9-179] vergleichbar.

Die Notwendigkeit für einen solchen Befehl entsteht dort, wo schon durch eine Befehlsausführung der Status des Rechnersystems unwiderrufbar verändert wird. Es gibt zum Beispiel E/A-Bausteine, deren Leseregister nach einem Lesevorgang mit einem neuen Wert geladen werden. Ist einem solchen Leseregister nun eine feste Adresse im Speicherraum zugeordnet, so kann die DLX mit dem Lade-Befehl Werte vom E/A-Baustein in die allgemeinen Register holen. Bei der Ausführung des Lade-Befehls werden die Daten zunächst in den Reorder-Buffer kopiert, bevor sie in einem der darauffolgenden Takte in das Zielregister übernommen werden. Es kann jedoch vorkommen, daß ein vorausgegangener Befehl, der sich noch im Reorder-Buffer befindet, eine Ausnahmebehandlung auslöst und die Daten nicht vom Reorder-Buffer in den Registersatz gelangen. Wenn der Lade-Befehl nach der Ausnahmebehandlung neu gestartet wird, befindet sich ein neuer Wert im Leseregister des E/A-Bausteins. Der Wert, der ursprünglich gelesen werden sollte, ist verloren.

Diese Fehlfunktion kann verhindert werden, indem der externe Interrupt gesperrt und dem Lade-Befehl ein 'TRAP 0x102' vorangestellt wird. Die Ausführung kann so nicht unterbrochen werden.

3.2.8 TRAP 0x103

Der Befehl nach dem TRAP wird aus dem Pipelineregister zwischen Holphase und Decodierungsphase gelöscht und neu vom Befehls-Cache angefordert. Dies ist mit dem 'isync' Befehl des PowerPc™ Prozessor [2; S.6-12, 3; S.9-76] vergleichbar.

Ein solcher 'Refetch' ist dann notwendig, wenn durch einen vorausgehenden Befehl gezielt der Befehls-Cache oder der Befehls-Adreßumsetzungspuffer verändert wurde. Der Programmierer erwartet, daß solche Änderungen schon im nächsten Befehl wirksam werden. Wenn nun eine Architektur mit einem Befehl-Prefetch-Buffer vorliegt, müssen alle Befehle im Puffer neu gelesen werden. Aus diesem Grund muß einem Befehl, der den Befehls-Cache oder der Befehls-Adreßumsetzungspuffer verändert, ein 'TRAP 0x103' nachgestellt werden.

Bei der DLX könnte eine Fehlfunktion auch durch einen nachfolgenden NOP-Befehl vermieden werden. Bei Architekturen, die in verschiedenen Implementierungen über verschieden lange Befehl-Prefetch-Buffer verfügen, ist dies nicht so leicht möglich.

3.2.9 TRAP 0x104

Keine Aktion. Es werden nur solange Wartezyklen in die Pipeline eingeschoben, bis der Schreibpuffer leer ist. Dieser Befehl ist für einen Prozeßwechsel wegen der Behandlung von möglichen Speicherschreibfehler notwendig. Siehe hierzu auch die Beschreibung der Ausnahmebehandlung von Speicherschreibfehlern.

3.2.10 TRAP 0x105

Der Schreibpuffer wird gelöscht. Dieser Befehl kann angewendet werden, wenn ein Speicherschreibfehler aufgetreten ist und das Betriebssystem daraufhin den verursachenden Prozeß beendet. Siehe hierzu auch die Beschreibung der Ausnahmebehandlung von Speicherschreibfehlern.

3.2.11 TRAP 0x106

Der Befehls-Adreßumsetzungspuffer wird gelöscht. Dies ist notwendig, wenn das Betriebssystem eine Speicherseite aus dem Hauptspeicher löscht.

3.2.12 TRAP 0x107

Der Daten-Adreßumsetzungspuffer wird gelöscht. Dies ist notwendig, wenn das Betriebssystem eine Speicherseite aus dem Hauptspeicher löscht.

3.2.13 TRAP 0x108

Der Befehls-Cache wird gelöscht. Dies ist notwendig, wenn das Betriebssystem eine Speicherseite aus dem Hauptspeicher löscht.

3.2.14 TRAP 0x109

Der Daten-Cache wird gelöscht. Dies ist notwendig, wenn das Betriebssystem eine Speicherseite aus dem Hauptspeicher löscht.

3.2.15 TRAP 0x10A

Der Branch-Target-Buffer wird gelöscht. Dieser Befehl ist nur zur Vollständigkeit implementiert. Der Prozessor löscht den Branch-Target-Buffer bei einem Kontextwechsel der Adreßumsetzung selbständig. Dies ist erforderlich, weil die Adressen der Branch-Befehle im Puffer effektive Adressen sind und deswegen nach einem Kontextwechsel ihre Gültigkeit verlieren.

Alle anderen TRAPs haben keine besondere Bedeutung. Sie werden wie 'TRAP 0x102' behandelt, das heißt, daß sie aus der Pipeline verschwinden sobald der Reorder-Buffer leer ist.

3.3 Ausnahmesituationen

In Ausnahmesituationen muß die DLX eine Routine zur Ausnahmebehandlung aufrufen. Dabei werden in einem Takt folgende Aktionen durchgeführt:

- Die Wort-Adresse, an der nach der Ausnahmebehandlung das unterbrochene Programm fortgesetzt werden soll, wird im Return-From-Exception Register gerettet.
- Der Status des Interrupt-Enable-Flag wird ebenfalls im Return-From-Exception Register gerettet.
- Der Befehlszähler wird mit der Adresse der Routine zur Ausnahmebehandlung geladen. Diese ergibt sich aus der aufgetretenen Ausnahmesituation.
- Das Interrupt-Enable-Flag wird gelöscht. Hierdurch wechselt der Prozessor in den Kernel-Mode, in dem die automatische Adreßumsetzung nicht aktiv ist

Falls in der Pipeline durch einen Befehl eine Ausnahmesituation verursacht wird, dann darf die daraus resultierende Ausnahmebehandlung erst gestartet werden, wenn alle vorangegangenen Befehl beendet sind. Dieses Vorgehen ist für eine präzise Ausnahmebehandlung notwendig und ergibt sich aus der Pflicht zum In-Reihe-Beenden von Befehlen.

Wird eine Ausnahmesituation für einen Befehl erkannt, dann wird diese in das Pipeline-Register des Befehls eingetragen und damit bis an das Ende der Pipeline durchgereicht. Von dort startet der Prozessor die präzise Ausnahmebehandlung.

Es folgt eine Zusammenstellung aller Ausnahmesituationen zusammen mit der physikalischen Adresse der Behandlungsroutinen.

3.3.1 Reset (Adresse 0x00000000)

Das externe Reset-Signal setzt den Befehlszähler auf Null, löscht alle Valid-Flags und initialisiert einen Zeiger auf einen Eintrag im Reorder-Buffer.

3.3.2 Speicherschreibfehler (Transfer-Error-Store) (Adresse 0x00000100)

Das externe Transfer-Error-Signal wurde während einem Schreibvorgang in den Speicher aktiv. Die Daten werden von der letzten Stufe des Schreibpuffers in den Speicher geschrieben. Diese ist nicht Teil der Pipeline, wodurch ein Speicherschreibfehler zu einer nicht präzisen Ausnahmebehandlung führt.

Das Betriebssystem muß deswegen den verursachenden Prozeß beenden. Um zu wissen welcher Prozeß den verbotenen Zugriff ausgelöst hat, muß das Betriebssystem vor einem Prozeßwechsel den Schreibpuffer leer laufen lassen. Hierdurch kann ein aufgetretener Speicherschreibfehler nur von dem aktuellen Prozeß erzeugt worden sein. Der Schreibpuffer muß bei einem Speicherschreibfehler gelöscht werden, da alle weiteren Einträge ebenfalls zu dem jetzt beendeten Prozeß gehören.

3.3.3 Befehls-Speicherlesefehler (Transfer-Error-Fetch) (Adresse 0x00000200)

Das externe Transfer-Error-Signal wurde während einem Befehls-Lesevorgang aktiv. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.4 Daten-Speicherlesefehler (Transfer-Error-Load) (Adresse 0x00000300)

Das externe Transfer-Error-Signal wurde während einem Daten-Lesevorgang aktiv. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.5 Ausrichtungsfehler (Alignment-Error) (Adresse 0x00000400)

Die folgenden Zustände führen zu einem Ausrichtungsfehler:

- die Adresse eines Wort-Zugriff ist kein Vielfaches von 4
- die Adresse eines Halbwort-Zugriff ist kein Vielfaches von 2
- die Adresse eines Sonderregister-Zugriff ist kein Vielfaches von 4

Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.6 Arithmetikfehler (Arithmetic-Error) (Adresse 0x00000500)

Die Funktion in der Arithmetic-Logic-Unit ist nicht definiert. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.7 Undefinierter Befehl (Illegal-Instruction) (Adresse 0x00000600)

Die Befehlsdecodierung im Dispatcher hat einen undefinierten Befehl festgestellt. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.8 Privilegfehler (Privilege-Error) (Adresse 0x00000700)

Nicht implementiert.

3.3.9 Division durch Null (Divide-By-Zero) (Adresse 0x00000800)

Der Divisor einer Division ist Null. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.10 Multiplikation-Division-Überlauf (Multiply-Divide-Overflow) (Adresse 0x00000900)

Das Resultat einer Multiplikation oder Division paßt nicht in das 32 Bit breite Zielregister. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.11 Externer Interrupt (External-Interrupt) (Adresse 0x00000A00)

Eine externe Interrupt-Anforderung wurde angenommen. Dies führt zu einer präzisen Ausnahmebehandlung.

3.3.12 Aufruf eines Betriebssystemdienstes (System-Call) (Adresse 0x00000B00)

Mit Hilfe des 'TRAP 0x101' Befehl können Benutzer diese Ausnahmebehandlung aufrufen und damit Funktionen des Betriebsystems ausführen lassen. Es ist möglich, in einem festgelegtem allgemeinen Register die Adresse eines Parameterblocks an das System zu übergeben. Das Betriebsystem muß vor dem Lesen des Blocks die vom Benutzerprozeß erhaltene effektive Adresse mit Hilfe der Seitentabelle des aktuellen Prozeß in eine reale Adresse umwandeln, da die automatische Adreßumsetzung bei der Ausnahmebehandlung nicht aktiv ist.

3.3.13 Befehls-Adreßumsetzungsfehler (Fetch-Translation-Miss) (Adresse 0x00000C00)

Der Befehls-Adreßumsetzungspuffer kann die virtuelle Seitennummer zum Holen eines Befehls nicht in eine physikalische Seitennummer umsetzen. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.14 Daten-Lese-Adreßumsetzungsfehler (Load-Translation-Miss) (Adresse 0x00000D00)

Der Daten-Adreßumsetzungspuffer kann die virtuelle Seitennummer zum Lesen von Daten nicht in eine physikalische Seitennummer umsetzen. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

3.3.15 Daten-Schreib-Adreßumsetzungsfehler (Store-Translation-Miss) (Adresse 0x00000E00)

Der Daten-Adreßumsetzungspuffer kann die virtuelle Seitennummer zum Schreiben von Daten nicht in eine physikalische Seitennummer umsetzen oder das Modified-Flag für die ausgewählte Speicherseite muß in der Seitentabelle gesetzt werden. Diese Ausnahmesituation führt zu einer präzisen Ausnahmebehandlung.

Das Modified-Flag wird zusammen mit der physikalischen Seitennummer in den Daten-Adreßumsetzungspuffer geschrieben. An dieser Stelle sei darauf hingewiesen, daß das Setzen des Modified-Flag in der Seitentabelle immer mit einer Seitenauslagerung vom Hauptspeicher auf die Festplatte verbunden ist. Demgegenüber ist der Aufwand für eine Ausnahmebehandlung zu vernachlässigen.

3.4 Sonderregister (Special-Purpose-Registers)

Allen Sonderregistern ist eine feste Adresse innerhalb der oberen 128 Bytes des physikalischen Speicherraums zugeordnet (0xFFFFFF80 bis 0xFFFFFFFF). Die Registeradressen sind ein Vielfaches von vier und sämtliche Lese- und Schreibvorgänge müssen als Wort-Zugriffe (LW, SW) durchgeführt werden. Programme von Benutzern können nur dann auf die Sonderregister zugreifen, wenn das Betriebssystem einen entsprechenden Eintrag in der Seitentabelle zur Daten-Adreßumsetzung bereitstellt.

Es folgt eine Zusammenstellung aller Sonderregister.

3.4.1 Interrupt-Enable Register (Adresse 0xFFFFFF80)

Bit 31-1:

nicht benutzt

Bit 0:

0 Externe Interrupts sind gesperrt. In diesem Fall arbeitet der Prozessor im Kernel-Mode und die automatische Adreßumsetzung ist nicht aktiv. Die reale Speicheradresse entspricht so der effektiven Adresse.

1 Externe Interrupts sind zugelassen. Dieses Bit wird während des Starts einer Ausnahmebehandlung gelöscht. Die Ausführung des RFE-Befehl setzt dieses Flag auf den Wert des Bit 0 aus dem Return-From-Exception Register.

3.4.2 Return-From-Exception Register (Adresse 0xFFFFFF84)

Während der Prozessor eine Ausnahmebehandlung startet, wird der Status des Prozessors in diesem Register gerettet. Wenn das unterbrochene Programm später durch die Ausführung des RFE-Befehl fortgesetzt wird, kann so der ursprüngliche Status wieder hergestellt werden.

Bit 31-2:

Wort-Adresse, an der das unterbrochene Programm nach der Ausführung von RFE fortgesetzt werden soll.

Bit 1:

nicht benutzt

Bit 0:

Interrupt-Enable-Flag

3.4.3 Process-Identifier Register (Adresse 0xFFFFFF88)

Die Prozeßidentifikationsnummer kann als eine Erweiterung der effektiven Adresse angesehen werden. Die Adreßumsetzungspuffer benutzen diesen Wert, um die virtuelle Seitennummer eines Prozeß in eine physikalische Seitennummer umzuwandeln. Eine Adresse kann nur dann umgesetzt werden, wenn die angesprochene virtuelle Seite und der Inhalt des Process-Identifier Register einem Eintrag im Adreßumsetzungspuffer entsprechen.

Die Prozeßidentifikationsnummer Null ist für den Kernel-Mode reserviert. In diesem Modus ist die automatische Adreßumsetzung nicht aktiv und die physikalische Speicheradresse entspricht der effektiven Adresse.

Bit 31-4:

nicht benutzt

Bit 3-0:

Prozeßidentifikationsnummer des laufenden Prozeß

3.4.4 ITB-Physical-Page Register (Adresse 0xFFFFFF8C)

Dieses Register wird benutzt, um einen neuen Eintrag in den Befehls-Adreßumsetzungspuffer (instruction translation buffer, ITB) zu schreiben. Wenn ein Programm ein Datenwort an dieser Adresse ablegt, dann enthält dieses Wort die physikalische Seitennummer (physical page) des neuen Eintrags. Die Prozeßidentifikationsnummer und die virtuelle Seitennummer werden immer aus dem Virtual-Page Register entnommen. Da der Adreßumsetzungspuffer als ein direkt abbildender Cache implementiert ist, ergibt sich die Position des neuen Eintrags aus der virtuellen Seitennummer. Aus diesem Grund macht es keinen Sinn, das ITB-Physical-Page Register zu lesen und alle Leseversuche liefern den Wert Null als Resultat.

Bit 31 - 7:

Physikalische Seitennummer des neuen Eintrags

Bit 6 - 0:

nicht benutzt

3.4.5 DTB-Physical-Page Register (Adresse 0xFFFFFF8C)

Dieses Register wird benutzt, um einen neuen Eintrag in den Daten-Adreßumsetzungspuffer (data translation buffer, DTB) zu schreiben. Wenn ein Programm ein Datenwort an dieser Adresse ablegt, dann enthält dieses Wort die physikalische Seitennummer (physical page) des neuen Eintrags. Die Prozeßidentifikationsnummer und die virtuelle Seitennummer werden immer aus dem Virtual-Page Register entnommen. Da der Adreßumsetzungspuffer als ein direkt abbildender Cache implementiert ist, ergibt sich die Position des neuen Eintrags aus der virtuellen Seitennummer. Aus diesem Grund macht es keinen Sinn, das DTB-Physical-Page Register zu lesen und alle Leseversuche liefern den Wert Null als Resultat.

Bit 31 - 7:

Physikalische Seitennummer des neuen Eintrags

Bit 6 - 1:

nicht benutzt

Bit 0:

Modified-Flag des neuen Eintrag

3.4.6 Virtual-Page Register (Adresse 0xFFFFFF94)

Dieses Register enthält die Prozeßidentifikationsnummer und die virtuelle Seitennummer zum Schreiben eines neuen Eintrags in einen der beiden Adreßumsetzungspuffer. Wenn das Betriebssystem einen Adreßumsetzungspuffer mit einem neuen Eintrag laden will, muß es die Nummer der neuen physikalischen Seite in das ITB-Physical-Page Register oder das DTB-Physical-Page Register schreiben. Der neue Eintrag entsteht aus dieser physikalischen Seitennummer und dem Inhalt des Virtual-Page Register.

Während dem Start einer Ausnahmebehandlung für einen Adreßumsetzungsfehler kopiert der Prozessor selbständig die zu übersetzende virtuelle Seitennummer und die aktuelle Prozeßidentifikationsnummer in das Virtual-Page Register.

Bit 31-7

Virtuelle Seitennummer

Bit 6-4:

nicht benutzt

Bit 3-0:

Prozeßidentifikationsnummer

3.5 Beschreibung der externen Signale

Dieses Kapitel beschreibt die externen Signale des DLX-Prozessors. In dieser technologieunabhängigen Implementierung eines Modellprozessors sind alle Kontrollsignale high-aktiv. Das heißt, daß der durch die Bezeichnung beschriebene Zustand dann vorliegt, wenn ein Eins-Pegel am Signal anliegt.

Abbildung 3-4 zeigt die externen Signale der DLX.

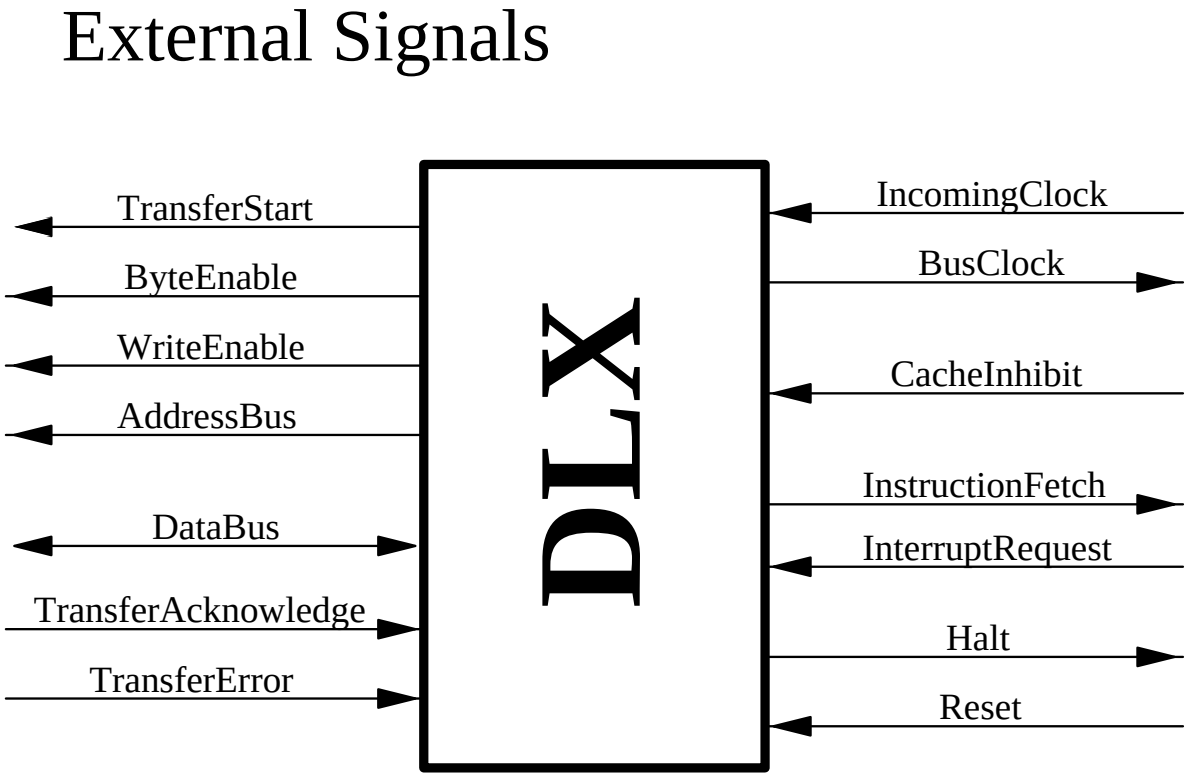

Abbildung 3-4: Externe Signale der DLX

Es folgt eine Beschreibung aller externen Signale.

3.5.1 IncomingClock - Input

Dies ist das externe Taktsignal der DLX. Während der Prozessor arbeitet, sich also nicht im Halt-Zustand befindet, entspricht das interne Taktsignal der IncomingClock. Alle Registerzuweisungen werden durch die positive Flanke des internen Taktsignals ausgelöst. Da die Verhaltensbeschreibung der DLX keine verzögerten Signalzuweisungen enthält, kann das Taktsignal jede beliebige Frequenz besitzen.

3.5.2 BusClock - Output

Dies ist das Taktsignal für die Steuerung des externen Bus. Es entsteht, indem der Takt der IncomingClock durch zwei geteilt wird. Eine positive Flanke der BusClock entspricht dabei immer auch einer positiven Flanke der IncomingClock.

Alle Signale des externen Bus sind auf der positiven Flanke der BusClock gültig bzw. müssen dann gültig sein.

3.5.3 TransferStart - Output

Jeder Speicherzugriff startet mit einer positiven Flanke der BusClock. Dabei aktiviert der Prozessor das TransferStart-Signal für die Dauer einer vollen Taktperiode.

3.5.4 WriteEnable - Output

Durch das WriteEnable-Signal unterscheidet der Prozessor zwischen einem Lese- und einem Schreibzugriff auf den Speicher. Ein Schreibzugriff liegt vor, wenn WriteEnable aktiv ist. Dieses Signal ist während einem aktiven TransferStart-Signal gültig.

3.5.5 ByteEnable(7-0) - Output

Bei einem Schreibzugriff übernimmt der Speicher nur die Bytes des Datenbus, für die das ByteEnable-Signal aktiv ist. Dem höchstwertigen, linken Byte ist hierbei ByteEnable(7) zugeordnet. Dieses Signal ist während einem aktiven TransferStart-Signal gültig und hat für Lesezugriffe keine Bedeutung.

3.5.6 TransferAcknowledge - Input

Das aktivierte TransferAcknowledge-Signal zeigt an, daß der Speicher bzw. die Umgebung des Prozessors einen Speicherzugriff erfolgreich abgeschlossen hat. Die CPU beendet daraufhin den Vorgang. Mit der nächsten positiven Flanke der BusClock kann das Bus-Interface einen neuen Speicherzugriff starten.

3.5.7 TransferError - Input

Das aktivierte TransferError-Signal zeigt an, daß der Speicher bzw. die Umgebung des Prozessor einen Speicherzugriff nicht erfolgreich abschließen konnte. Die CPU beendet daraufhin den Vorgang und meldet einer Ausnahmebehandlung an. Mit der nächsten positiven Flanke der BusClock kann das Bus-Interface einen neuen Speicherzugriff starten.

3.5.8 AddressBus(31-0) - Output

Der AddressBus gibt die physikalische Adresse für einen Speicherzugriff an. Die Adresse ist während dem gesamten Zugriff gültig, also von TransferStart bis TransferAcknowledge oder TransferError.

3.5.9 DataBus(63-0) - Input/Output

Bei einem Schreibzugriff legt der Prozessor die Daten auf den DataBus. Die Werte sind dort ab dem zweiten Bustakt, also erst nach dem TransferStart, gültig. Nachdem der Vorgang durch TransferAcknowledge oder TransferError beendet wurde, schaltet der Prozessor den DataBus in den hochohmigen Zustand.

Bei einem Lesezugriff werden die Daten während dem aktiven TransferAcknowledge-Signal übernommen.

3.5.10 CacheInhibit - Input

Wenn die Umgebung des Prozessor das CacheInhibit-Signal während einem Lesezugriff für Daten aktiviert, werden die Daten nicht in den Daten-Cache der DLX übernommen.

Das CacheInhibit-Signal muß zum Beispiel dann gesetzt werden, wenn das Datenregister von einem E/A-Baustein gelesen wird.

3.5.11 InstructionFetch - Output

Das InstructionFetch-Signal ist aktiv während der Prozessor einen Befehl aus dem Speicher liest. Damit läßt sich von außen ein Codezugriff von einem Datenzugriff unterscheiden.

3.5.12 InterruptRequest - Input

Wenn der externe Interrupt von der CPU freigegeben ist, startet das aktive InterruptRequest-Signal eine Ausnahmebehandlung. Die Routine zur Bearbeitung des Interrupt muß dann die Ursache der Unterbrechung ermitteln und die Ausführung bestätigen. Danach kann das InterruptRequest-Signal inaktiv werden. In der Umgebung des Prozessor müssen die verschiedenen Interrupt-Quellen durch einen Controller verwaltet und an die CPU weitergegeben werden. Ein Register dieses Controllers enthält jeweils die Nummer des unterbrechenden Gerätes. Durch das Lesen dieses Registers kann die DLX gleichzeitig die Ursache des Interrupt erkennen und die Annahme bestätigen.

3.5.13 Reset - Input

Das Reset-Signal setzt den Befehlszähler auf Null, löscht alle Valid-Flags und initialisiert einen Zeiger auf einen Eintrag im Reorder-Buffer.

3.5.14 Halt - Output

Das Halt-Signal zeigt an, daß der Prozessor in den Halt-Zustand versetzt und die interne Clock angehalten wurde. Nur ein Reset kann den Prozessor wieder starten.

4 Einführung in das Arbeiten mit der DLX

Dieses Kapitel gibt eine Einführung in das Arbeiten mit der VHDL-Beschreibung der DLX.

4.1 Installieren der Dateien

Zunächst muß ein Verzeichnis zum Arbeiten mit der DLX-Beschreibung erstellt werden. Dort wird die Datei 'dlx.zip' entpackt. Die Ausführung des 'Makefile' richtet ein Arbeitsverzeichnis für das VHDL-System ein und compiliert die VHDL-Quelldateien. Im Verzeichnis 'test' befindet sich unter anderem die Datei 'Demo.asm', die ein kurzes 'Demonstrationsprogramm für den DLX-Prozessor enthält. Das Kommando 'dlxasm test/Demo.asm' übersetzt das Programm. Der DLX-Code in ASCII-Form befindet sich nun in der Datei 'dlx.out' und wird während dem Start einer Simulation in den Speicher der Prozessorumgebung geladen.

4.2 Durchführen einer Simulation

Nach dem Aufruf des VHDL-Simulator mit dem Kommando 'vsim' erscheint ein Fenster mit einer Auswahl von Designeinheiten. Die Entity 'testbench' enthält das Design für die Simulation der DLX. Von diesem Design wird jetzt die Architektur 'behavior' geladen. In ihr sind die DLX und eine Prozessorumgebung mit Speicher und Interrupt-Controller zu einem Gesamtsystem verbunden.

Das Wave-Fenster, das die Simulationsresultate anzeigen wird, kann aus dem Menü 'View' durch die Auswahl von 'Wave' geöffnet werden. Es wird empfohlen, zur Interpretation eines Simulationsresultats alle Register, Flags und die externen Signale der DLX im Wave-Fenster anzuzeigen. Hierfür ist die Kommandodatei 'AllRegistersAndFlags.do' vorgesehen. Sie wird durch die Funktion 'Execute command file' im Menü 'File' ausgeführt.

Das Programm 'Demo.asm' benötigt 161 Takte zur Simulation. Bei einer willkürlich vorgegebenen Taktzeit von 1µsec entspricht dies 161µsec. Im Menü 'Properties' wird nun die Simulationszeit unter 'Default Run' auf beispielsweise 180.000nsec festgelegt. Anschließend kann eine Simulation über diese Zeit aus dem Menü 'Run' gestartet werden.

Im Wave-Fenster muß unter dem Punkt 'Range' im Menü 'Zoom' der Zeitausschnitt ausgewählt werden, der auf dem Bildschirm dargestellt wird. Hier sollte zu Beginn ein Bereich von 0 - 8.000nsec eingestellt werden. So sind alle Signalwerte und die Zeitskala sichtbar.

4.3 Interpretieren der Simulation

Im Anhang befinden sich tabellarische und bildliche Darstellungen, die den Befehlsfluß von 'Demo.asm' durch die Pipeline dokumentieren. Diese Auszüge zeigen die wichtigsten Registerinhalte zur Interpretation des Simulationsresultats. Die Bearbeitung der einzelnen Befehle in der DLX kann so Schritt für Schritt verfolgt werden.

An dieser Stelle soll noch einmal darauf hingewiesen werden, daß der Dispatcher einen Befehl nicht nur an die entsprechende Ausführungseinheit weiterleitet, sondern gleichzeitig auch in den Reorder-Buffer einträgt. In den Registern der Ausführungseinheit befindet sich immer ein Verweis auf den zugeordneten Eintrag im Reorder-Buffer. Dort speichert die DLX die weiteren Befehlsparameter.

5 Verwendete Literatur

[1] John L. Hennessy, David A. Patterson; Computer Architecture:
A Quantitative Approach, Morgan Kaufmann Publishers, second edition

[2] PowerPC MPC603e RISC Microprocessor User's Manual; Motorola 9/1995

[3] PowerPC MPC500 Family Reference Manual; Motorola 1994

[4] Andrew S. Tanenbaum, Modern Operating Systems, Prentice Hall

[5] John L. Hennessy, David A. Patterson; Rechnerarchitektur:
Analyse, Entwurf, Implementierung, Bewertung; Vieweg Verlag

[6] Gunther Lehmann, Bernhard Wunder, Manfred Selz; Schaltungsdesign mit VHDL;
Franzis Verlag

6 Anhang

Der Anhang enthält die folgenden Ergänzungen:

1. Ausdruck von Testprogrammen mit Angabe der Simulationsdauer
2. Bildliche Darstellung des Befehlsflusses durch die Pipeline
3. Tabellarische Darstellung des Befehlsflusses durch die Pipeline
4. Interfaces aller Entwurfseinheiten
5. Ausdruck der Datei DlxPackage.vhd
6. Ausdruck der Datei Dlx.vhd
7. Ausdruck der Datei Environment.vhd
8. Ausdruck der Datei TestBench.vhd