9. Übung

Fachgebiet Architektur eingebetteter Systeme
Hardwarepraktikum (TI)

AES Embedded Systems Architecture
SoSe 20

Ausgabe: 15.06.2020

Theorie Präsentation: 29./30.06.2020
Praxis Abgabe: 12.07.2020

Arbeitsziel

Verständnis für den Aufbau des Datenpfades und für die Notwendigkeit und Realisierung von Bypassleitungen, Implementierung der Bypasssteuerung im Kontrollpfad.

Arbeitsmaterialien

- Modul `ArmDataPath.vhd`
- Testbench `ArmDataPath_tb.vhd`
- EDIF-Netzlisten ArmProgramStatusRegister, ArmWordManipulation
- Modul `ArmBypassCtrl.vhd`
- Testbench `ArmBypassCtrl_tb.vhd`

Abgabedateien

- Datei `ArmBypassCtrl.vhd`
- Datei `ArmDataPath.vhd`

Theoretische Vorbetrachtungen Die folgenden Fragen sind von der eingeteilten Gruppe in einem kurzen Vortrag zu beantworten. Alle Fragen bezüglich der ARM-Architekur beziehen sich immer auf ISA ARMv4:

- Was wird im Zusammenhang mit Prozessoren unter einer Pipeline verstanden?
- Wie beeinflusst eine Pipeline tendenziell die Ausführungszeit eines Befehls und wie die eines aus vielen Befehlen bestehenden Programms?
- Was ist, bezogen auf einen Prozessor mit Pipeline, ein *Datenflusskonflikt* (bzw. *Datenkonflikt* oder *data hazard*)?
- Unser ARM-Prozessor orientiert sich am ARM9TDMI. Wie viele Pipelinestufen hat dieser Prozessor, wie heißen sie?
- Beschreiben Sie kurz, wie Datenflusskonflikte **ohne** Bypassleitungen in Software und Hardware (also durch die Struktur des Programms bzw. durch das Verhalten des Prozessors) vermieden oder gelöst werden können.
- Was ist ein *Load-Use*-Konflikt und warum kann er durch Bypässe nicht (sinnvoll) gelöst werden?
- Der Registerspeicher wird in unserer Realisierung mit 180° Taktphasenversatz zu den Fließbandregistern betrieben. Schreibzugriffe auf den Registerspeicher erfolgen damit effektiv zur fallenden Flanke des Systemtakts, während aus dem Registerspeicher gelesene Operanden mit der steigenden Flanke des Systemtakts in die Ausgangsregister der *Decode*-Stufe geschrieben werden. Welche Auswirkung hat diese Maßnahme qualitativ auf die Zahl der notwendigen Bypassleitungen und warum ist das so?

Empfohlene Quellen zur Bearbeitung:

- Rechnerorganisation und -entwurf, [2, Kapitel 6.1 bis 6.5]

- Moderne Prozessorarchitekturen, [1, Kapitel 2.2.3]

Aufgabenbeschreibung

Für diese Aufgabe wird ihnen eine fast vollständige Version des Prozessorkern-Datenpfades zur Verfügung gestellt (**ArmDataPath.vhd**). Sie erhalten einige der darin verwendeten Komponenten als Netzliste:

- **ArmProgramStatusRegister**: Im Modul werden die Statusregister (CPSR, SPSRs) des Prozessors verwaltet. Seiner Komplexität durch die Berücksichtigung zahlreicher Randbedingungen beim Beschreiben und Lesen steht aber kein angemessener Lerneffekt gegenüber, weshalb Sie diese Komponente nicht selbst realisieren müssen. Schreibzugriffe auf die Statusregister erfolgen mit der steigenden Systemtaktflanke.
- **ArmWordManipulation**: Aus dem Speicher gelesene Daten müssen nach verschiedenen Kriterien nachbearbeitet werden, bevor Sie in ein Prozessorregister übernommen werden können. Z.B. ist es notwendig, ein Byte oder Halbwort auf 32 Bit zu erweitern, entweder mit null, oder dem Vorzeichen des Datums. Die Manipulation geschieht noch in der MEM-Stufe in der Wortmanipulationseinheit.

Der unvollständige Datenpfad ist in Abbildung 1 visualisiert. Innerhalb des Datenpfades werden folgende globale Steuersignale verwendet:

- `DPA_RST`: Versetzt alle Prozessorregister in ihren Initialzustand (gewöhnlich 0).
- `DPA_CLK`: Gemeinsamer Takt der Status- und Fließbandregister sowie des Instruktionsadressregisters.
- `DPA_INV_CLK`: Taktsignal des Registerspeichers, mit 180° Phasenversatz zu `DPA_CLK`.
- `DPA_ENABLE`: Gemeinsames Aktivierungssignal (fast) aller Register des Datenpfades. Mit `DPA_ENABLE` = 0 können keine neuen Daten mehr in die Fließbandregister geschrieben werden. Einige andere Steuersignale sind mit `DPA_ENABLE` UND-verknüpft, dies wird lediglich durch die Färbung (grün) der Steuersignaleingänge an den gesteuerten Komponenten angedeutet. Durch das globale Enable-Signal kann der Datenpfad angehalten werden.

Nachdem Sie den Datenpfad um die Bypassleitungen erweitert haben, wird die Bypasssteuerung für den Prozessor-Kontrollpfades implementiert.

Aufgabe 1 (2 Punkte) Vervollständigung der Execute-Stufe

Fügen Sie ArmALU, ArmShifter und ArmMultiplier in den Datenpfad (`ArmDatapath.vhd`) ein. Die Komponenten wurden bereits in der VHDL Beschreibung deklariert und ihre Instantiierung ist bereits vorbereitet. Lediglich die Auswahl des Zweiten ALU-Eingangs, siehe Abbildung 1, ist noch unvollständig. Die Entscheidung, ob das Ergebnis des Shifters (`0`) oder des Multiplizierers (`1`) verwendet wird, wird durch das Steuersignal (`DPA_EX_OPB_ALU_MUX_CTRL`) getroffen.

Aufgabe 2 (3 Punkte) Identifikation und Realisierung notwendiger Bypassleitungen

Sämtliche erzeugten Ergebnisse (durch Operandenverknüpfung in der *Execute*-Stufe oder durch Lesen aus dem RAM in der *Memory*-Stufe), die dem Registerspeicher oder dem Statusregister zugeführt werden, können ihrerseits beliebige Operanden nachfolgender Instruktionen sein. Hiervon ist auch ein evtl. erzeugter Condition-Code betroffen. Um eine hohe Auslastung der Pipeline zu gewährleisten, müssen diese Ergebnisse nachfolgenden Operationen durch Bypassleitungen zur Verfügung gestellt werden, noch bevor sie wieder in den Registerspeicher oder das Statusregister zurückgeschrieben

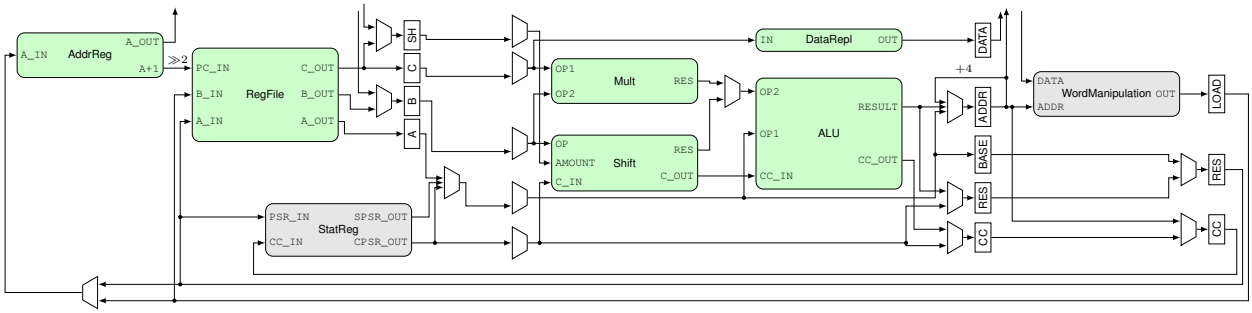

Abbildung 1: Datenpfad des Prozessors ohne Bypass-Leitungen (grün: bereits realisierte Komponenten; grau: vorgegebene Komponenten

werden. Die Operanden/Condition-Code-Bypassleitungen enden in Multiplexern am Beginn der *Execute*-Stufe. Eine Instruktion erhält fehlende Ergebnisse vorangegangener Instruktion, sobald Sie die *Execute*-Stufe erreicht. Identifizieren und notieren Sie die notwendigen Bypässe: welche Signale müssen am Beginn der *Execute*-Stufe zur Verfügung stehen, um Datenkonflikte zu lösen und wo können sie abgegriffen werden?
Um diese Aufgabe lösen zu können, ist ein Grundverständnis der Datenpfad-Struktur notwendig: Operanden werden in der *Instruction Decode*-Stufe gelesen. Zu jedem der 3 Leseports des Registerspeichers gehört ein Ausgangsregister, sie bilden insgesamt das Ende der ID-Stufe im Datenpfad. Diese Register heißen `EX_OPA_REG`, `EX_OPB_REG` und `EX_OPC_REG`. `EX_OPB_REG` nimmt alternativ zu dem aus Port B des Registerspeichers gelesenen Datum einen 32-Bit-Direktoperanden entgegen, der noch im Kontrollpfad des Prozessors auf Basis der aktuell decodierten Instruktion erzeugt wird. Ein viertes Operandenregister, `EX_SHIFT_REG`, ist nur 8 Bit breit und nimmt entweder das niederwertige Byte aus Registerport C oder einen Direktoperanden aus dem Kontrollpfad auf. In der Execute-Stufe treibt das Register den Steuereingang für die Schiebeweite des Shifters (`SHIFT_AMOUNT`).
Die Benennung aller Fließbandregister orientiert sich an der Stufe, deren Beginn sie bilden (EX, MEM, WB) und am Zweck ihrer Verwendung (OP: Operand, RES: errechnetes Ergebnis, Load: aus dem Speicher gelesenes Datum).

In der Execute-Stufe ermittelte Ergebnisse, die in den Registerspeicher oder das Statusregister zurückgeschrieben werden sollen, durchlaufen die Fließbandregister `MEM_RES_REG` und `WB_RES_REG`. Außerdem besteht in der WB-Stufe eine Verbindung zum Instruktionsadressregister, da Operationen mit Ziel R15 einen Sprung verursachen. Aus dem Arbeitsspeicher gelesene Operanden werden in der MEM-Stufe erst in ein Register und anschließend in der WB-Stufe in den Registerspeicher geschrieben. Auch hier besteht die Möglichkeit, ein Ergebnis gleichzeitig in das Instruktionsadressregister zu schreiben und so einen Sprung zu verursachen.

Das Statusregistermodul gibt sowohl das CPSR als auch das SPSR des aktuellen Modus aus, in den Modi USER und SYSTEM hat das Datum am SPSR-Ausgang keine Bedeutung. Beide Ausgänge gehören bereits zum Beginn der Execute-Stufe, weshalb ihnen keine Operandenregister mehr nachgeschaltet sind. Aus dem CPSR wird in der Execute-Stufe der aktuelle Wert des Condition-Codes zur Verwendung in Shifter und ALU abgeleitet, daneben können CPSR und SPSR die Execute-Stufe aber auch in voller Breite als Operand durchlaufen. Das ist notwendig zur Realisierung des MRS-Befehls (Kopie eines Statusregisterinhalts in den Registerspeicher). Der in der Execute-Stufe erzeugte Condition-Code durchläuft die zwei Fließbandregister, bevor er ggf. in das Statusregister zurückgeschrieben wird. Zur Unterstützung eines speziellen Coprozessorbefehls kann der CC-Wert in der MEM-Stufe überschrieben werden. Dies ist im Hardwarepraktikum aber nicht von Bedeutung.

Das in Abbildung 1 mit `BASE` gekennzeichnete Fließbandregister am Ende der Execute-Stufe nimmt den Basiswert von Adressberechnungen für Speicherzugriffe auf und speichert ihn, bis der Speicherzugriff erfolgreich abgeschlossen wurde. Die ARM-ISA ermöglicht im Rahmen von LDM-Instruktionen, dass das Basisregister der Zugriffsadresse durch ein neues Datum überschrieben wird. Scheitert anschließend einer der weiteren Lesezugriffe, muss das Basisregister wiederhergestellt werden. Die Wiederherstellung erfolgt aus diesem `BASE` Register. Das Register spielt für die Operandenbypässe **keine** Rolle!

Für die Operanden A, B und C und SHIFT kommen jeweils 3 Bypassquellen infrage, für den Condition-Code sind zwei Bypassquellen zu finden. Ergänzen Sie in der Abbildung 1 die Eingänge der Bypassmultiplexer in der Execute-Stufe um die von ihnen identifizierten Bypässe. Vervollständigen Sie außerdem die Beschreibung der Bypassmultiplexer in **ArmDataPath.vhd** entsprechend. Alle Bypassmultiplexer sind vom 4:1-Typ. Die Erzeugung der jeweiligen Steuersignale wird Teil der vierten Aufgabe sein sein. Für die Verbindung der Bypassmultiplexer mit den Bypassleitungen soll gelten: Der Bypass des ALU-Ergebnisses aus der *Memory*-Stufe wird durch die Multiplexersteuerleitungskombination `01` durchgeschaltet, der entsprechende Wert in der *Writeback*-Stufe durch `10`. Allgemein soll gelten: je „kleiner“ die Steuerleitungskombination, desto näher liegt die Bypassquelle an der *Execute*-Stufe. Bypassleitungen, die mit der Steuersignalkombination „11“ an den Bypassmultiplexer angeschlossen sind, sind maximal weit von der *Execute*-Stufe entfernt. Die Eingänge zu den Steuerleitungskombinationen 11 und 10 von `EX_CC_MUX` werden mit dem selben Signal verbunden.

Der Test Ihrer Lösung erfolgt in der nächsten Teilaufgabe.

Aufgabe 3 (2 Punkte) Test und Synthese des Datenpfades

Testen Sie den vervollständigten Datenpfad mit der Testbench **ArmDataPath_tb.vhd**. Für die Simulation werden Simulationsmodelle der vorgegebenen Komponenten **ArmProgramStatusRegister** und **ArmWordManipulation** benötigt. Daneben erhalten Sie Simulationsmodelle für Komponenten aus vorangegangenen Aufgabenblättern (**ArmDataReplication**, **ArmShifter**, **ArmALU**,**ArmMultiplier**, **ArmRegfile**), die Sie nur verwenden dürfen, wenn Ihre eigene Implementierung nicht korrekt funktioniert hat. Kommentieren Sie die korrespondierenden *USE*-Anweisungen in **ArmDataPath.vhd** ein, damit keine fehlerhaften Teilkomponenten zur Simulation verwendet werden.

Führen Sie eine Synthese des Datenpfades durch. Kommentieren Sie dazu die *LIBRARY*- und *USE*-Anweisungen zu **ARM_SIM_LIB** in **ArmDataPath.vhd** vollständig aus. Einige der im Datenpfad notwendigen Komponenten stehen Ihnen nicht im Quelltext zur Verfügung, sondern werden als EDIF-Netzliste im Verzeichnis
/afs/tu-berlin.de/units/Fak_IV/aes/lib/hwpti/ARM_LIB bereitgestellt. Entfernen Sie alle Komponenten aus dem Projekt, die in den bisherigen Aufgabenstellungen nicht fehlerfrei implementiert worden sind. Auch dafür stehen Netzlisten im AFS bereit. Stellen Sie sicher, dass die Synthese des Datenpfades möglich ist, keine Latches entstehen und keine nicht nachvollziehbaren Warnungen auftreten. Warnungen mit Bezug auf vorgegebene Komponenten können ignoriert werden.

Aufgabe 4 (5 Punkte) Implementierung der Bypasssteuerung

Sie haben den Prozessor-Datenpfad um die notwendigen Bypassleitungen ergänzt. Nun soll die dazu gehörende Steuerung der Bypass-Multiplexer als Teil des Kontrollpfades implementiert werden. Außerdem identifiziert und meldet das Steuerungsmodul *Load-Use*-Konflikte, die nicht durch Bypassing/Forwarding von Operanden aufgelöst werden können.

Die Grafik 2 auf der letzten Seite des Aufgabenblattes verdeutlicht, wo und auf welcher Basis die

Steuersignale für die Bypass-Multiplexer erzeugt werden. Dies geschieht bereits in der Decode-Stufe, obwohl sich die Multiplexer am Beginn der Execute-Stufe befinden. Am Ende der ID-Stufe im Kontrollpfad werden die ermittelten Steuersignale in Fließbandregister geschrieben und stehen im Folgetakt frühzeitig in der EX-Stufe zur Verfügung. Die Erzeugung der Steuersignale und ihre Wirkung auf die Bypässe sind also um einen Takt versetzt!

Zur Formulierung der Anforderungen an die Bypasssteuerung gilt folgende Vereinbarung:
Die Instruktion, deren Bypasssteuerleitungen bestimmt werden sollen, wird als INST0 bezeichnet und befindet sich in der *Decode*-Stufe des Prozessors. Die unmittelbar vorhergehende Instruktion befindet sich zeitgleich in der *Execute*-Stufe und heißt INST1. In der *Memory*-Stufe befindet sich zu diesem Zeitpunkt INST2. Während des Decodierens von INST0 befindet sich eine vierte Instruktion, INST3, in der *Write-Back*-Stufe. Sie spielt für die Bypasssteuerung keine Rolle, da sie den Registerspeicher aktualisiert, bevor INST0 Operanden liest.
INST1 und INST2 können jeweils bis zu 2 Werte in den Registerspeicher schreiben. Zu diesem Zweck werden Rückschreib-Registeradressen und Registerspeicher-Write-Enable-Signale gemeinsam mit den Instruktionen durch den Kontrollpfad-Teil des Prozessor-Fließbandes bewegt. Außerdem können beide Instruktionen den Condition-Code im Statusregister ändern. Dies geschieht genau dann, wenn zwei Steuersignale, die parallel zur Instruktion das Fließband durchlaufen, gesetzt sind.

Die Bypasssteuerung ist nicht unmittelbar Teil der Prozessorsteuerung, um deren Komplexität zu verringern und einfacher auf Änderungen der Pipelinestruktur regieren zu können.
Es ist aber möglich, die Bypasssteuersignale aus den Signalen abzuleiten, die die Prozessorsteuerung für INST0, INST1 und INST2 erzeugt hat. Die Signale von INST0 sind unmittelbar in der *Decode*-Stufe verfügbar. Die Steuersignale von INST1 stehen in Pipelineregistern des Kontrollpfades zwischen *Decode*- und *Execute*-Stufe zur Verfügung, während sich INST0 in der *Decode*-Stufe befindet. Die Steuersignale von INST2 können gleichzeitig aus Pipelineregistern im Kontrollpfad zwischen *Execute*- und *Memory*-Stufe gelesen werden.

Bypässe müssen immer dann geschaltet werden, wenn ein Datenkonflikt vorliegt, INST1 oder INST2 also ein Datum aktualisieren, welches von INST0 gelesen werden soll. Im Kern beruht die Erkennung von Konflikten auf dem Vergleich zwischen den Register-Leseadressen von INST0 und den Register-Schreibadressen von INST1 und INST2. Die Adressvergleiche in unserer Implementierung finden dabei mit den übersetzten 5-Bit-Registeradressen statt.

Die Bypasssteuerung muss im Modul **ArmBypassCtrl.vhd** implementiert werden. Die Eingänge des Moduls repräsentieren die Steuersignale und Adressen von INST0, INST1 und INST2 aus den verschiedenen Pipelinestufen.

`ABC_INST0_R_PORT_A_ADDR(4:0)`	`IN`	Zugriffsadresse des Registerspeichers für Operand **A** von **INST0** nach der Adressübersetzung.
`ABC_INST0_R_PORT_B_ADDR(4:0)`	`IN`	Zugriffsadresse des Registerspeichers für Operand **B** von **INST0** nach der Adressübersetzung.
`ABC_INST0_R_PORT_C_ADDR(4:0)`	`IN`	Zugriffsadresse des Registerspeichers für Operand **C** von **INST0** nach der Adressübersetzung.
`ABC_INST1_W_PORT_A_ADDR(4:0)`	`IN`	Rückschreibadresse für ein Ergebnis der *Execute*-Stufe von **INST1**
`ABC_INST1_W_PORT_B_ADDR(4:0)`	`IN`	Rückschreibadresse für ein Datum aus dem Datenspeicher von **INST1**

ABC_INST2_W_PORT_A_ADDR(4:0)	IN	Rückschreibadresse für ein Ergebnis der *Execute*-Stufe von **INST2**
ABC_INST2_W_PORT_B_ADDR(4:0)	IN	Rückschreibadresse für ein Datum aus dem Datenspeicher von **INST2**
ABC_INST1_W_PORT_A_EN	IN	*Write-Enable* für Write-Port **A** des Registerspeichers. **INST1** erzeugt in der *Execute*-Stufe ein Ergebnis und wird dieses in den Registerspeicher schreiben.
ABC_INST1_W_PORT_B_EN	IN	*Write-Enable* für Write-Port **B** des Registerspeichers. **INST1** lädt ein Datum aus dem Speicher und wird dieses in den Registerspeicher schreiben. Bei gleicher Adresse für Port **A** und **B** wird Port **A** automatisch priorisiert
ABC_INST2_W_PORT_A_EN	IN	*Write-Enable* für Write-Port **A** des Registerspeichers. **INST2** erzeugt in der *Execute*-Stufe ein Ergebnis und wird dieses in den Registerspeicher schreiben
ABC_INST2_W_PORT_B_EN	IN	*Write-Enable* für Write-Port **B** des Registerspeichers. **INST2** lädt ein Datum aus dem Speicher und wird dieses in den Registerspeicher schreiben.
ABC_INST1_WB_PSR_EN	IN	*Write-Enable* des Statusregisters. **INST1** wird ein Datum aus der *Execute*-Stufe (Ergebnis oder Condition-Code) in das Statusregister schreiben.
ABC_INST1_WB_PSR_SET_CC	IN	Für WB_PSR_EN = 1 und WB_PSR_SET_CC = 1 aktualisiert **INST1** den Condition-Code im Statusregister. Andere Schreibzugriffe auf das Statusregister spielen für die Bypasssteuerung **keine** Rolle.
ABC_INST2_WB_PSR_EN	IN	Write-Enable des Statusregisters. **INST2** wird ein Datum aus der *Execute*-Stufe (Ergebnis oder Condition-Code) in das Statusregister schreiben.
ABC_INST2_WB_PSR_SET_CC	IN	Für WB_PSR_EN = 1 und WB_PSR_SET_CC = 1 aktualisiert **INST2** den Condition Code im Statusregister.
ABC_INST0_REGS_USED(2:0)	IN	Steuersignale die Anzeigen, welche Registerspeicher-Leseports **INST0** wirklich benötigt. Index 2 entspricht Operand C, Index 1 entspricht Operand B, Index 0 entspricht Operand A.
ABC_INST0_SHIFT_REGS_USED	IN	Steuersignal das Anzeigt, ob INST0 die Schiebeweite des Shifters aus Registerspeicherport C erhält (1) oder stattdessen einen Direktoperanden verwendet (0).
ABC_INST0_OPA_BYPASS_MUX_CTRL(1:0)	OUT	Steuersignale des Operand **A** - Bypassmultiplexers von **INST0**.
ABC_INST0_OPB_BYPASS_MUX_CTRL(1:0)	OUT	Steuersignale des Operand **B** - Bypassmultiplexers von **INST0**.

ABC_INST0_OPC_BYPASS_MUX _CTRL(1:0)	OUT	Steuersignale des Operand **C** - Bypassmultiplexers von **INST0**.
ABC_INST0_SHIFT_BYPASS_MUX _CTRL(1:0)	OUT	Steuersignale des **SHIFT** - Bypassmultiplexers von **INST0**.
ABC_INST0_CC_BYPASS_MUX _CTRL(1:0)	OUT	Steuersignale des Condition-Code - Bypassmultiplexers von **INST0**.
ABC_LOAD_USE_CONFLICT	OUT	Zeigt an, dass **INST1** ein Datum aus dem Speicher lädt, das als Operand (**A**, **B**, **C** oder **SHIFT**) von **INST0** benötigt wird.

INST0 kann bis zu 3 Operanden aus dem Registerspeicher lesen (**A**, **B** und **C**) und zusätzlich den aktuellen **Condition-Code** aus dem Statusregister benötigen. Die Funktionsweise der Bypasssteuerung wird im folgenden für Operand **A** von INST0 besprochen, für das Bypassing der Operanden **B** und **C** gilt entsprechendes.

Operand **A** von INST0 kann ein beliebiges Datum aus dem Registerspeicher sein, gelesen von Adresse ABC_INST0_R_PORT_A_ADDR. Eine Instruktion kann in der WB-Stufe bis zu zwei Ergebnisse in den Registerspeicher schreiben: Ein Datum, das in der *Execute*-Stufe erzeugt wurde und ein Datum, das aus dem Datenspeicher gelesen wurde. Ergebnisse aus der *Execute*-Stufe werden immer in Schreibport **A** des Registerspeichers geschrieben, Daten aus dem Datenspeicher immer in Schreibport **B**. Versucht eine Instruktion, über Port **A** und **B** auf die gleiche Adresse zu schreiben, wird das Datum von Schreibport **A** priorisiert (Sie haben dieses Verhalten im Registerspeicher implementiert).
Stimmt die Leseadresse ABC_INST0_R_PORT_A_ADDR von Operand **A** von INST0 mit einer der Rückschreibadressen von INST1 oder INST2 überein, muss potenziell eines der Ergebnisse von INST1 oder INST2 durch Bypassleitungen der Instruktion INST0 zur Verfügung gestellt werden.

Die Operand **A**-Adresse von INST0 muss also mit insgesamt 4 Rückschreibadressen verglichen werden. Dieser Adressvergleich ist in **ArmBypassCtrl.vhd** bereits vorgegeben. Die vier Vergleichsergebnisse vom Typ boolean tragen die Bezeichnung A0_equal_A1, A0_equal_B1, A0_equal_A2 und A0_equal_B2.

Für A0_equal_A2 = true gilt: INST2 schreibt ein Ergebnis in den Registerspeicher, dass INST0 als Operand **A** benötigt. Dieser Wert liegt in WB_RES_REG vor, wenn sich INST0 in der *Execute*-Stufe befindet. Steuern Sie den Bypassmultiplexer EX_OPA_MUX durch ABC_INST0_OPA_BYPASS _MUX_CTRL entsprechend an.

Für A0_equal_B2 = true gilt: INST2 liest ein Datum aus dem Speicher, dass INST0 als Operand **A** benötigt. Das Datum liegt im Register WB_LOAD_REG vor, wenn INST0 die EX-Stufe erreicht. Steuern Sie den Bypassmultiplexer entsprechend an.

Für A0_equal_A1 = true gilt: INST1 wird über Schreibport **A** ein Datum in den Registerspeicher schreiben. Wenn sich INST0 in der *Execute*-Stufe befindet, kann dieses Datum aus dem Register MEM_RES_REG gelesen werden.

Für A0_equal_B1 = true gilt: INST1 wird in der *Memory*-Stufe ein Datum aus dem Datenspeicher lesen, das INST0 als Operand **A** benötigt. In diesem Fall liegt ein *LOAD-USE*-Konflikt vor, das

Datum kann nicht gebypasst werden. Die Bypasssteuerung signalisiert der Prozessor-Hauptsteuerung diesen Zustand durch eine 1 an Ausgang `ABC_LOAD_USE_CONFLICT`. Außerdem wird `ABC_INST0_OPA_BYPASS_MUX_CTRL` auf `00` gesetzt. Die Prozessor-Hauptsteuerung reagiert, indem sie Instruktion 0 für einen weiteren Takt in der Decode-Stufe hält und sie dann erneut decodiert. Ein NOP füllt die Lücke, die die verzögerte Instruktion 0 im Fließband hinterlässt.

Potenziell könnten mehrere (sogar alle) Adressvergleiche identische Adressen aufzeigen. Es gilt: Ergebnisse von INST1 sind gegenüber Ergebnissen von INST2 beim Bypassing priorisiert (denn sie werden später erzeugt). Ergebnisse an Schreibport **A** sind gegenüber Ergebnissen an Schreibport **B** priorisiert.

Jeder Instruktion sind im Kontrollpfad zwangsläufig zwei Rückschreibadressen zugeordnet (die entsprechenden Signale können nicht „nichts“ anzeigen), auch wenn nicht in den Registerspeicher zurückgeschrieben werden soll. Zu jeder Rückschreibadresse existiert daher ein *Write-Enable*-Signal. Ein erfolgreicher Adressvergleich darf nur dann zum Bypassing von Operanden oder dem Anzeigen eines Load-Use-Konflikts führen, wenn auch das zugehörige *Write-Enable*-Signal gesetzt ist. Stimmen beispielsweise die Leseadresse **A** von INST0 und die Schreibadresse **A** von INST1 überein (`A0_equal_A1 = true`), wobei das *Write-Enable*-Signal `ABC_INST1_W_PORT_A_EN` nicht gesetzt ist, so darf der Wert in `MEM_RES_REG` nicht durch die Bypassleitung in der *Execute*-Stufe verwendet werden, denn er wird auch nicht in den Registerspeicher zurückgeschrieben.

Ein ähnliches Problem ergibt sich für die Registerspeicher-Leseadressen. Alle drei haben für INST0 einen definierten Wert. Es ist aber möglich, dass die Instruktion generell weniger Operanden benötigt, oder einer der beteiligten Operanden nicht aus dem Registerspeicher stammt (also ein Direktoperand ist). Wird ein Operand gar nicht benötigt oder ist er ein Direktoperand, hat die zugeordnete Registerspeicher-Leseadresse keine Bedeutung. Es darf in diesem Fall weder eine Bypassleitung geschaltet, noch ein Load-Use-Konflikt angezeigt werden! Ob eine Instruktion einen bestimmten Operanden aus dem Registerspeicher verwendet, kann am Steuervektor `INST0_REGS_USED` abgelesen werden. `INST0_REGS_USED = 110` bedeutet beispielsweise, dass INST0 nur die Operanden C und B aus dem Registerspeicher verarbeitet und nur dafür Bypässe geschaltet und Konflikte angezeigt werden dürfen. Einen Bypass nicht zu schalten bedeutet, dass die Steuersignale des jeweiligen Bypassmultiplexers den Wert `00` annehmen.

Eine Besonderheit ergibt sich für die Bypässe der 8-Bit-Schiebeweite (Multiplexer EX_SHIFT_MUX im Datenpfad). Seine Steuerung (Ausgang `ABC_INST0_SHIFT_BYPASS_MUX_CTRL`) ist fast vollständig mit der der Operand C Bypässe identisch. Eine aus dem Registerspeicher gelesene Schiebeweite entspricht ja dem niederwertigen Byte von Operand C. Weil Operand C aber auch anderweitig verwendet werden kann, muss ein weiteres Steuersignal anzeigen, ob die Schiebeweite wirklich aus Registerport C stammt oder ein Direktoperand ist. Hierzu dient `INST0_SHIFT_REGS_USED`. Nur wenn diese Signal gemeinsam mit `INST0_REGS_USED(2)` gesetzt ist, darf EX_SHIFT_MUX Bypässe für die Schiebeweite schalten. Auch die Schiebeweite kann einen Load-Use-Konflikt verursachen.

Die Steuerung des Condition-Code-Multiplexers gestaltet sich sehr einfach. INST1 aktualisiert den Condition-Code genau dann, wenn sowohl `ABC_INST1_WB_PSR_EN` als auch `ABC_INST1_WB_PSR_SET_CC` gesetzt ist. INST2 aktualisiert den Condition-Code genau dann, wenn sowohl `ABC_INST2_WB_PSR_EN` als auch `ABC_INST2_WB_PSR_SET_CC` gesetzt ist. Auch

hier ist INST1 gegenüber INST2 beim Bypassing priorisiert, wenn beide den Condition-Code verändern. Der Condition-Code wird immer in die *Execute*-Stufe gebypasst, wenn eine Instruktion (INST1, INST2) ihn verändert, es bestehen keine weiteren Bedingungen. Der CC-Bypassmultiplexer ist so anzusteuern, dass entweder die Bits 31:28 des CPSR-Ausgangs des Statusregister-Moduls oder die Werte der Register `MEM_CC_REG` bzw. `WB_CC_REG` in der *Execute*-Stufe zur Verfügung stehen, je nachdem, ob INST1 und/oder INST2 den Condition-Code verändern.

Vervollständigen Sie **ArmBypassCtrl.vhd**, sodass alle fünf Bypassmultiplexer der *Execute*-Stufe korrekt angesteuert werden. Testen Sie die Implementierung durch eine Verhaltenssimulation mit **ArmBypassCtrl_TB.vhd**.

HINWEIS

- **ArmBypassCtrl.vhd** ist rein kombinatorisch. Sämtliche Pipelineregister sind im Kontrollpfad bereits implementiert.
- Die *Enable*-Signale der Execute-Eingangsregister (`EX_OPA_REG_EN` etc.) im Datenpfad sind permanent gesetzt und müssen daher nicht berücksichtigt werden.
- Zeigt eine Instruktion an, dass Sie ein Ergebnis in den Registerspeicher schreiben wird, sind auch die *Enable*-Signale der Pipelineregister (`MEM_RES_REG_EN` und `WB_RES_REG_EN` bzw. `WB_LOAD_REG_EN`) gesetzt und müssen daher nicht berücksichtigt werden.
- `PSR_EN` ist das Schreibzugriffs-Steuersignal des Statusregister-Moduls. Nur wenn es gesetzt ist, können die Statusregister verändert werden. Die Detailwirkung hängt von zahlreichen Steuersignalen ab, von denen für die Bypässe aber nur `PSR_SET_CC` ausgewertet wird.
- Nach jeder Veränderung des Statusregisters über den Condition-Code hinaus werden für einige Takte `NOP`s in die Pipeline geschrieben. Währenddessen muss das Bypassing nicht korrekt funktionieren, weshalb die übrigen Steuersignale des Statusregisters nicht berücksichtigt werden.

Mündliche Rücksprache - 4 Punkte

Literatur

[1] Matthias Menge. *Moderne Prozessorarchitekturen. Prinzipien und ihre Realisierungen*. 1. Aufl. Springer, Berlin, März 2005. ISBN: 3540243909.

[2] David A. Patterson und John L. Hennessy. *Rechnerorganisation und-entwurf*. Spektrum Akademischer Verlag, Sep. 2005, S. 593. ISBN: 3827415950, 9783827415950.

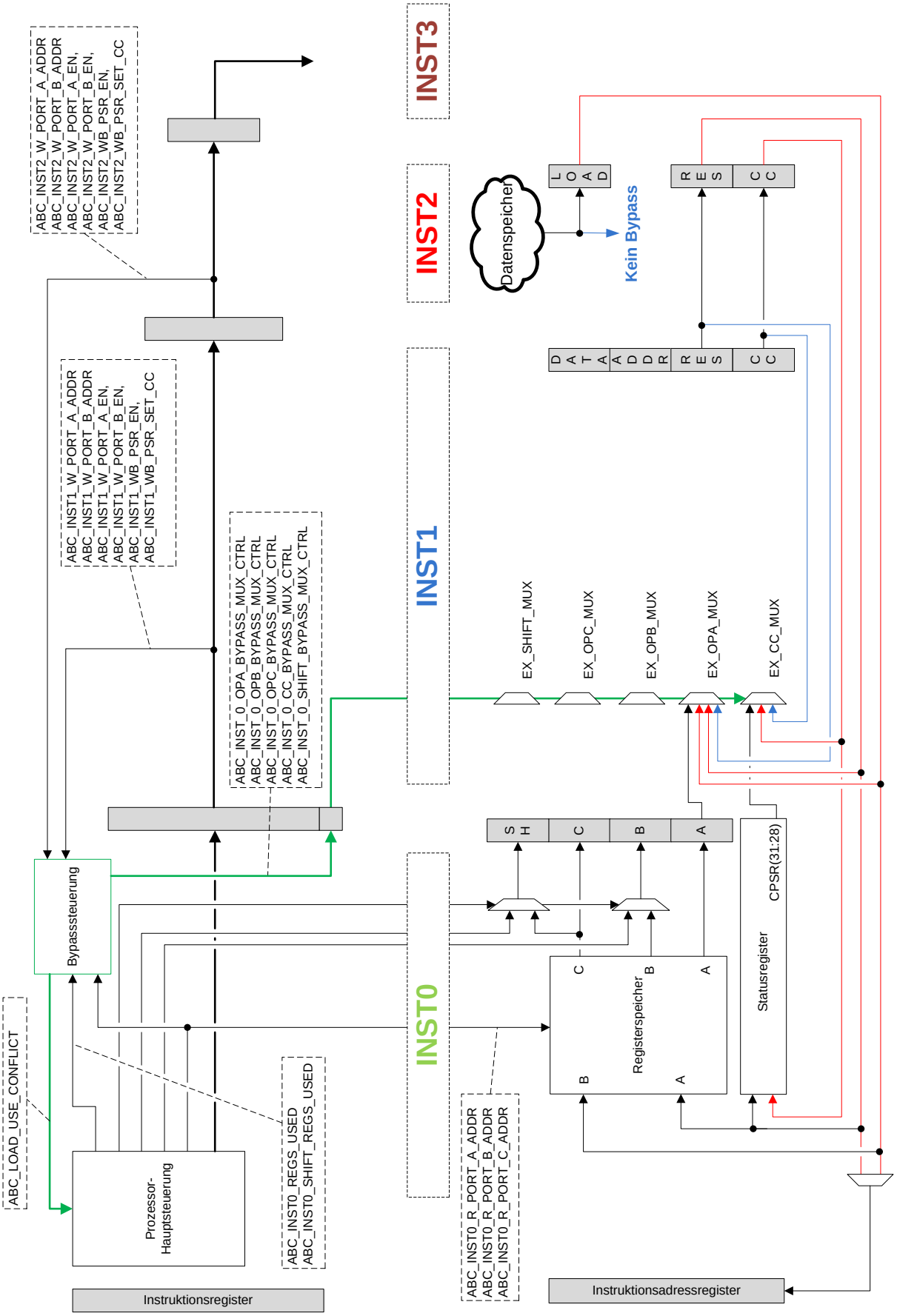

Abbildung 2: Bypassleitungen