georg.von-der-brueggen [☺] tu-dortmund.de
harun.teper [☺] tu-dortmund.de

Hardwarepraktikum
Wintersemester 21/22

Übungsblatt 9 – 58 Punkte

(Block C2 – insgesamt 92 Punkte)

Bearbeiten ab Samstag, 8. Januar 2022.
Abgabe bis spätestens Freitag, 21. Januar 2022, 23:59 Uhr.

Vorwort: In Übung 8 war es eure Aufgabe, einen RAM-Baustein in VHDL zu implementieren. Aus Rechnerstrukturen (RS) wisst ihr, dass Speicher ein elementarer Bestandteil von Prozessoren ist. Während dies in RS am Beispiel von RISC V erklärt wird, wird hier die verwandte MIPS-Architektur verwendet. Wir werden uns anschauen, wie diese Prozessorarchitektur schrittweise in VHDL aufgebaut werden kann. Durch ausführliches Testen der einzelnen Komponenten werden wir sicherstellen, dass der Prozessor korrekt ausführt. Danach werden wir das Prozessor-Design weiter vervollständigen. Wenn ihr die Design-Prinzipien dieses Prozessors verstanden habt, wird es euch leichter fallen, den Aufbau und die Funktionsweise anderer Architekturen zu verstehen.

Mit euren bisherigen Erfahrungen in sequentiellen und kombinatorischen Schaltungen, Speichern, und komplexeren Bausteinen wie Addierern und Multiplizierern, beherrscht ihr bereits alle notwendigen Grundlagen um diese Übung zu bearbeiten.

Ihr müsst mit der begrenzten Zeit, die wir im HaPra haben, natürlich nicht den ganzen Prozessor selbst implementieren. Wir stellen euch den Code für die einzelnen Bausteine zur Verfügung. Eure Aufgabe in dieser Übung ist es, den Prozessorcode zu verstehen, sicherzustellen, dass alle Komponenten und Funktionen korrekt implementiert wurden und danach mit euch bekannten Bausteinen zu erweitern. In der folgenden Aufgabe werden wir zuerst den MIPS-Datenpfad aufbauen und uns erst später mit der Steuerung beschäftigen.

9.1 MIPS-Datenpfad (21 Punkte)

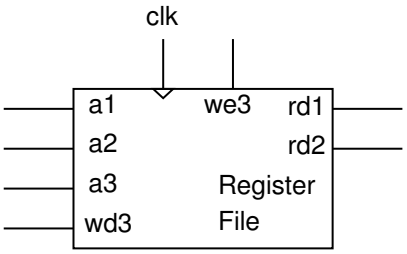

Abbildung 1: Der MIPS-Registersatz

Der Datenpfad der MIPS-Prozessorarchitektur arbeitet mit 32-Bit Worten und enthält Speicher, einen Registersatz, eine ALU, Multiplexer und einige weitere einfache Komponenten.

9.1.1 Zustandselemente

Für das Design des Datapath fangen wir mit Komponenten an, die den Zustand des Prozessors beschreiben. Dazu gehören die Register. In Abbildung 1 ist der Registersatz dargestellt. Zudem findet ihr in der Vorlage den Ordner *regfile*, in welchem der Registerbaustein implementiert wurde. Hier werdet ihr einige Ähnlichkeiten zu dem RAM-Baustein in der letzten Übung feststellen. Bearbeitet zunächst folgende Aufgaben:

a. (2 Punkte) Beschreibt den Aufbau des Registerbausteins und erklärt, wie Daten gespeichert bzw. gelesen werden.

b. (2 Punkte) Schreibt eine Testbench für den Registerbaustein und testet, indem ihr zwei verschiedene Werte in zwei veschiedenen Registern speichert und diese danach auslest.

Ein weiteres Zustandselement ist der Befehlszähler, auch program counter (PC) genannt. Der PC gibt die Adresse der Instruktion aus, die ausgeführt werden soll. In dieser Übung wird der PC als synchron rücksetzbares Flip Flop (FF) realisiert. Die Implementierung des PCs befindet sich im Ordner *syncresff* und das Diagramm in Abbildung 2.

c. (1 Punkt) Beschreibt die Funktionsweise des PCs und testet den Baustein in einer Testbench.

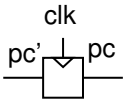

Abbildung 2: Befehlszähler

9.1.2 Designprozess

Generell ist es im Designprozess ratsam, so wie oben mit den Zustandselementen anzufangen. Die Speicherelemente für die Register und den PC reichen uns vorerst. Eigentlich benötigen wir noch weitere Speicher, unter anderem für Instruktionen und Daten. Die Details dieser Bausteine behandeln wir aber später. Wir nehmen einfach an, dass der *-Baustein (in Abbildung 3) die Ausgabe des PCs in die zu ausführende Instruktion umwandelt.

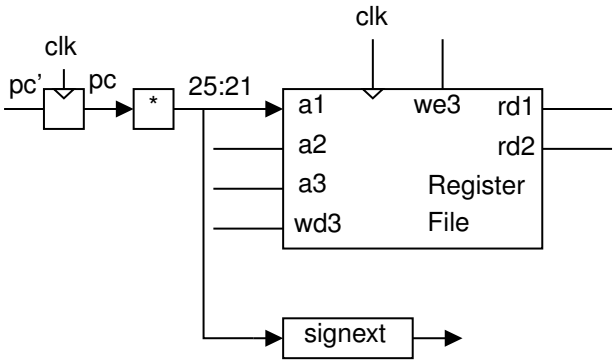

Abbildung 3: PC, Instruktionsspeicher (*-Baustein), Registersatz und der Baustein zur Vorzeichenerweiterung

Unser nächstes Ziel im Designprozesses ist es, mithilfe von Instruktionen den neuen Zustand des Prozessors ausgängig von dem vorherigen Zustand zu berechnen. Dies erreichen wir, indem wir kombinatorische Schaltungen zwischen die Zustandslemente legen. Wir müssen sicherstellen, dass unser Prozessor jederzeit die Instruktion korrekt ausführt und dabei den Zustand des Prozessors auch korrekt modifiziert.

9.1.3 Load Word (lw)

Eine der elementarsten MIPS-Instruktionen ist die load word (lw) Instruktion. lw liest ein Wort aus dem Datenspeicher in ein Register ein. Um die Adresse im Datenspeicher des zu ladenden Wortes zu berechnen, wird der Wert des Registers als Basisadresse genutzt, welches im dem Feld 25:21 der Instruktion spezifiziert ist. Zudem wird noch ein Offset in dem Feld 15:0 angegeben, welcher zu dem Wert im Register addiert wird. Der Offset muss aber vor der Addition erst auf 32 Bit erweitert werden (es ist schließlich eine 32-Bit CPU). Für die Vorzeichenerweiterung nutzen wir den *signext*-Baustein (siehe Abbildung 3).

a. (1 Punkt) Beschreibt die Funktionsweise des signext-Bausteins und testet diesen in einer Testbench.

Der signext-Baustein ermöglicht es, den Wert in den 16 niederwertigsten Bits der Instruktion als Offset zu nutzen, den wir zur Basisadresse im dafür spezifizierten Register addieren. Jetzt muss der Prozessor den Offset zu der Basisadresse hinzuaddieren. Zum Addieren nutzen wir eine ALU, welche auch für andere Operationen zuständig ist. Das Ergbnis der ALU, das die Adresse des zu ladenden Wortes darstellt, wird an den **-Baustein (Datenspeicher) weitergeleitet. In Abbildung 4 ist eine ALU an die Register, an den signext-Baustein und an den Datenspeicher angeschlossen. Die Implementierung der ALU könnt ihr im Ordner *alu* einsehen.

b. (2 Punkt) Beschreibt die Funktionsweise der ALU und testet alle Funktionen in einer Testbench.

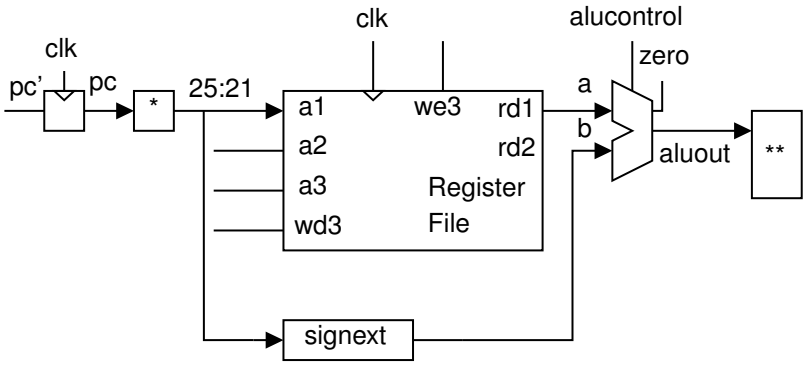

Abbildung 4: Die ALU wurde an die Register und an den signext-Baustein angeschlossen. Der **-Baustein stellt den Datenspeicher dar.

Wie oben bereits beschrieben, wird das Ergebnis der ALU an den Datenspeicher (**-Baustein) weitergeleitet. Dieser gibt einen Wert aus, welcher an den wd3-Port des Registersatzes angelegt wird. Hierbei wird das Register, in welches das Ergebnis gespeichert werden soll (Bits 20:16 in der Instruktion), an den a3-Port angelegt. Dieser Vorgang ist in Abbildung 5 dargestellt.

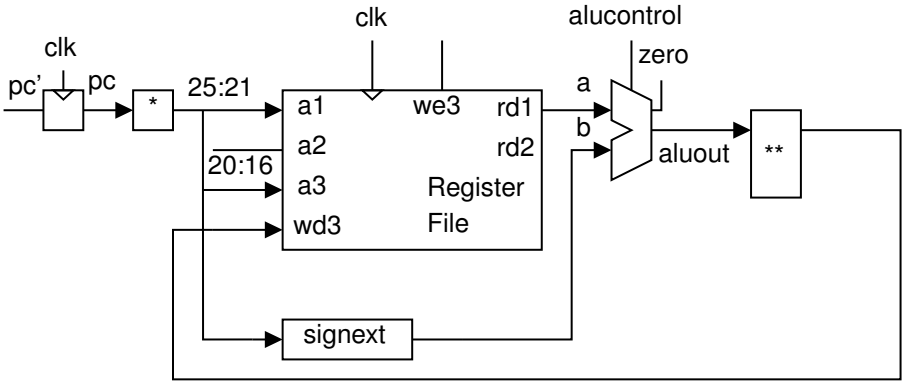

Abbildung 5: lw-fähiger MIPS-Prozessor

Die lw-Instruktion kann nun ausgeführt werden. Im normalen Betrieb gibt es für den Prozessor aber fast immer eine nächste Instruktion. Deswegen muss, während die lw-Instruktion ausgeführt wird, der Wert im PC erhöht werden, damit die Adresse der nächsten Instruktion ausgegeben werden kann. Um dies zu ermöglichen, nutzen wir einen Addier-Baustein, der in jedem Zyklus die Zahl 4 zum aktuellen Wert im PC hinzuaddiert. Der Addier-Baustein findet ihr im Ordner *adder* und in Abbildung 6 wurde die Schaltung entsprechend ergänzt. Die Zahl vier muss addiert werden, weil der Instruktionsspeicher Bytes speichert und wir mit einem 32-Bit Prozessor arbeiten.

c. (1 Punkt) Testet den Addier-Baustein in einer Testbench.

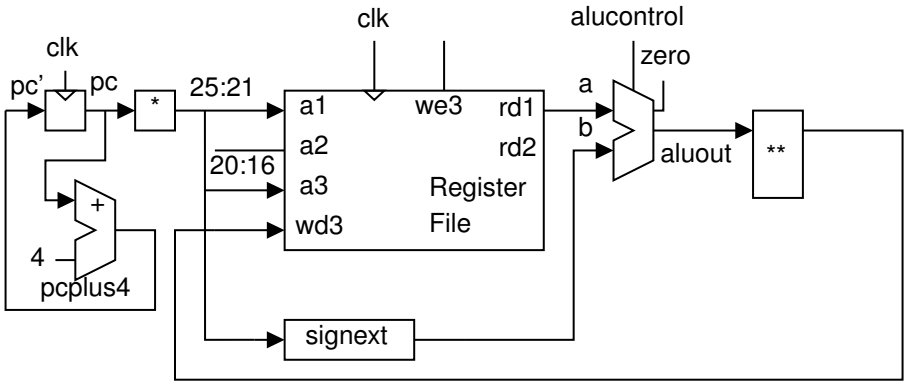

Abbildung 6: lw-fähiger MIPS-Prozessor mit pcplus4-Schaltung

9.1.4 Store Word (sw)

Die nächste Instruktion, die wir mit unserem Prozessor ausführen wollen, ist store word (sw). Bei der sw-Instruktion geben die fünf Bits 20:16 in der Instruktion das Register an, dessen Inhalt in den **-Baustein gespeichert werden soll. Die Bits 20:16 werden also an den a2-Port angelegt, wodurch der Inhalt des Registers am rd2-Port ausgegeben und zum Datenspeicher (**) weitergeleitet wird. Die Adresse, an welcher das Wort geschrieben werden sollen, befinden sich dabei im den Register, welches in den Bits 25:21 spezifiziert ist. Der Inhalt dieses Registers wird der ALU direkt übergeben. Der zweite Input der ALU wird durch den Offset angegeben (Bits 15:0). Dieser wird auch hier auf 32-Bit erweitert. Die ALU berechnet dann die entgültige Adresse, an welcher der Wert, der an rd2 ausgegeben wird, gespeichert wird. Die für die sw-Instruktion benötigten Verbindungen im Datenpfad wurden in Abbildung 7 ergänzt.

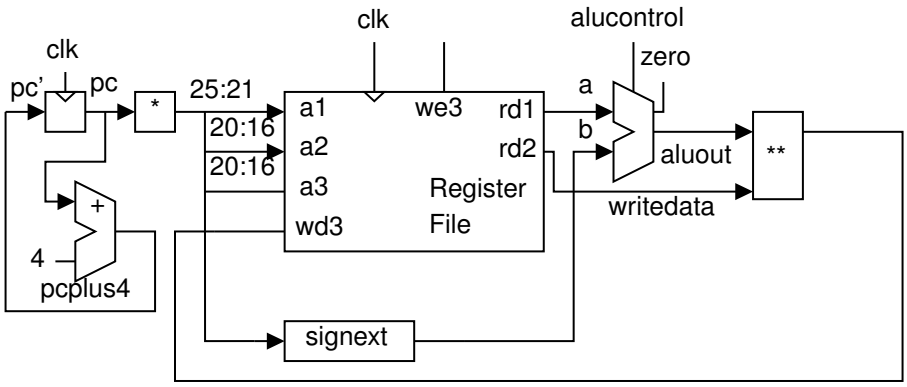

Abbildung 7: sw-fähiger MIPS-Datenpfad

9.1.5 Register-Typ (R-Typ)

Der Prozessor in Abbildung 7 hat zwar eine ALU, aber bisher haben wir diese nur für das Addieren von Offset und Basisadresse genutzt. Wir wollen aber auch Instruktionen des R-Typs (Register) ausführen, d.h. add, sub, and, or und slt. Dabei werden in der Instruktion zwei Register spezifiziert, auf deren Inhalt die ALU eine der vorhin genannten Operationen ausführt. Um dies zu ermöglichen, müssen wir den Datapath um drei Multiplexer (mux) ergänzen. In Abbildung 8 wählt der regdst-mux aus, aus welchem Feld die Adresse des Registers gewählt werden soll. Der alusrc-mux wählt aus, ob der zweite Operant aus den Registern oder von dem immediate field gewählt werden soll. Der memtoreg-mux wählt aus, ob das Ergbenis der ALU oder jedoch die Ausgabe des **-Bausteins in ein Register geschrieben werden soll (den pcsrc-mux könnt ihr vorerst ignorieren). Im Ordner *mux* findet ihr eine Implementierung des Multiplexers.

a. (1 Punkt) Testet den mux-Baustein in einer Testbench.

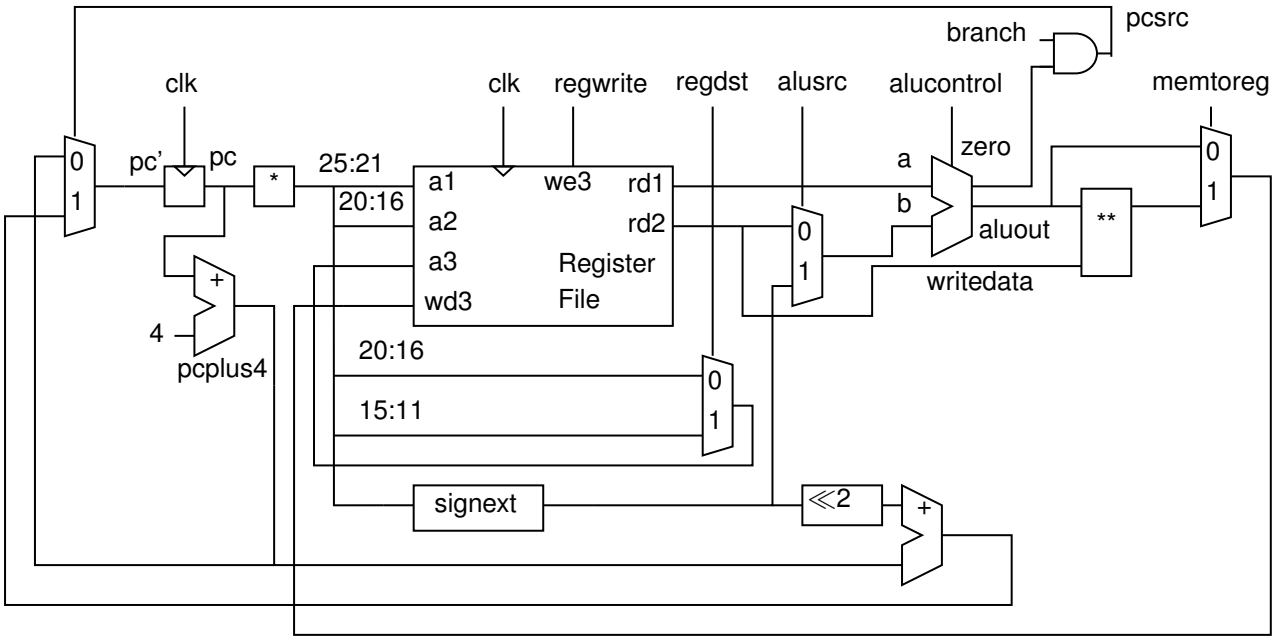

Abbildung 8: Mit Multiplexern ergänzter MIPS-Datenpfad

9.1.6 Branch On Equal (beq)

Jetzt erweitern wir das Design so, dass die Branch on Equal (beq) Instruktion ausgeführt werden kann. Hierbei werden die Werte von zwei Registern verglichen. Falls diese gleich sind, dann wird zum PC der Wert im Offset-Feld addiert (bei Ungleichheit geht es mit PC+4 weiter). Vorher muss jedoch der Wert im Offset-Feld auf 32 Bit erweitert und mit 4 multipliziert werden. Das Multiplizieren realisieren wir mit einem Baustein, der die Eingabe um zwei Stellen nach links schiebt. In Abbildung 8 (siehe unten, «2 und der Addierer) wurden die Schaltungen ergänzt. Die Implementierung des Schiebebausteins könnt ihr in im Ordner *sl2* finden. Das Addieren führen wir mit dem bereits bekannten Addier-Baustein durch.

a. (1 Punkt) Testet den sl2-Baustein in einer Testbench.

9.1.7 Jump (j)

Bei einer Jump (j) Instruktion nimmt der PC den Wert an, der in den Bits 25:0 spezifiziert ist. Der 26-Bit lange Wert wird vorher um zwei Stellen nach links geschoben. Jetzt fehlen für eine komplette Adresse noch 4 Bit. Die werden aus den vier höchstwertigsten Bits von dem Wert PC+4 übernommen. Der jump-fähige MIPS-Datenpfad ist in Abbildung 9 dargestellt.

9.1.8 Zusammensetzen des Datenpfades

In der Datei *datapath.vdhl* wurden die Komponenten des Datenpfades bereits zusammengesetzt. Dort findet ihr auch zu jeder Komponente eine Zahl in eckigen Klammen (z.B. [7] für *rf: regfile*). In Abbildung 9 fehlen auch noch die Beschriftungen der Verbindungen.

a. (8 Punkte) Tragt die Zahlen in eckigen Klammern in der Datei *datapath.vdhl* an die zugehörigen Komponenten in Abbilding 9 (z.B. eine [7] in die Register File Komponente) ein. Vervollständigt zudem die Beschriftung der Verbindungen in Abbildung 9. Schaut dazu auch in die Datei *datapath.vdhl*. Es wurden z.B. schon *pcplus4*, *writedata*, *zero* und *aluout*, sowie alle Kontrollsignale (außer für Speicher) eingetragen. Es fehlen noch: *pcjump*, *pcnext*, *pcnextbr*, *pcbranch*, *signimm*, *signimmsh*, *srca*, *srcb*, *result*. Tragt diese auch in Abbildung 9 ein.

b. (2 Punkt) Erstellt eine Liste der Steuersignale und beschreibt kurz, welchem Zweck diese jeweils dienen.

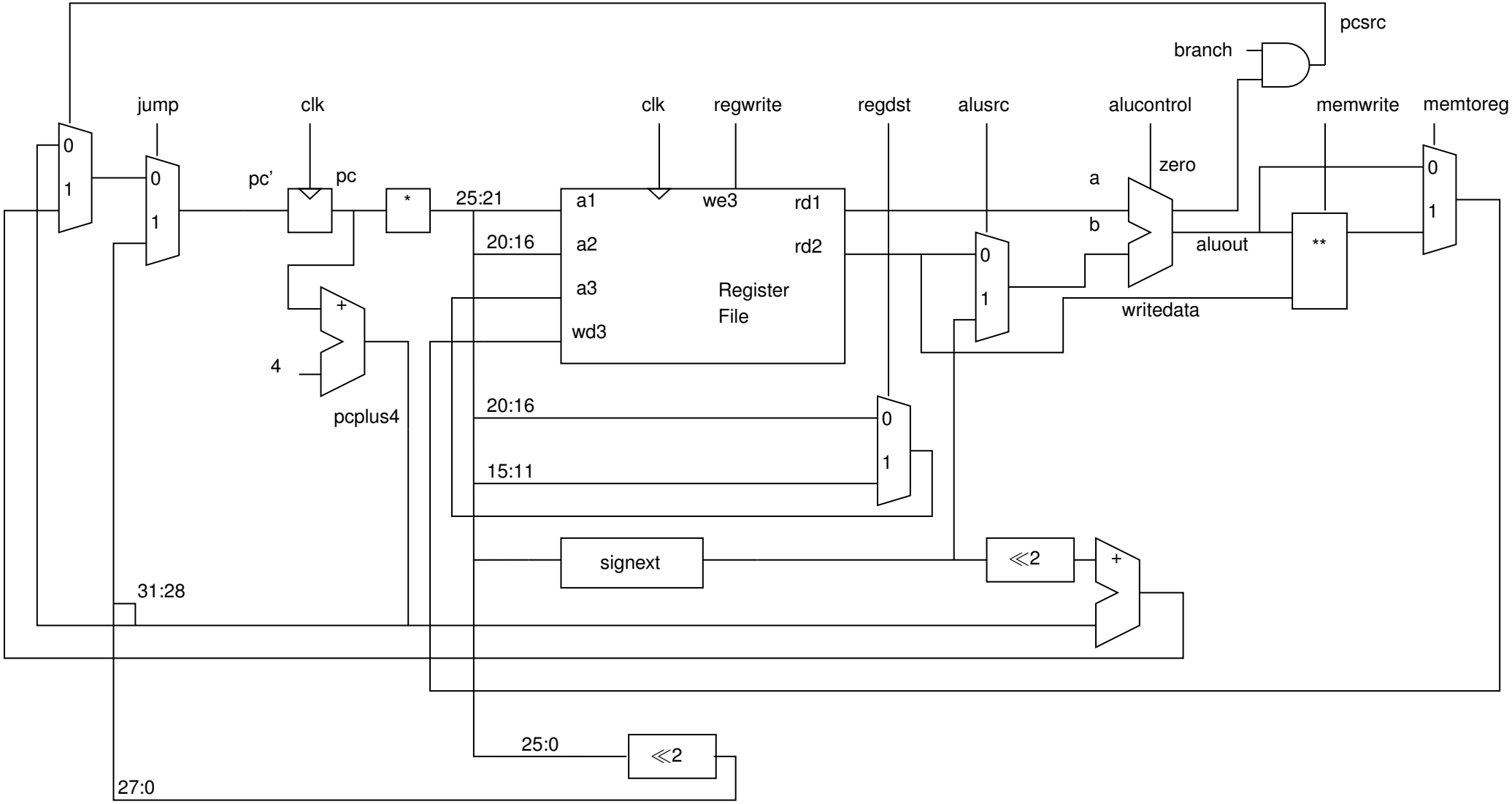

Abbildung 9: Jump-fähiger MIPS-Datenpfad

9.2 MIPS-Steuerung (27 Punkte)

Jetzt fassen wir den gesamten Datenpfad als einen Baustein auf. In Abbildung 10 ist unser Datenpfad-Baustein dargestellt und Instruktions- und Datenspeicher sind bereits angeschlossen (in dem Code noch nicht!).

9.2.1 Manuelle Steuerung

Da wir nun alle Komponenten einzeln getestet haben, möchten jetzt den gesamten Datenpfad-Baustein testen. Hier überprüfen wir, ob die Komponenten des Prozessors (in *datapath.vhdl*) korrekt zusammengesetzt wurden. Dazu übernehmen wir jetzt die Aufgabe des Steuerelements. In den folgenden Aufgaben sollt ihr eine Testbench für den Datenpfad implementieren. Für die Tests müsst ihr die einzelnen Bits der Instruktion und die Steuersignale selbst einstellen. Kommentiert eure Testbench während der Bearbeitung dieser Aufgaben genau. Dies geschieht in den folgenden Schritten:

a. (4 Punkte) Sucht euch zwei unterschiedliche Zahlen zwischen eins und zehn aus (Bitte schreibt eure Zahlen in als Kommentar in den Code). Speichert diese zwei Zahlen in zwei verschiedenen Register. Lest die beiden Werte aus, und überprüft ob diese korrekt sind.

b. (4 Punkte) Addiert die beiden Zahlen in den Registern und speichert das Ergebnis in einem dritten Register, welches von den beiden zu addierenden verschieden ist. Lasst euch den Inhalt des dritten Registers ausgeben und überprüft, ob korrekt addiert wurde.

c. (4 Punkte) Addiert nun einen Offset (0-10, bitte in den Kommentaren angeben) zu dem Wert, der im dritten Register steht, und speichert das Ergebnis in einem vierten Register. Gebt den Inhalt des vierten Registers aus und überprüft das Ergebnis.

d. (2 Punkte) Überprüft, ob jump korrekt ausgeführt wird. Zum Überprüfen könnt ihr euch dazu die gespeicherten und anliegenden Werte am PC Flip Flop anschauen.

e. (2 Punkte) Überprüft, ob beq korrekt ausgeführt wird. Auch hier könnt ihr zum Überprüfen euch die gespeicherten und anliegenden Werte am PC Flip Flop anschauen.

9.2.2 Automatisierte Steuerung

Da es schwierig wäre die CPU immer selbst steuern zu müssen, wurden in der Vorlage Steuereinheiten implementiert.

a. (2 Punkte) Beschreibt die Funktionalität von *aludecoder* und testet diesen Baustein in einer Testbench.

b. (2 Punkte) Beschreibt die Funktionalität von *maindecoder* und testet diesen Baustein in einer Testbench.

c. (1 Punkt) Beschreibt die Funktionalität von *controller*.

d. Studiert die *mips.vhdl* und schreibt eine Testbench. Testet in der Testbench folgende Befehle:

   (1) (2 Punkte) Nutzt die addi Instruktion, um, wie in 9.2 (gleiche Zahlen und gleiche Register), die beiden Zahlen in die zwei Register zu speichern. Lest die Werte zum Überprüfen wieder aus.

   (2) (2 Punkte) Nutzt die R-type Instruktion, um die Addition der beiden Zahlen auszuführen und in das dritte Register zu schreiben. Lest den Wert zum Überprüfen aus.

   (3) (1 Punkt) Testet die jump Instruktion. Zum Überprüfen könnt ihr euch dazu die gespeicherten und anliegenden Werte am PC Flip Flop anschauen.

   (4) (1 Punkt) Testet die beq Instruktion. Auch hier könnt euch dazu die gespeicherten und anliegenden Werte am PC Flip Flop anschauen.

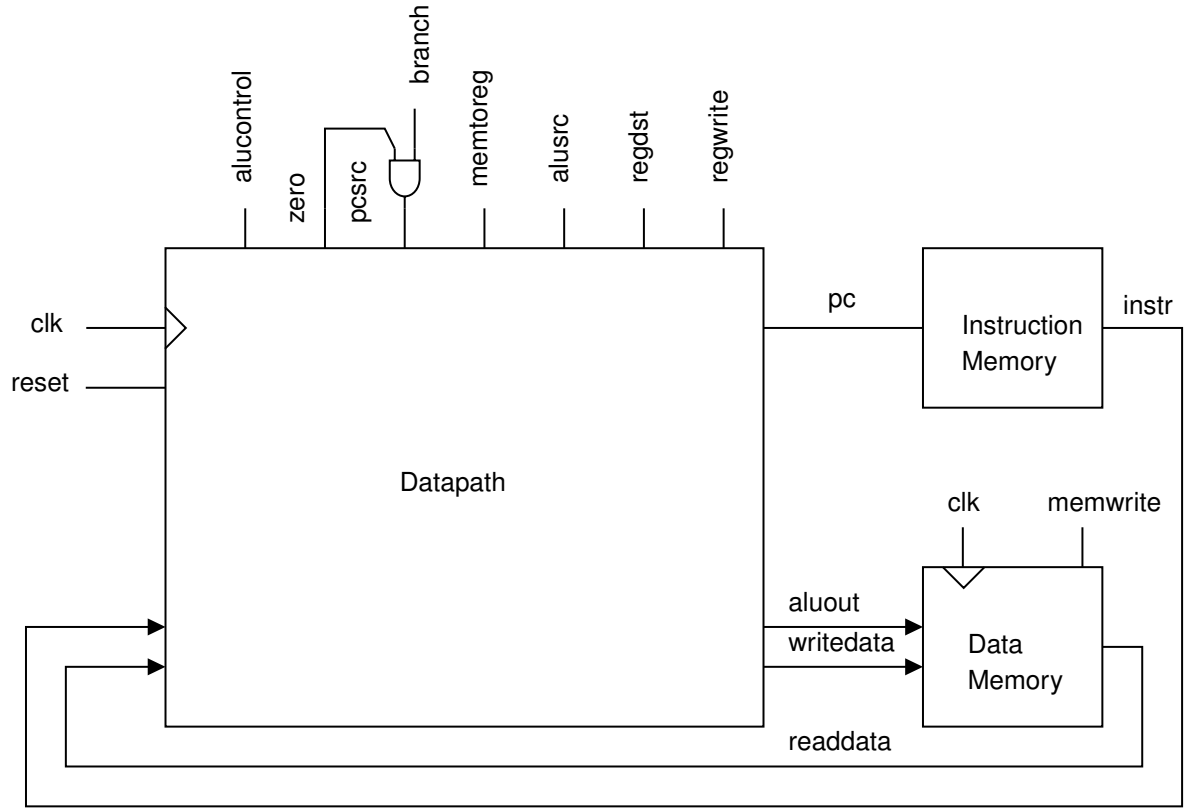

Abbildung 10: MIPS-Datenpfad mit angeschlossenen Speichern und Steuereingängen

9.3 Anschluss der Speicher (10 Punkte)

Bisher haben wir ohne Daten- und Instruktionsspeicher gearbeitet. In dieser Aufgabe werden wir den MIPS-Prozessor um diese beiden Speicherbausteine erweitern.

a. (5 Punkte) Implementiert einen Instruktionsspeicher, der in seinem Speicherfeld Bytes nutzt. Ihr könnt euch hierzu euren RAM-Speicher als Vorlage nehmen. Schreibt dort eure 4 Instruktionen hinein und testet den Baustein in einer Testbench. Schließt danach den Speicher an die CPU an und lasst die CPU in einer Testbench die Instruktionen ausführen. Wir nehmen hier einen read-only Instruktionsspeicher an.

b. (5 Punkte) Implementiert einen Datenspeicher, der ebenfalls Bytes speichert. Testet die Funktionalität in einer Testbench. Auch hierzu könnt ihr euren RAM-Speicher als Vorlage nehmen. Schließt den Datenspeicher an den Prozessor an und ermöglicht es, Daten zu schreiben und zu lesen (mit sw and lw). Testet das Schreiben und Lesen mit sw und lw Instruktionen in einer Testbench.

9.4 Verbesserungen des MIPS/Implementierung einer besseren CPU (5 Punkte)

Arbeitet die Nachteile dieses CPU-Designs heraus und verbessert sie, oder implementiert eine ganz andere CPU. Das ist aber natürlich nur eine Zusatzaufgabe (und entsprechend sind die 5 Punkte die hier erreicht werden können reine Bonuspunkte). Wer Interesse hat kann bei uns am Lehrstuhl ähnliche Projekte im Fachprojekt oder in Form einer Bachelorarbeit bearbeiten! ☺

Literatur

[1] Harris and Harris, Digital Design and Computer Architecture. Morgan Kaufman Publishers Inc., 2007