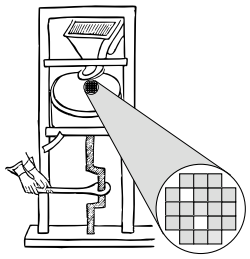
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch
Holger Lange
Mathias Halbach (FG Rechnerarchitektur)

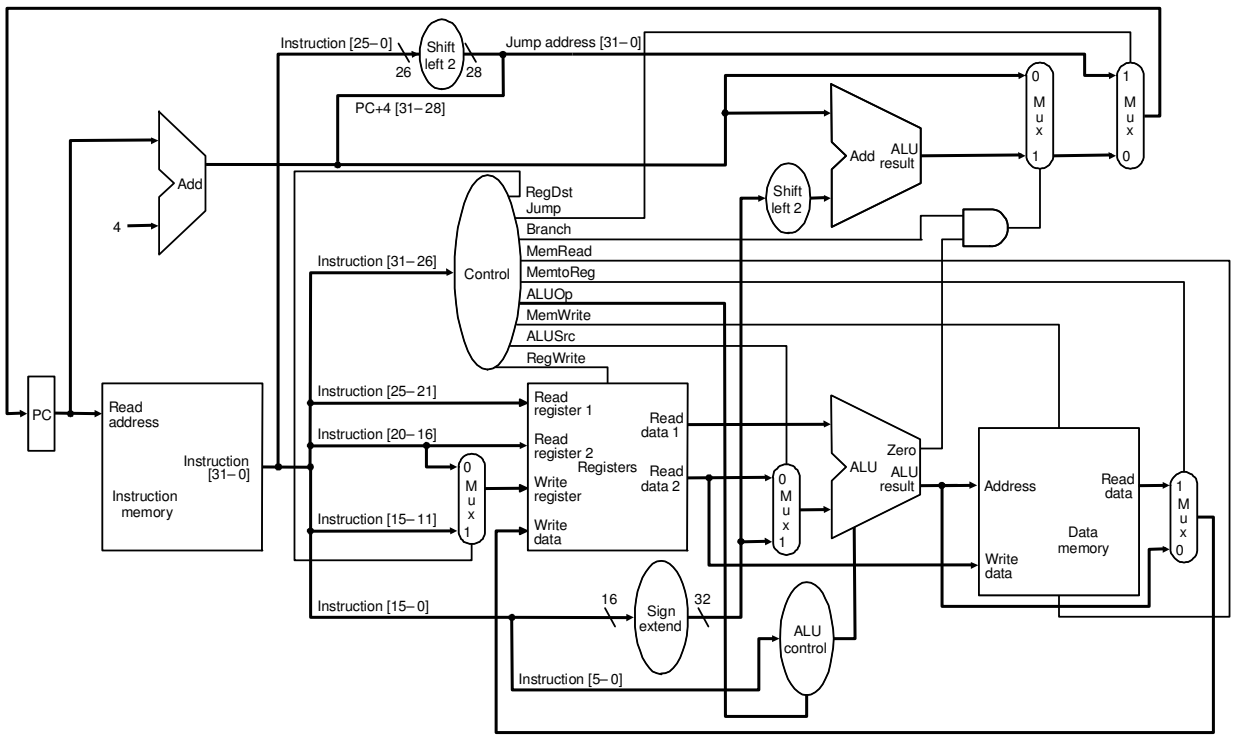


06.07.2006

Technische Grundlagen der Informatik II
9. Übung – MIPS Datenpfad
Sommersemester 2006

Aufgabe 1: MIPS-Kontrollsignale

Für die folgenden 5 Befehlstypen sind jeweils im MIPS-Datenpfad der Eintaktimplementierung (Abbildung siehe unten, entnommen aus Vorlesung Kapitel 8, Folie 29) die genutzten Wege der Daten (nicht der Steuersignale) zu kennzeichnen. Geben Sie außerdem an, wie die Steuersignale gesetzt sein müssen.

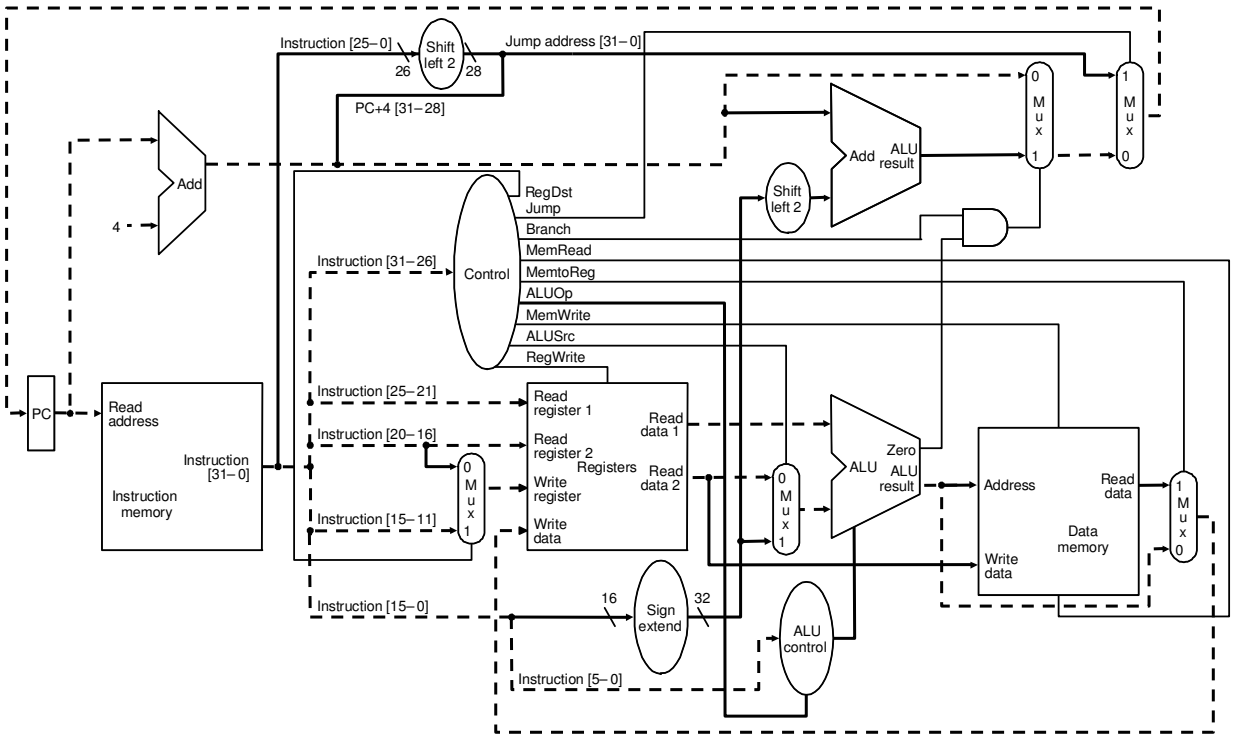


a) R-Format (add, sub, and, or, slt zwischen Registern)

Lösung:

Die genutzten Wege der Daten sind in der folgenden Lösung gestrichelt gezeichnet, die Steuersignalwerte sind der Vorlesung Kapitel 8, Folie 16 entnommen.

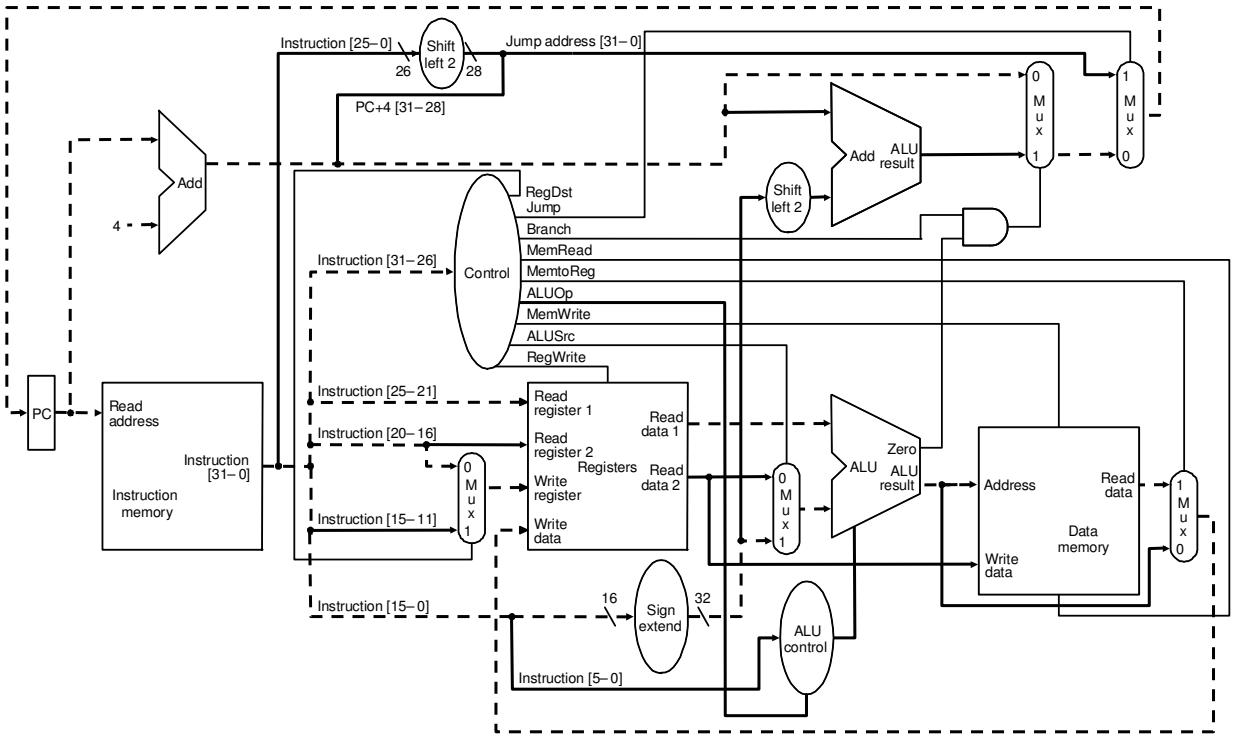
RegDst: 1, Jump: 0, Branch: 0, MemRead: 0, MemtoReg: 0, MemWrite: 0, ALUSrc: 0, RegWrite: 1, ALUOp: 10 (benutze FUNCT-Feld)



b) Load

Lösung:

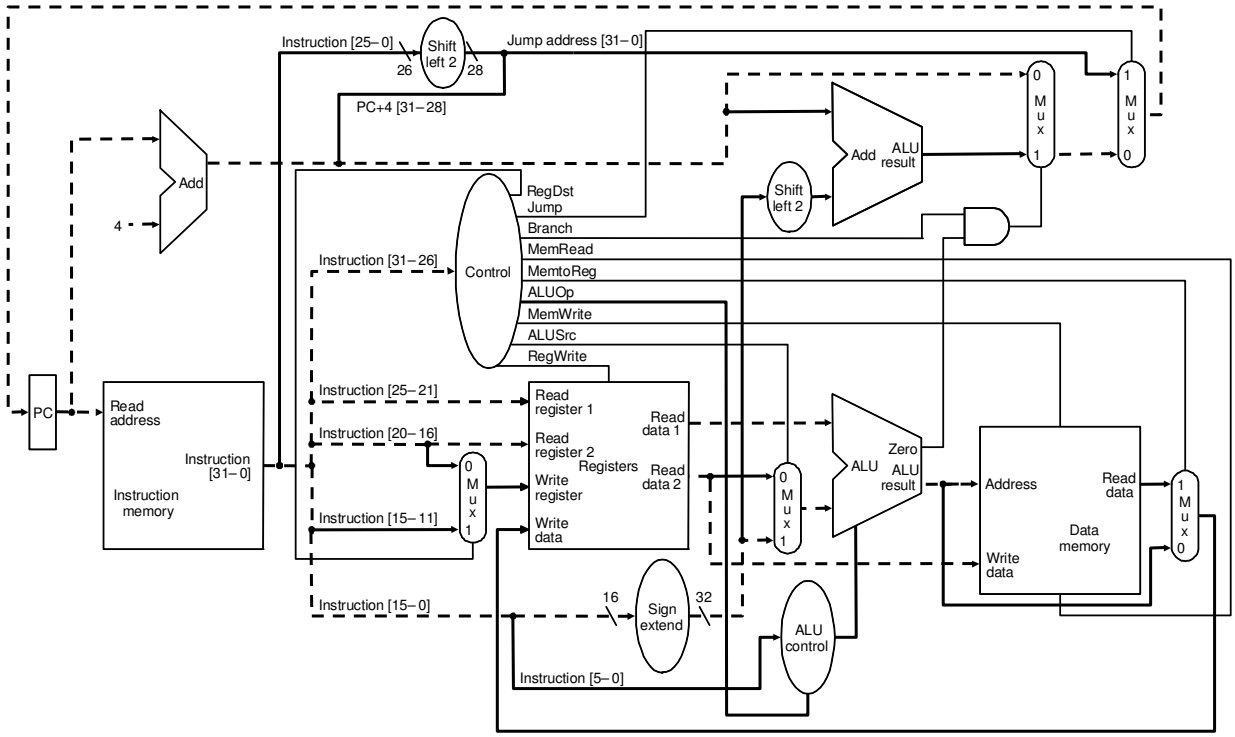
RegDst: 0, Jump: 0, Branch: 0, MemRead: 1, MemtoReg: 1, MemWrite: 0, ALUSrc: 1, RegWrite: 1, ALUOp: 00 (Adressberechnung)



c) Store

Lösung:

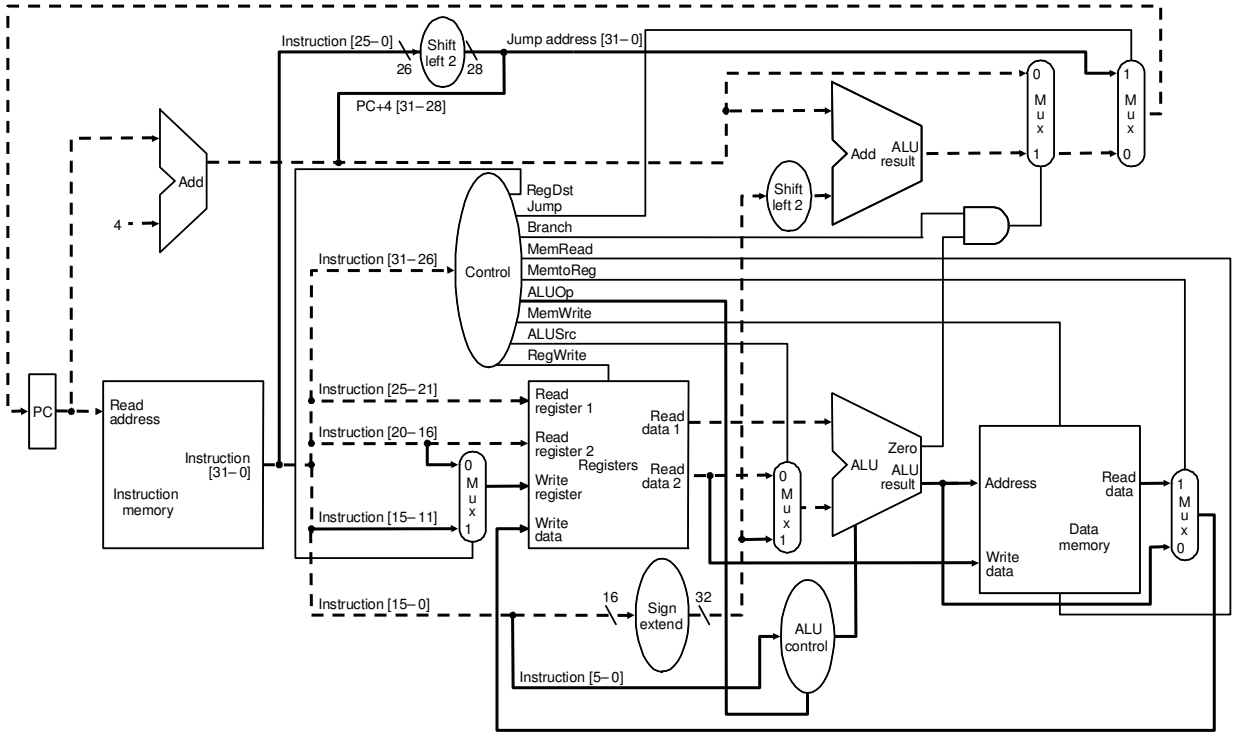
RegDst: x, Jump: 0, Branch: 0, MemRead: 0, MemtoReg: x, MemWrite: 1, ALUSrc: 1, RegWrite: 0, ALUOp: 00 (Adressberechnung)



d) Bedingte Verzweigung (beq)

Lösung:

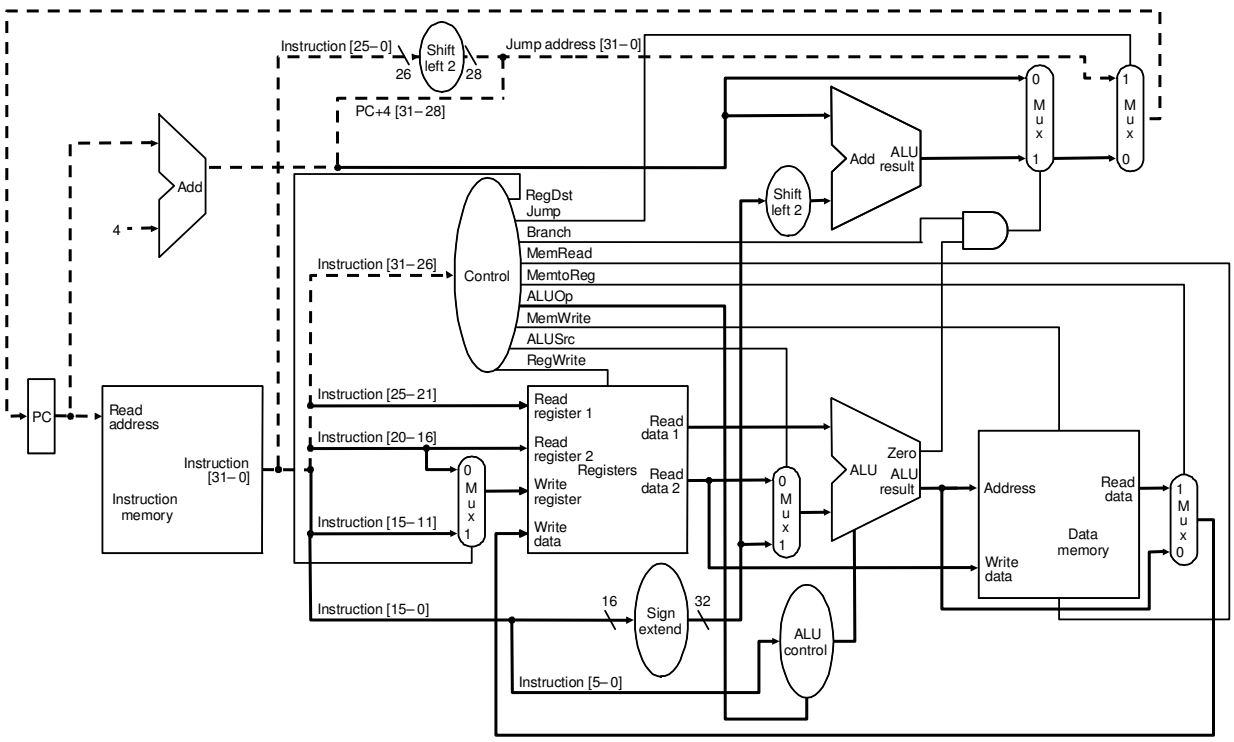
RegDst: x, Jump: 0, Branch: 1, MemRead: 0, MemtoReg: x, MemWrite: 0, ALUSrc: 0, RegWrite: 0, ALUOp: 01 (Subtraktion, um Gleichheit zu testen)



e) Jump

Lösung:

RegDst: x, Jump: 1, Branch: x, MemRead: 0, MemtoReg: x, MemWrite: 0, ALUSrc: x, RegWrite: 0, ALUOp: xx (Don’t care, beliebige Operation)



Aufgabe 2: MIPS-Signallaufzeiten

Für die Befehlstypen

a) R-Format

b) Load

c) Jump

aus Aufgabe 1 soll jeweils der kritische Pfad T in Formeldarstellung ermittelt werden, analog zu den Vorlesungsfolien Kapitel 8. Verwenden Sie dazu die im folgenden Absatz benannten Zeiten:

Bei einzelnen Registern, z.B. `PC`, ist der Wert am Ausgang erst nach einer Zeit t_{cto} (für $t_{Clock\,to\,Output}$) stabil. Andererseits muss ein neuer Wert vor einem Register mindestens t_{setup} vorher anliegen, um sicher den korrekten Wert speichern zu können. Der Datenspeicher wird innerhalb von $t_{readdata}$ gelesen und $t_{writedata}$ beschrieben, die Register innerhalb von $t_{regread}$ bzw. $t_{regwrite}$, der Befehlsspeicher innerhalb von t_{Pmem} (schreiben nicht möglich). Die Durchlaufzeit bei einer ALU soll mit t_{ALU} bezeichnet werden (für das Inkrementieren um 4 jedoch mit t_{plus4}), die für einen Multiplexer mit t_{mux}, die eines Und-Gatters mit t_{and} und die Steuersignalgenerierung Control mit t_{ctrl}.

Lösung:

Erläuterung: t_{ctrl} ist in verschiedenen Abschnitten der Berechnung notwendig und ist daher mehrfach aufgeführt – im Gegensatz zu den anderen Operationen, die jeweils nur einmal in der Reihenfolge der Benutzung addiert werden. Parallel laufende Operationen, deren Ergebnis Einfluss auf die Folgeberechnung hat, werden über eine `max`-Funktion zusammengeführt. Die Zeit des Multiplexers vor dem Eingang „Write Register" ist nicht dargestellt, da im parallel laufenden Zeitabschnitt ebenfalls Multiplexer verwendet werden, so dass seine Durchlaufzeit dahinter versteckt

wird.

a) R-Format

$T = t_{cto} + max(max(t_{plus4}, t_{Pmem} + t_{ctrl} + t_{and}) + 2 * t_{mux} + t_{setup}, t_{Pmem} + max(t_{regread}, t_{ctrl} + t_{mux}) + t_{ALU} + t_{mux} + t_{regwrite})$

$\approx t_{cto} + max(t_{plus4}, t_{Pmem}) + max(t_{ctrl} + t_{and} + 2 * t_{mux} + t_{setup}, max(t_{regread}, t_{ctrl} + t_{mux}) + t_{ALU} + t_{mux} + t_{regwrite})$

b) Load

$T = t_{cto} + max(t_{plus4}, t_{Pmem}) + max(t_{ctrl} + t_{and} + 2 * t_{mux} + t_{setup}, max(t_{regread}, t_{ctrl} + t_{mux}) + t_{ALU} + t_{readdata} + t_{mux} + t_{regwrite})$

c) Jump

$T = t_{cto} + max(t_{plus4}, t_{Pmem}) + t_{ctrl} + t_{mux} + t_{setup}$

Aufgabe 3: ALU-Kontrollogik

Implementieren Sie die ALU Kontrollogik von Kapitel 8, Folie 11 der Vorlesung in Verilog.

Lösung:

```
module alu_control(F, ALUOp, Operation);

  input  [5:0] F;
  input  [1:0] ALUOp;
  output [2:0] Operation;

  wire OR_F0F3 = F[0] | F[3];
  wire AND_F1_ALUOP1 = F[1] & ALUOp[1];
  assign Operation[0] = OR_F0F3 & ALUOp[1];
  assign Operation[1] = ~ALUOp[1] | ~F[2];
  assign Operation[2] = ALUOp[0] | AND_F1_ALUOP1;

endmodule
```

Aufgabe 4: Evaluation

Füllen Sie bitte den in der Vorlesung verteilten Evaluationsfragebogen aus und geben Sie diesen ab *oder* beantworten Sie die Fragen online unter

`http://www.D120.de/feedback/fragebogen.php`.