Kap.3
Mikroarchitektur

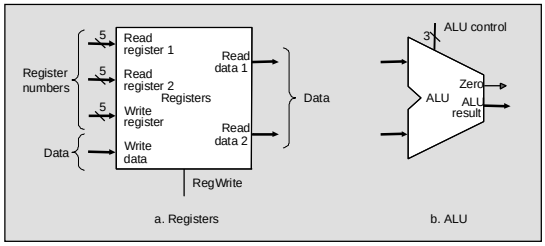
Prozessoren, interne Sicht

- **3.1** Elementare Datentypen, Operationen und ihre Realisierung (siehe 2.1)
- **3.2** Mikroprogrammierung
- **3.3** Einfache Implementierung von MIPS
- **3.4** Pipelining

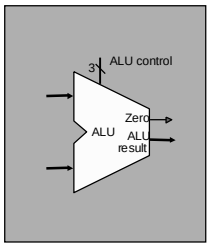
Betrachte der Übersichtlichkeit nur eine Untermenge der MIPS-Sprache:

- Load/Store-Befehle: lw, sw
- Arithmetisch-logische Befehle: add, sub, and, or, slt (auch mit immediate)
- Sprungbefehle: beq, j

Annahme:
die benötigte ALU und Registerbank sind schon implementiert



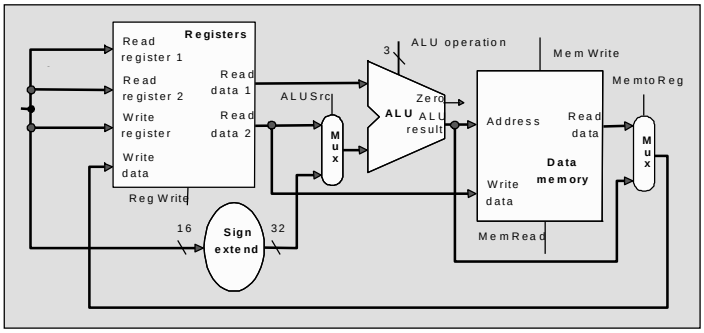
Steuerung der ALU



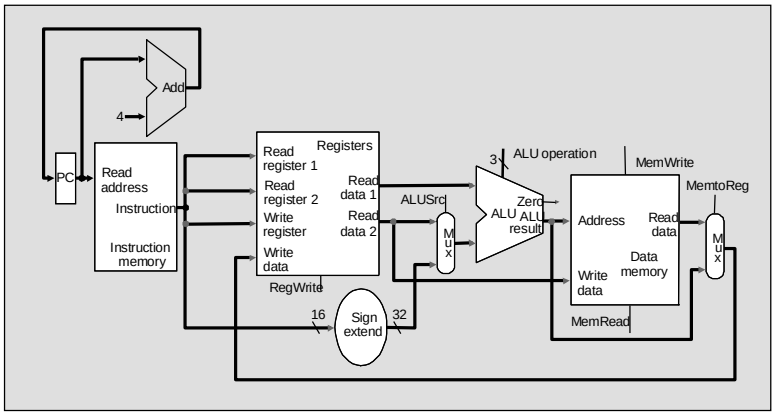
Die ALU kann fünf verschiedene Operationen ausführen, benötigt also 3 Steuerleitungen (**ALUcontrol**)

ALUcontrol	*Operation*
000	AND
001	OR
010	add
110	substract
111	set on less than

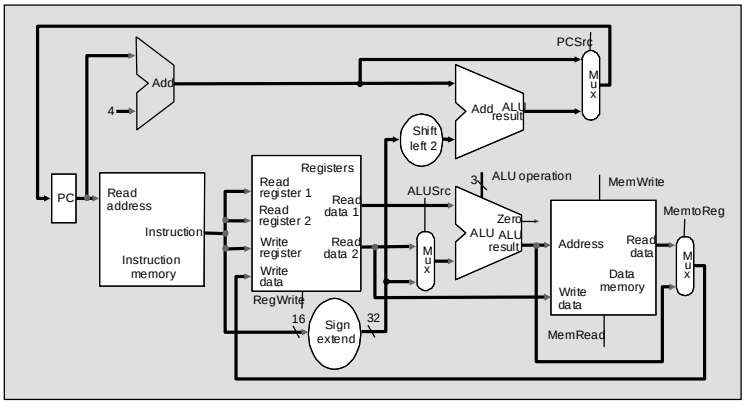
Datenpfad ohne Instruction fetch



Datenpfad mit Inkrementieren von $pc



Datenpfad mit Branch-Logik

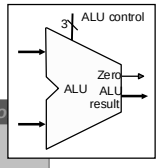


Rolle der ALU im Prozessor

ALU wird nicht nur im Rahmen der arithmetisch-logischen Befehle eingesetzt

- lw Rdest, imm_{16}(Rscr)
  - ➨ Addition zur Berechnung der Adresse
- sw Rsrc1, imm_{16}(Rsrc2)
  - ➨ Addition zur Berechnung der Adresse
- beq Rsrc1, Rsrc2, label
  - ➨ Substraktion zur Berechnung des Vergleiches

	Größe [bit]	6	5	5	5	5	6
Arithmetische Befehle	Format R	op	rs	rt	rd	shamt	funct
Immediate + LOAD/STORE	Format I	op	rs	rt	adr-teil		
Bedingte Sprungbefehle	Format J	op	Sprungadresse				



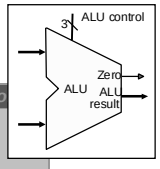
ALUcontrol	Operation
000	AND
001	OR
010	add
110	substract
111	set on less than

Bei welchen Instruktionen müssen nun welche Steuersignale gesetzt werden ?

ALUOp	Instruction opcode	Instruction operation	Funct field	desired ALU action	ALU control
00	lw	load word	xxxxxx	add	010
	sw	store word	xxxxxx	add	010
01	beq	branch equal	xxxxxx	substract	110
10	R-type	add	100000	add	010
	R-type	substract	100010	substract	110
	R-type	AND	100100	and	000
	R-type	OR	100101	or	001
	R-type	set on less than	101010	set less than	111

Ansteuerung der ALU ff

Die Belegung der Steuerleitungen *ALUcontrol* hängen somit nur von *ALUOp* und *Funct* ab.



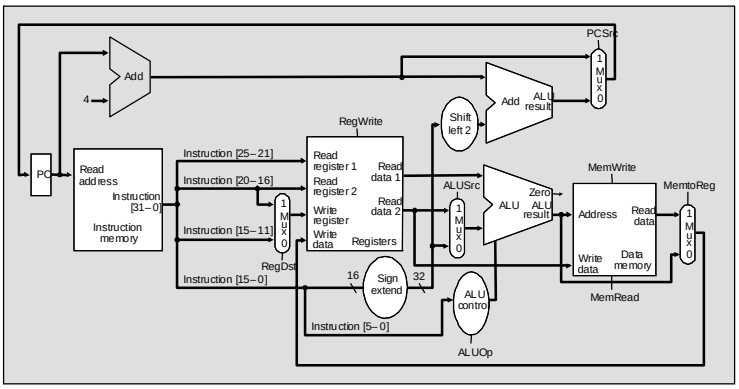
ALUcontrol	Operation
000	AND
001	OR
010	add
110	substract
111	set on less than

ALUOp	Funct field	ALU control
00	xxxxxx	010
00	xxxxxx	010
01	xxxxxx	110
10	100000	010
10	100010	110
10	100100	000
10	100101	001
10	101010	111

ALUOp hängt nur vom Instruktionscode ab

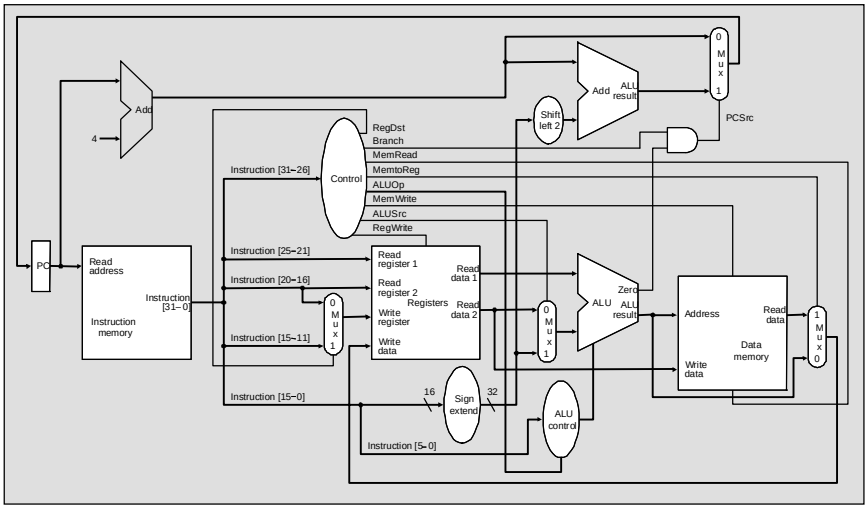
Instruction OPcode	ALUOp control
100011	00
101011	00
000100	01
000000	10

Resultierender Datenpfad



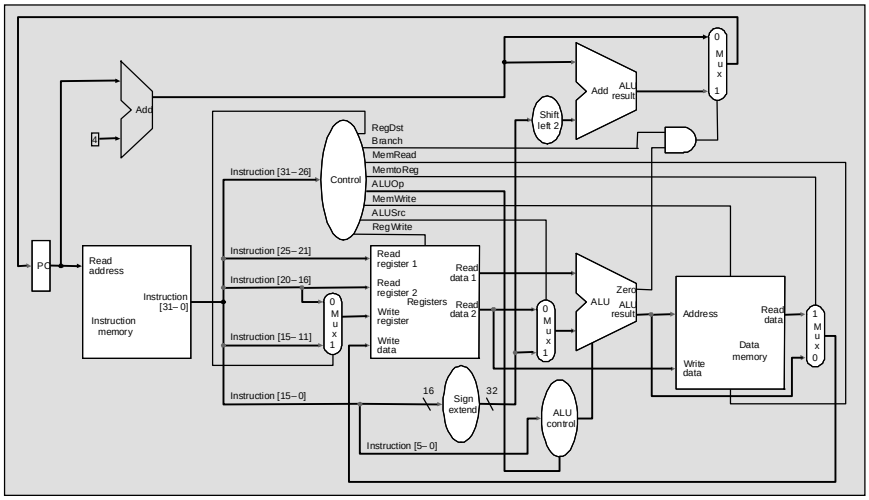
Generierung der Steuersignale

Damit der Datenpfad arbeitet, muss eine Steuereinheit die Steuersignale des Datenpfades in Abhängigkeit der anliegenden Instruktion richtig setzen !



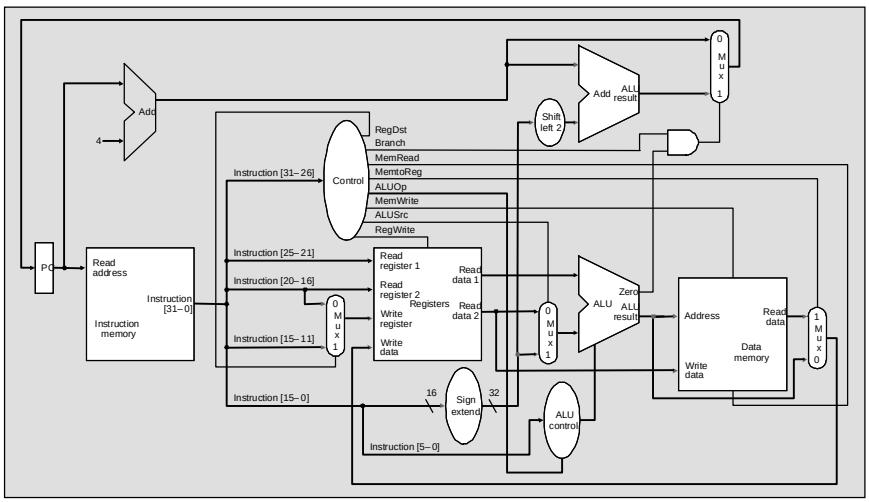
Erster Schritt einer R-type Instruktion

- Holen der Maschineninstruktion
- Erhöhung von $pc um 4



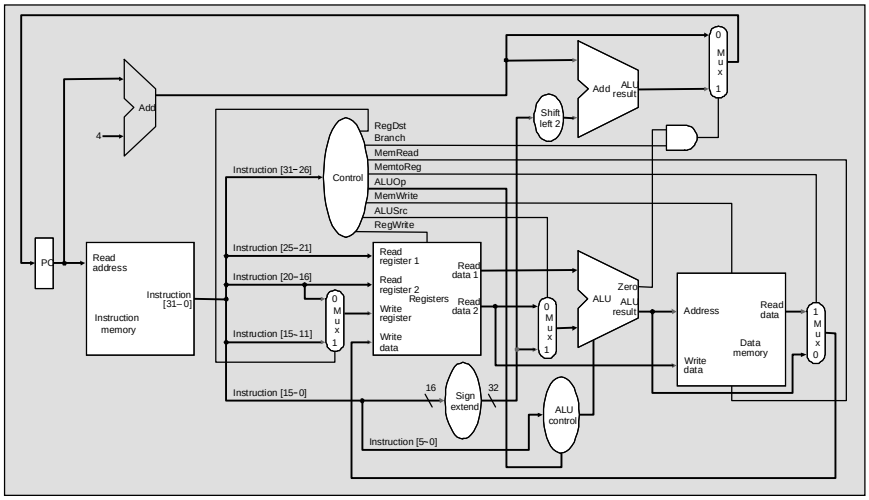
Zweiter Schritt einer R-type Instruktion

- Instruktions-Opcode zur Steuereinheit und Setzen der Steuerleitungen
- Lesen der beiden Register 1 und 2



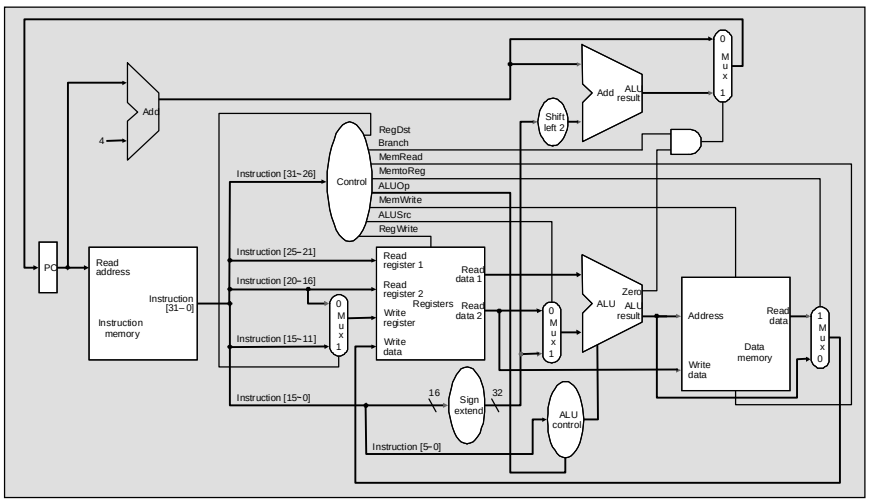
Dritter Schritt einer R-type Instruktion

- Berechnung der ALUcontrol Steuersignale
- Berechnung des Ergebnisses durch die ALU



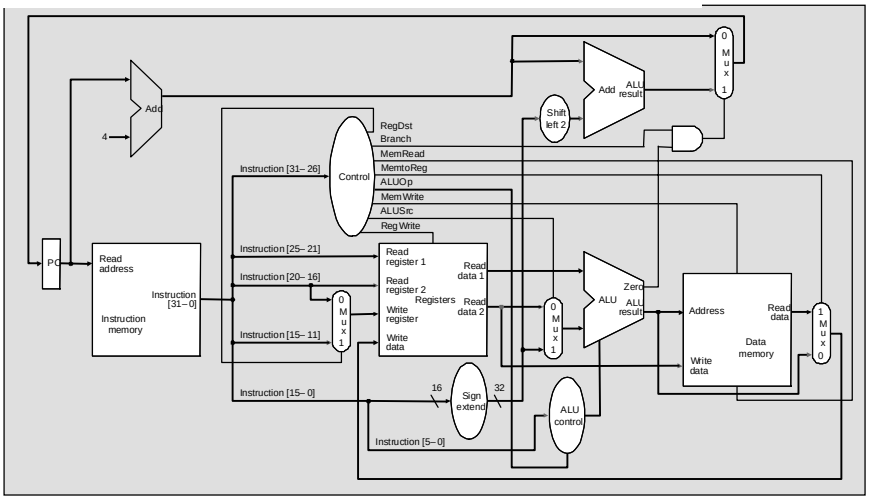
Letzter Schritt einer R-type Instruktion

- Zurückschreiben des Ergebnisses in die Registerbank
- Setzen von $pc auf seinen nächsten Wert pc_{old}$+4

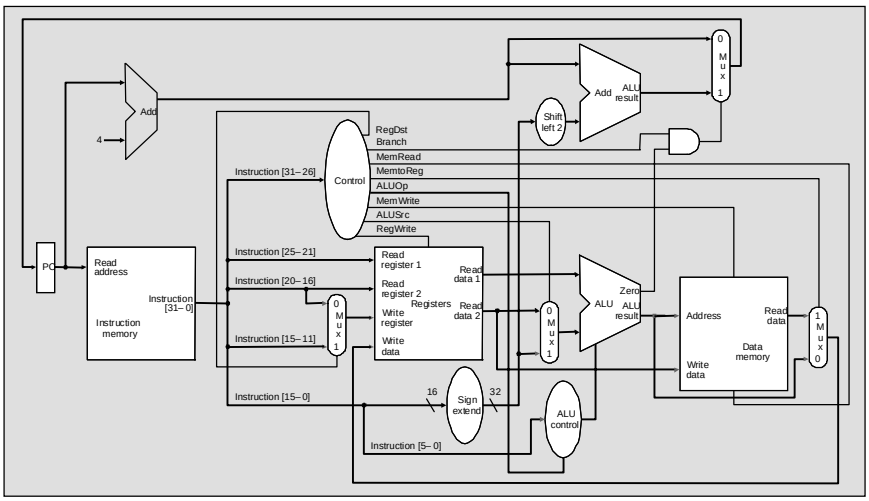


	Größe [bit]	6	5	5	5	5	6
Arithmetische Befehle	Format R	op	rs	rt	rd	shamt	funct
Immediate + LOAD/STORE	Format I	op	rs	rt	adr-teil		
Bedingte Sprungbefehle	Format J	op	Sprungadresse				

ad-

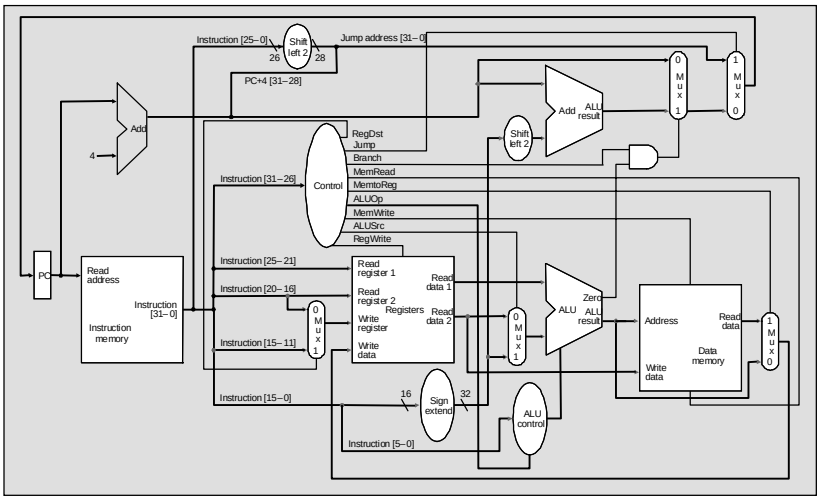


Datenpfad-Operationen bei Branch Equal

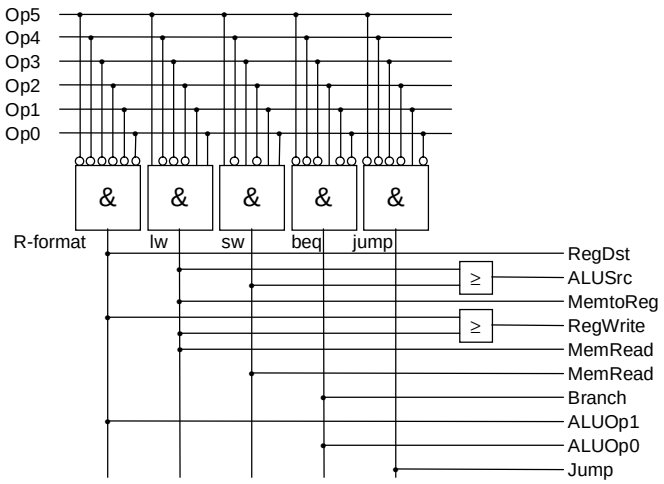


Erweiterung des Datenpfades für J-Instruktion

- In MIPS ergibt sich die Sprungadresse beim Befehl j target durch {($pc+4)[31:28] , instruction[25:0], 2`b00}



HW für die Steuerung



Probleme

- lw benötigt 5 Funktionale Einheiten
  - ➔ lange Gatterlaufzeit
  - ➔ große Taktperiode
  - ➔ alle Instruktionen werden langsamer
- Effekt verschlimmert sich bei noch komplexeren Befehlen (FP-instructions)

➔ Mehrtakt Implementierung

- ➔ Controller wird durch FSM implementiert

FSM-Steuerung