Rechnerarchitekturen 1*

Einführung

Prof. Dr. Alexander Auch

*Teilweise entnommen aus “Mikrocomputercomputertechnik 1” von Prof.Dr-Ing. Ralf Stiehler, sowie Patterson & Hennessy

Ziele der Veranstaltung

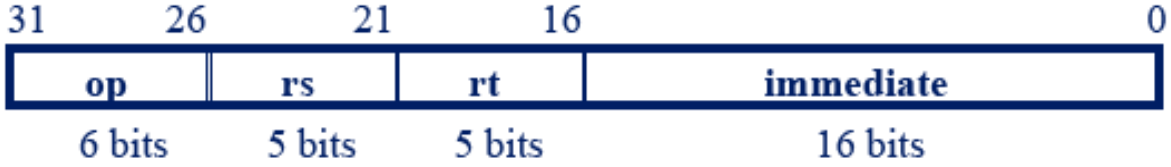
- **Rechnerentwurf**:
  - Prozessor, Speicher, Ein-/Ausgabe
  - Entwurfs- und Optimierungsmöglichkeiten
- **Prozessorentwurf**:
  - Befehlsverarbeitung
  - Entwurfs- und Optimierungsmöglichkeiten
- **Assemblerprogrammierung**:
  - im MIPS-Simulator MARS

MIPS-Light : Wir bauen einen Computer

⇒ **„Großes Ziel“ ist der Aufbau eines Prozessors MIPS-Light, der die folgende Teilmenge an MIPS-Instruktionen verarbeiten kann.**

- `add, sub, and, or, slt`

- `lw, sw, beq`



- `j`

⇒ **Erster Schritt ist der Aufbau einer einfachen ALU (Arithmetic Logical Unit)**

Die CPU - Datenpfad und Steuerwerk

In diesem Kapitel geht es um einen Teil innerhalb der CPU, den Datenpfad. Dieser transferiert und bearbeitet Daten. Zum Datenpfad gehören die Register der Befehlsarchitektur (MIPS-ISA-Register) und weitere Register.

Datenpfad für MIPS-Light

Im Rahmen der Vorlesung werden wir einen Datenpfad für eine Teilmenge des MIPS-Instruktionssatzes (*MIPS-light*) konstruieren.

⇒ Arithmetisch/logische Operationen : add, sub, and, or, slt, nor
⇒ Speicherzugriffsoperationen : lw, sw
⇒ Verzweigungsoperationen : beq
⇒ Sprungbefehl : j

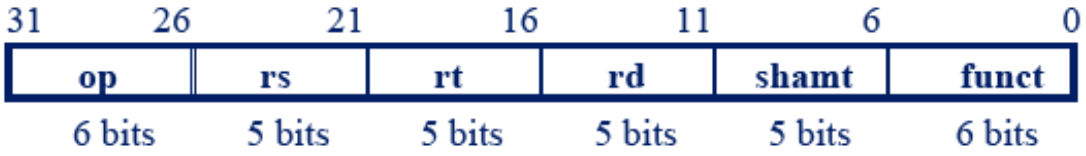
Steuerwerk für MIPS-Light

Das Steuerwerk (Control) steuert die Elemente des Datenpfads. Sie wird im Rahmen der Vorlesung rein kombinatorisch (single cycle) bzw. als Zustandsdiagramm (multi cycle) entworfen.

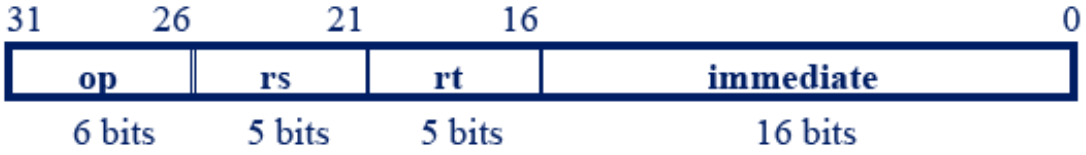
Evtl. später Realisierung als VHDL-Code in einen FPGA……

Rückblick : MIPS-Befehlssatz

- add, sub, and, or, slt



- lw, sw, beq



- j

⇒ op: Opcode des Befehls
⇒ rs, rt, rd: Nummer von Source- und Destination-Register
⇒ shamt: shiftamount
⇒ funct: Auswahl einer Variante des op-Feldes
⇒ address / immediate: Adress-Offset oder Immediate-Wert
⇒ target address: Zieladresse des Jump-Befehls

Grundlegende Schritte der Befehlsausführung (NEUMANN-ZYKLUS)

1. **IF Instruction Fetch**: Adressieren des Instruktionsspeichers mit dem Wert des Befehlszählers und Laden der nächsten Instruktion

2. **ID Instruction decode, Register fetch**: Lesen eines (z.B. bei I-Typ) oder zweier (z.B. bei R-Typ) Register, um Operanden bereit zu stellen

3. **EX Execute, Memory Address Computation, Branch Completion**: Operationsausführung oder Adressberechnung unter Verwendung der ALU

4. **MEM Memory Access, R-Type Instruction Completion**
   - Speicherzugriff (bei lw, sw)
   - ALU-Ausgang in Register schreiben (bei arithm./log. Instruktionen)
   - Neuen Wert in den Befehlszähler schreiben (branch/jump)

5. **WB Memory Read Completion** : (bei lw) Memorywert in Register schreiben

Mikrocomputertechnik 1 – Single Cycle Datenpfad und Steuerwerk

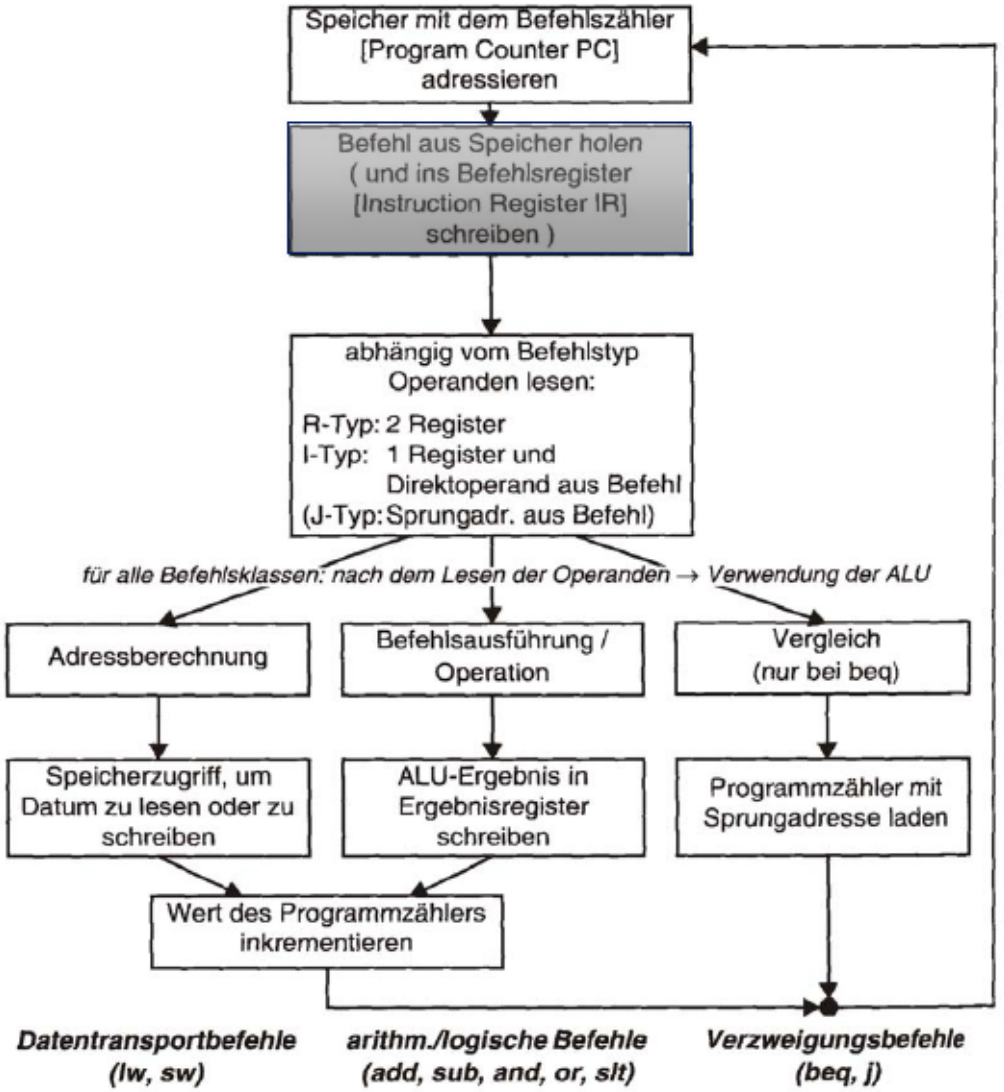
1. **IF Instruction Fetch**

2. **ID Instruction decode, Register fetch**

3. **EX Execute, Memory Address Computation, Branch Completion**

4. **MEM Memory Access R-Type Instruction Completion**

5. **WB Memory Read Completion**



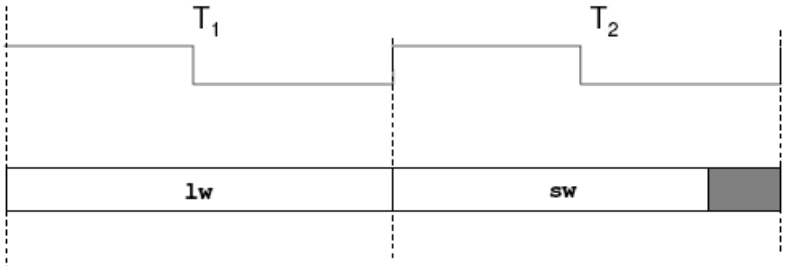
Cycle und Multi Cycle CPU

Die Leistungsfähigkeit eines Computers hängt u.a. ab von

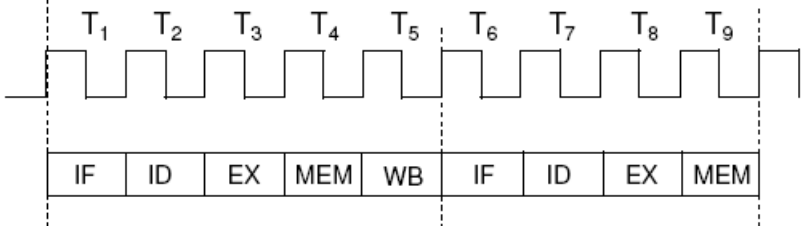
⇒Takt-Zykluszeit (Taktfrequenz)
⇒ Anzahl der Taktzyklen pro Befehl (*CPI: clock cycles per instruction*)

Man unterscheidet
⇒ Single Cycle CPU (jeder Befehl ein Taktzyklus) und
⇒ Multi Cycle CPU (mehrere Taktzyklen pro Befehl bei höherer Taktfrequenz)
⇒ CPU mit Pipeline-Strukturen

Single Cycle CPU =>



Multi Cycle CPU =>



Übersicht zu Single Cycle und Multi Cycle CPU

Single Cycle

⇒ jede Instruktion wird in einem (langen) Taktzyklus ausgeführt
⇒ während eines Instruktionszyklus kann ein Datenpfadsegment nur einmal benutzt werden => aufwändigere Hardware !
⇒ verschiedene Instruktionsklassen sollen dieselben Datenpfadelemente nutzen, hierzu bedarf es mehrerer Multiplexer
⇒ sehr einfache Steuerung (rein kombinatorisch)
⇒ der am längsten dauernde Befehl (Load Word) definiert die Taktperiode

Multi Cycle

⇒ Aufteilung der Instruktionsabarbeitung in mehrere Teilschritte
⇒ Jeder Teilschritt benötigt einen Taktzyklus
⇒ Taktperiode kann kürzer werden
⇒ je nach Instruktion kann die Anzahl der Schritte unterschiedlich sein
⇒ Datenpfadelemente können mehrfach verwendet werden => Hardwarereduktion !
⇒ zusätzliche Register zur Speicherung der Signale zwischen den Taktschritten nötig
⇒ Steuerung ist komplexer im Vergleich zum Single Cycle

Bausteine des Datenpfads

Die Analyse des Befehlssatzes legt die Verwendung folgender Hardware nahe :

⇒ externes Memory / Speicher (Befehle & Daten)

⇒ 32 x 32 Registersatz (register file) mit 32 Wörtern à 32 Bit

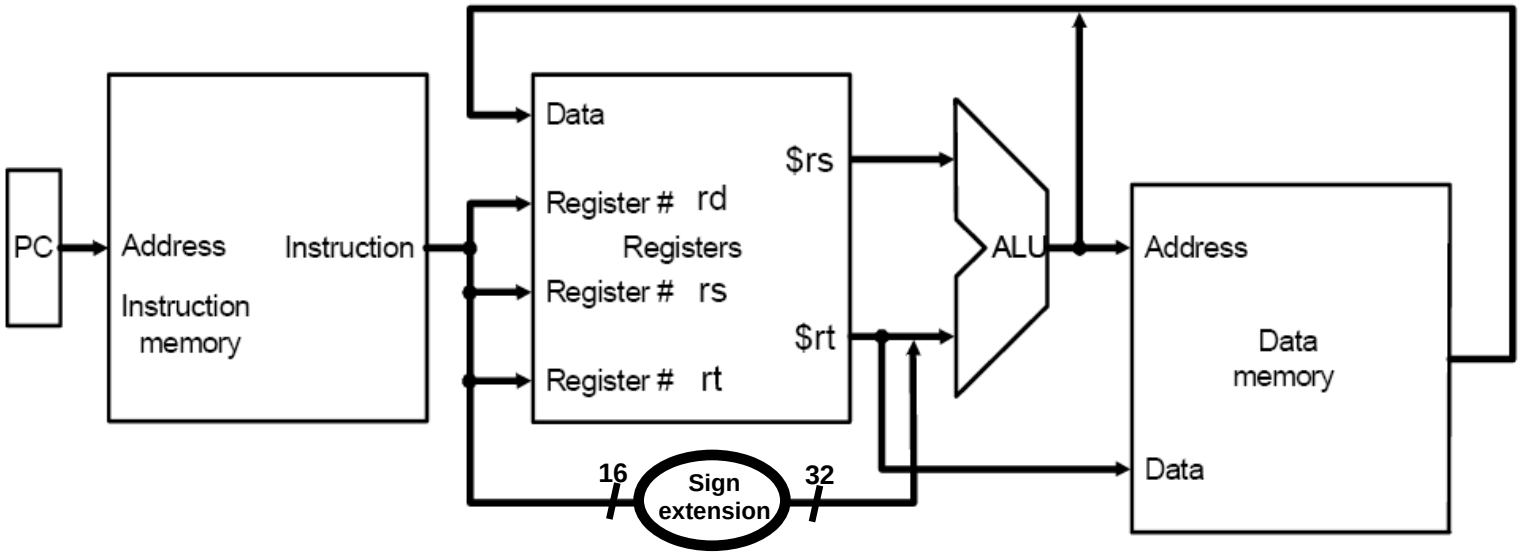
→ 5-Bit-Adresseingang für RS : Lesen von rs
→ 5-Bit-Adresseingang für RT : Lesen oder Schreiben von rt
→ 5-Bit-Adresseingang für RD : Schreiben von rd

⇒ Befehlszähler (Program counter) PC (ebenfalls ein Register)

⇒ 16-32 Bit-Extender (für Sign-Extension)

⇒ ALU (für add, sub, and, or, nor)

Datenpfad – Ein verläufiger grober Entwurf



⇒ Aus den genannten Schritten der Befehlsausführung ergibt sich diese erste vereinfachte Darstellung des Datenpfads.

⇒ Was fehlt ?

→ weitere Funktionseinheiten, insbesondere Steuerwerk und Kontrollleitungen

⇒ Die Schaltung wird im weiteren Verlauf der Vorlesung erweitert und detaillierter ausgeführt.

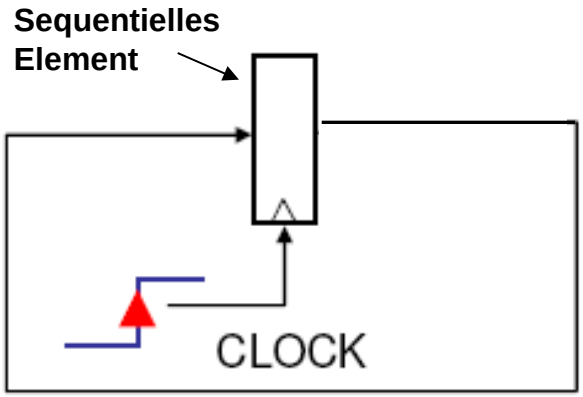
Kombinatorische und sequentielle Logik im Datenpfad

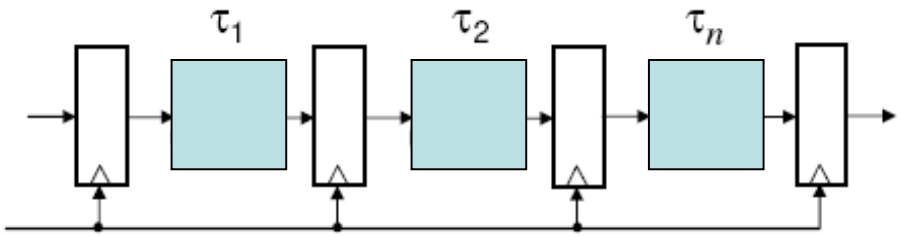
⇒ Ein sequentielles Element wird mit der steigenden oder fallenden Flanke eines systemweiten Taktsignals (Clock) gesteuert.

⇒ Sämtliche Register sind flankengesteuert und können in einer Taktperiode nur 1x (z.B. mit der steigender Flanke) beschrieben werden

⇒ Rein kombinatorische Logik enthält keine Speicher, bei Änderung der Eingangssignale erreicht der Ausgang nach einem Delay einen dauerhaften Endzustand.

⇒ Die CPU enthält sowohl kombinatorische als auch sequentielle Elemente

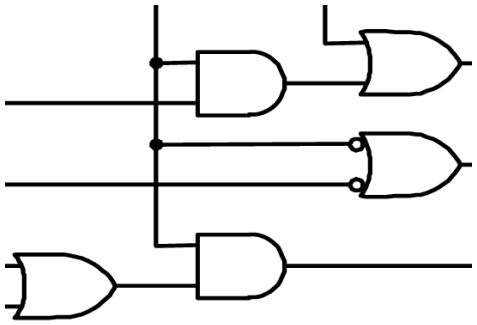


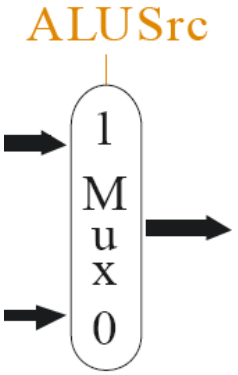


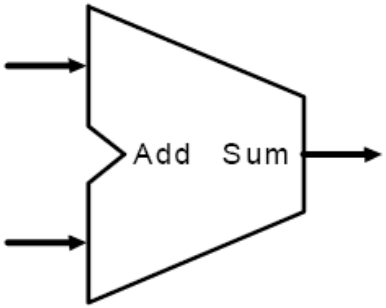
Die Taktperiode muss größer sein als das größte Delay in einem Kombinatorikpfad.

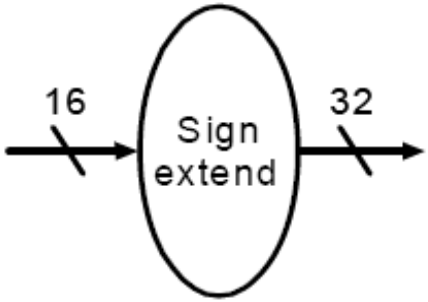
Rein kombinatorische Elemente des Datenpfads

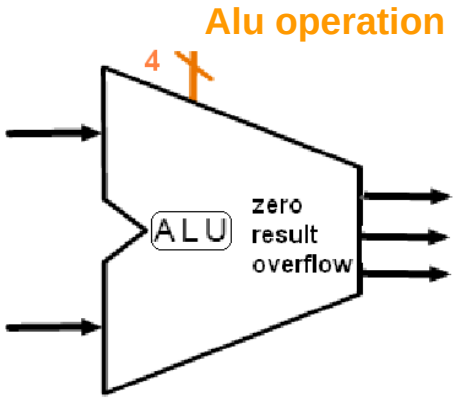
- ⇒ Einfache Gatter (AND, OR usw)
- ⇒ Multiplexer, Decoder
- ⇒ ALU
- ⇒ Sign Extender 16→ 32 Bit
- ⇒ „Mini-Alu“ (nur-Addierer), um 4 oder “extended Immediate” zum PC zu addieren







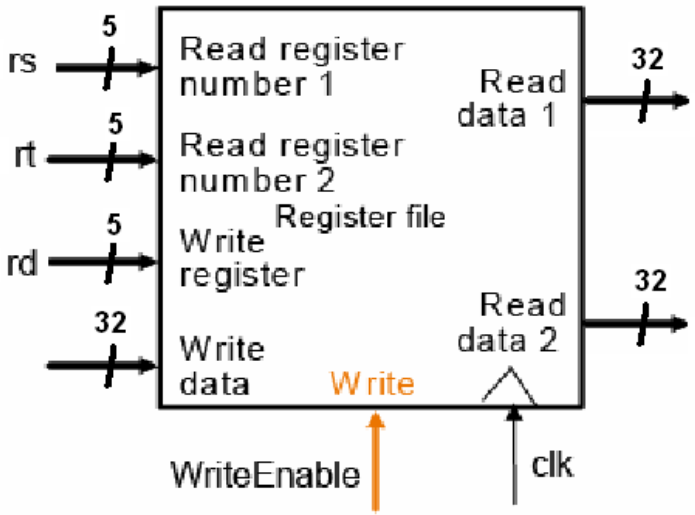




Sequentielle (speichernde) Elemente des Datenpfads

Der Registersatz bestehend aus 32 Registern

⇒ Ein 32-Bit Input-Bus „write data“
⇒ Zwei 32-Bit Output-Busse „read register data 1 bzw. 2“



Die Register werden selektiert durch:
⇒ rs: 5 Bit Adresse des ersten zu lesenden Registers
⇒ rt: 5 But Adresse des zweiten zu lesenden Registers
⇒ rd: 5-Bit Adresse des zu beschreibenden Registers

Lesen vom Register ist ein rein kombinatorischer Vorgang
⇒ wenn der 5-Bit Addresswert an rs oder rt angelegt wird, liegt nach einer gewissen sog. Zugriffszeit (Gatterdurchlaufzeiten) am entsprechenden 32-Bit Ausgang der Registerinhalt dort an (völlig unabhängig vom Clocksignal !)

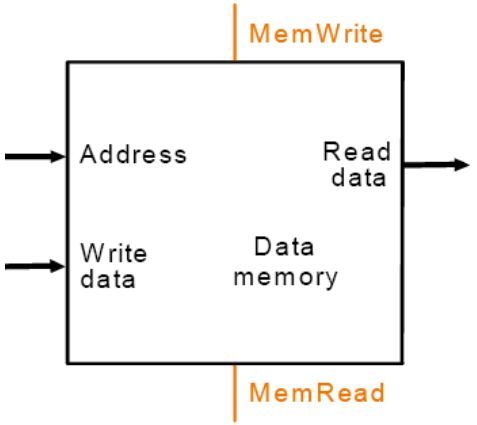
Schreiben in das Register geht mit dem Clock-Input (CLK)
⇒ Das am 32-Bit-Eingang „write data“ anliegende Wort wird genau dann ins Register geschrieben, wenn WriteEnable = 1 ist und die Clock eine steigende Flanke durchläuft
⇒ CLK wird ausschliesslich für Schreib-Operation benötigt

Sequentielle (speichernde) Elemente des Datenpfads

Externes Memory

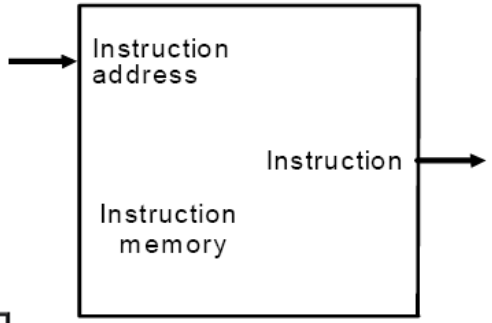
32-Bit-Adresse selektiert Wort im Speicher

⇒ bei MemWrite = 1 wird das 32-Bit-Wort am Eingang „Write data“ ins Memory geschrieben
⇒ bei MemRead = 1 wird das 32-Bit-Wort an den Ausgang „Read data“ gestellt



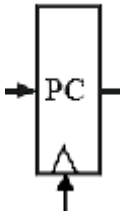
Befehlsspeicher

⇒ Adresse selektiert Wort im Speicher
⇒ 32-Befehls-Bit-Code liegt dann an „Instruction“ an
⇒ für Befehlsausführung quasi-kombinatorisch (da nur gelesen wird)



Program Counter

⇒ Zähler für die Adresse des Befehlsspeichers



Entwurf des Datenpfads

Datenpfad-Teilentwurf für INSTRUCTION FETCH

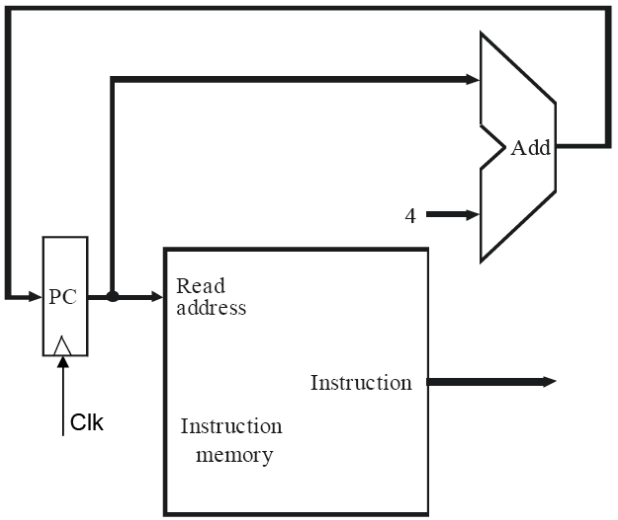
⇒ der erste Schritt „Befehl aus Befehlsspeicher holen“ ist für alle Befehle gleich

⇒ gleiche Hardware für R-Type, I-Type, J-Type

⇒ Erforderliche Hardware :

⇒ **Befehlsspeicher** : Speicher zur Ablage einer Befehlsfolge (Programm)

⇒ Register zur Adressierung des aktuellen Befehls im Befehlsspeicher (Program Counter: **PC**)



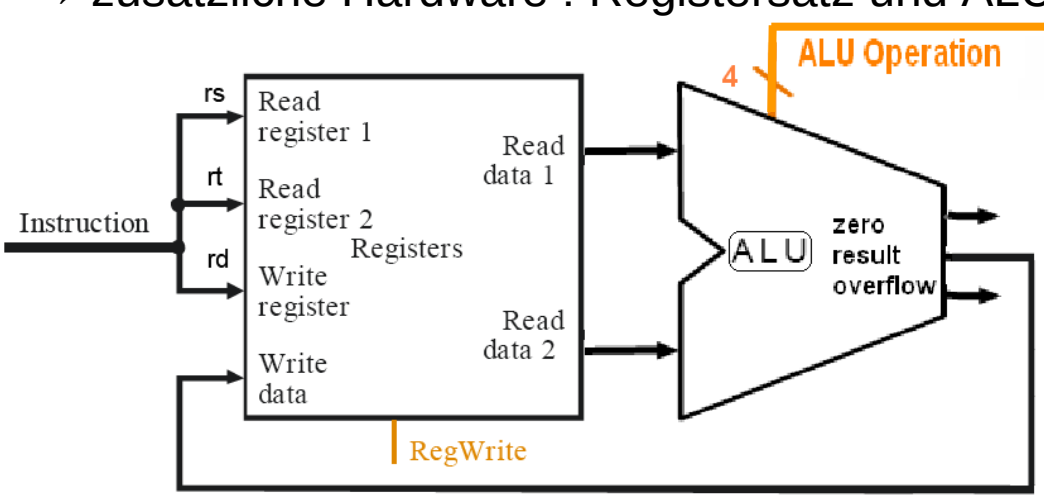
⇒ Mini-Alu = **Addierer** zur Berechnung der nächsten Instruktionsadresse

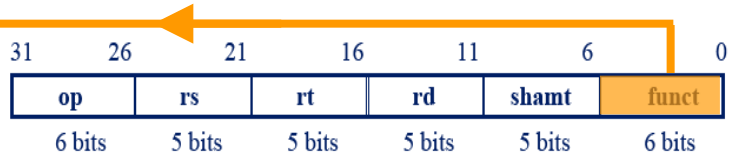
Entwurf des Datenpfads

Datenpfad-Teilentwurf für R-Type Instruktionen

→ Alle R-Type Instruktionen haben die Struktur **R[rd] ← R[rs] op R[rt]**

→ zusätzliche Hardware : Registersatz und ALU





→ **ALU operation** und **RegWrite** sind Steuersignale, die vom Steuerwerk gesetzt werden müssen (→ später...)

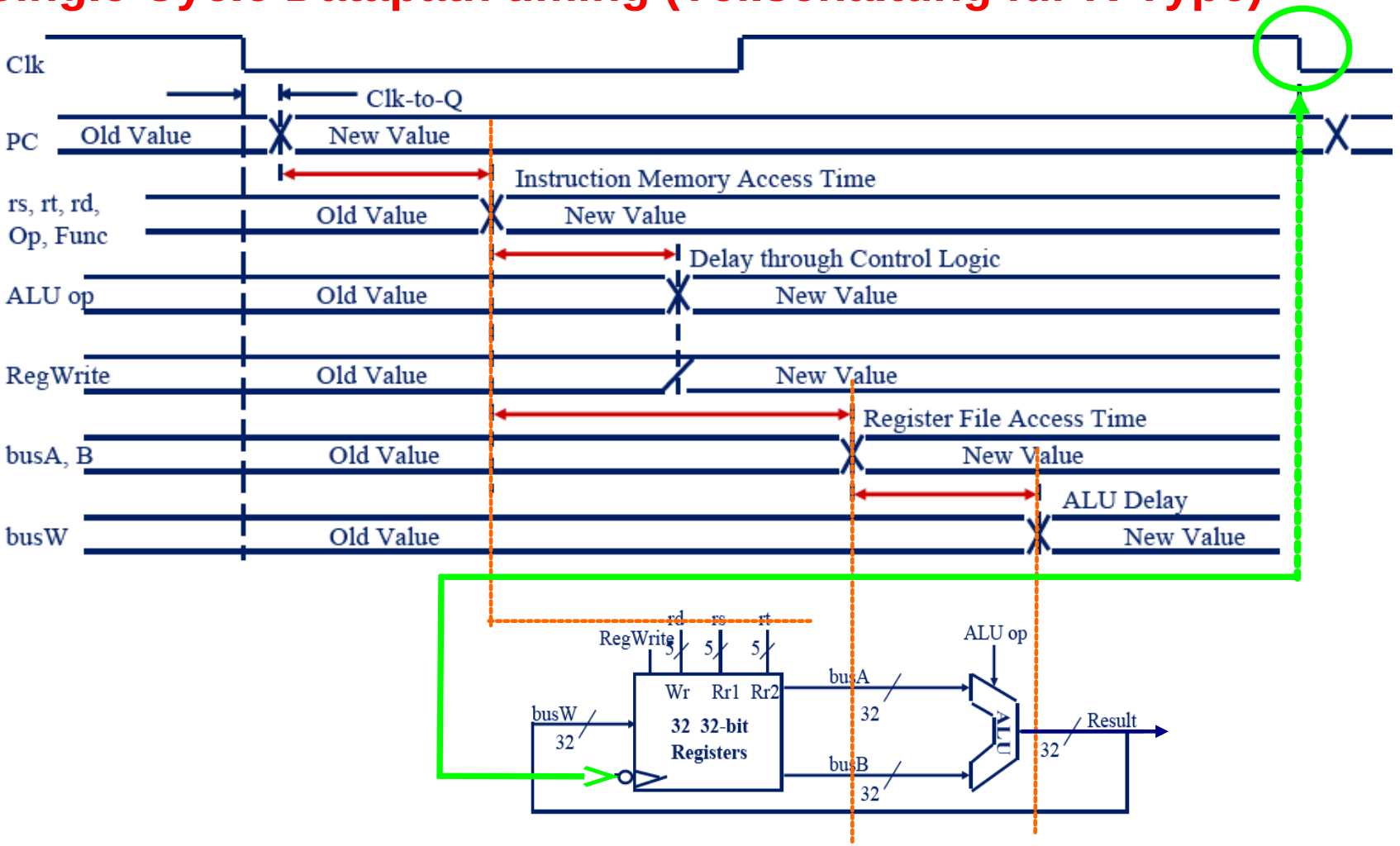
→ die Operandenfelder des Befehls (rs, rt, rd) adressieren den Registersatz

→ die 2 selektierten Registeradressen sind Eingangssignale für die ALU

→ das funct-Feld des Befehls bestimmt ALU-Operation
(OpCode bei R-Type = 0, Aufgabe der ALU wird nur durch funct bestimmt).

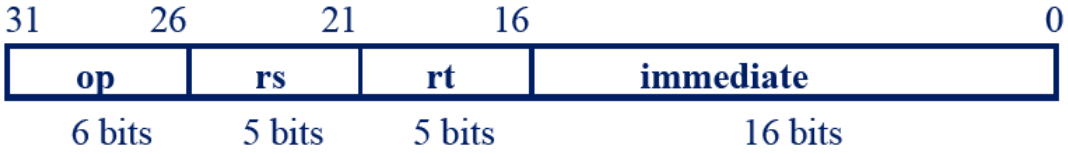
→ ALU-Ergebnis wird ins Registerfile zurückgeschrieben

Single Cycle Datapath timing (Teilschaltung für R-Type)



Entwurf des Datenpfads

Datenpfad-Teilentwurf für Speicherzugriffe (load/store word)



→ lw Struktur **R[rt] ← M[rs] + SignExt[immediate]**

→ sw Struktur **M[R[rs] + SignExt[immediate] ← R[rt]**

→ ein 32-Bit-Operand im registeradressierten Memory,
ein 16 Bit-Direktoperand im Befehl

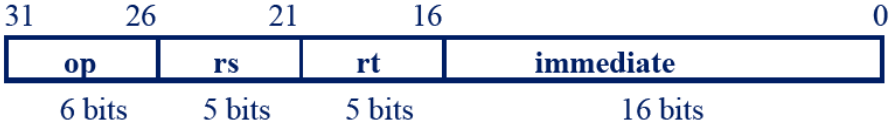
→ Notwendige Hardware : Memory und Sign Extender 16→ 32 Bit

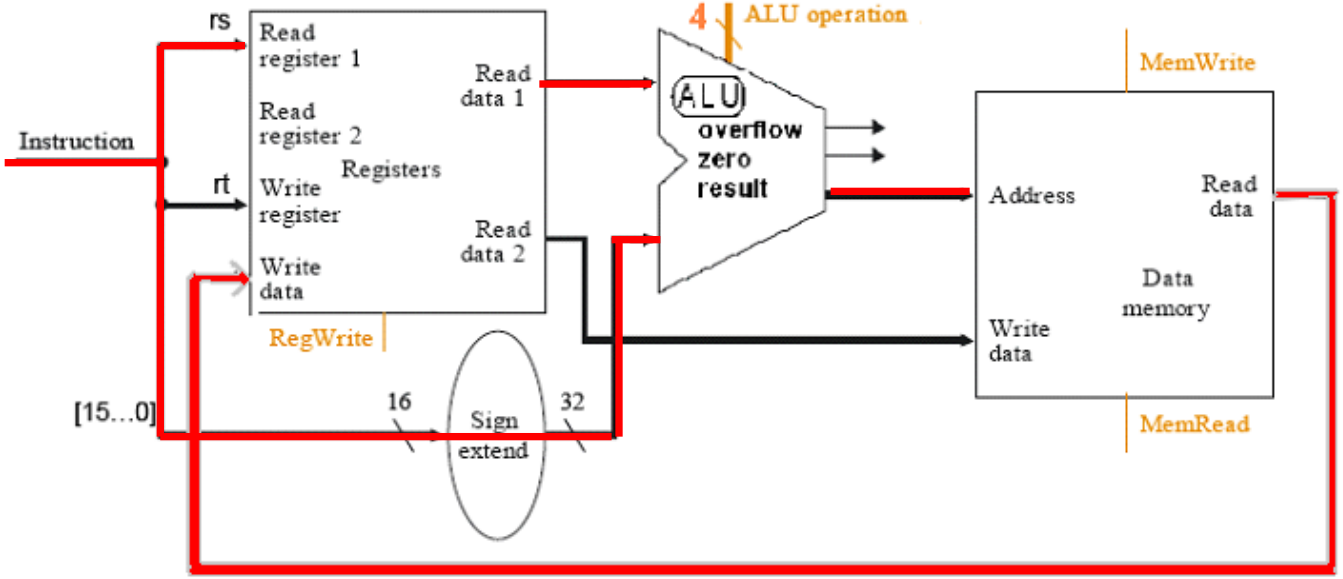
→ beachte : rd fehlt, Zielregister ist rt !!

Entwurf des Datenpfads

Datenpfad-Teilentwurf für Speicherzugriffe (load word)

→ lw Struktur **R[rt] ← M[rs] + SignExt[immediate]**

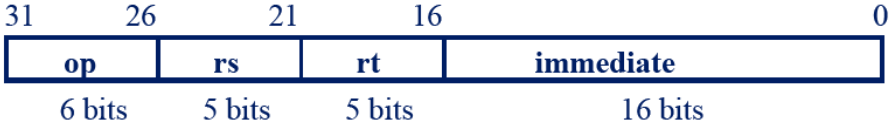


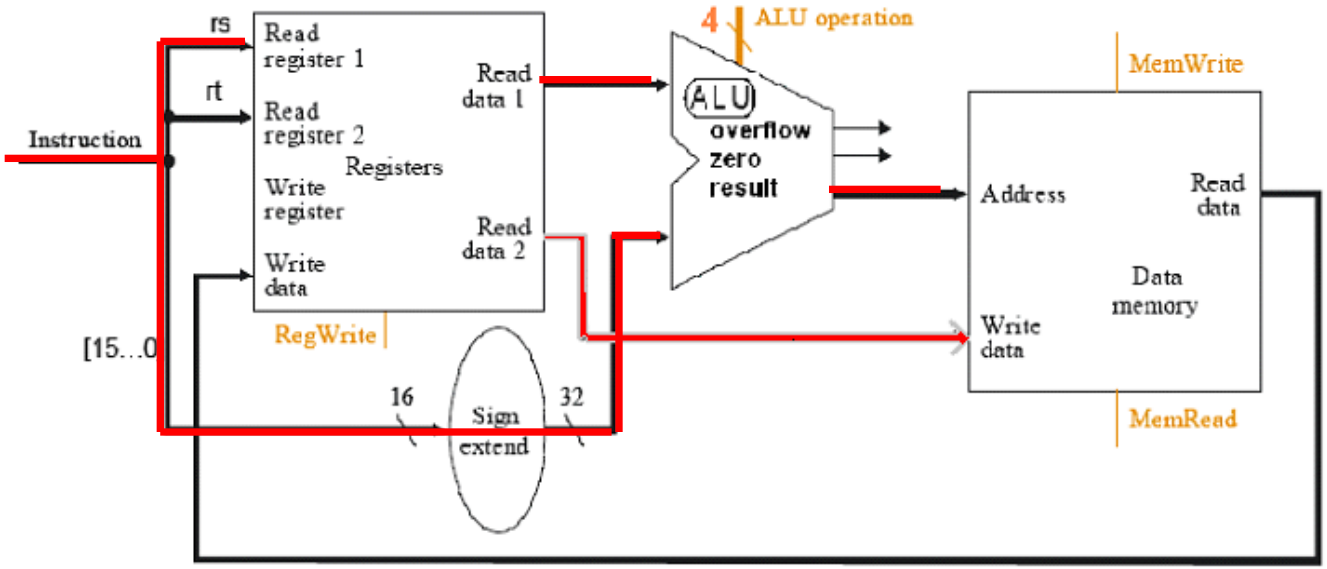


Entwurf des Datenpfads

Datenpfad-Teilentwurf für Speicherzugriffe (store word)

→ sw Struktur **M[R[rs] + SignExt[immediate] ← R[rt]**





Entwurf des Datenpfads

Datenpfad-Teilentwurf für beq, bne $t1, $t2, Offset

⇒ Vergleich zweier Registerinhalte durch Subtraktion und Auswertung des Zero-Flags der ALU

⇒ Je nach Befehl (beq,bne) und Zeroflag-Resultat bleibt der Befehlszähler unverändert oder ein neuer Wert muss berechnet und in den Befehlszähler geladen. Das Steuerwerk entscheidet in Abhängigkeit vom Zeroflag, was zu geschehen hat.

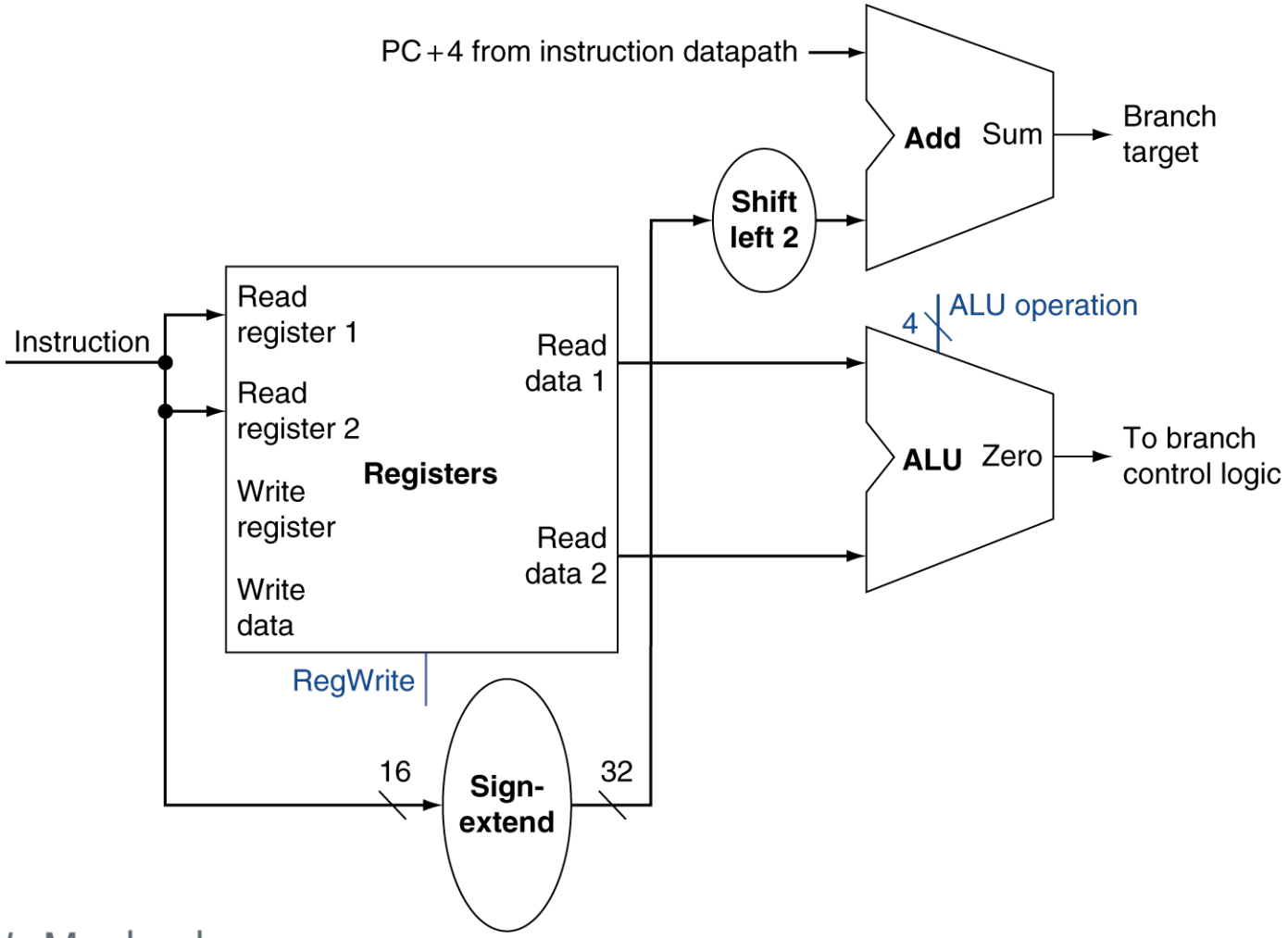
⇒ Der neue Wert des Befehlszählers basiert auf der Berechnung PC+4
=> wurde bereits berücksichtigt

⇒ Der in der Instruktion angegebene Offset beschreibt die Anzahl ganzer 32-Bit-Speicherworte. Da der Befehlszähler Bytes addressiert, ist der Offset um 2 Bits nach links zu shiften (*4).

⇒ für die Berechnung des neuen Befehlszählerstands wird ein weiterer Addierer benötigt, da die ALU bereits mit dem Vergleich beschäftigt ist.

Entwurf des Datenpfads

Datenpfad-Teilentwurf für beq, bne $t1, $t2, Offset

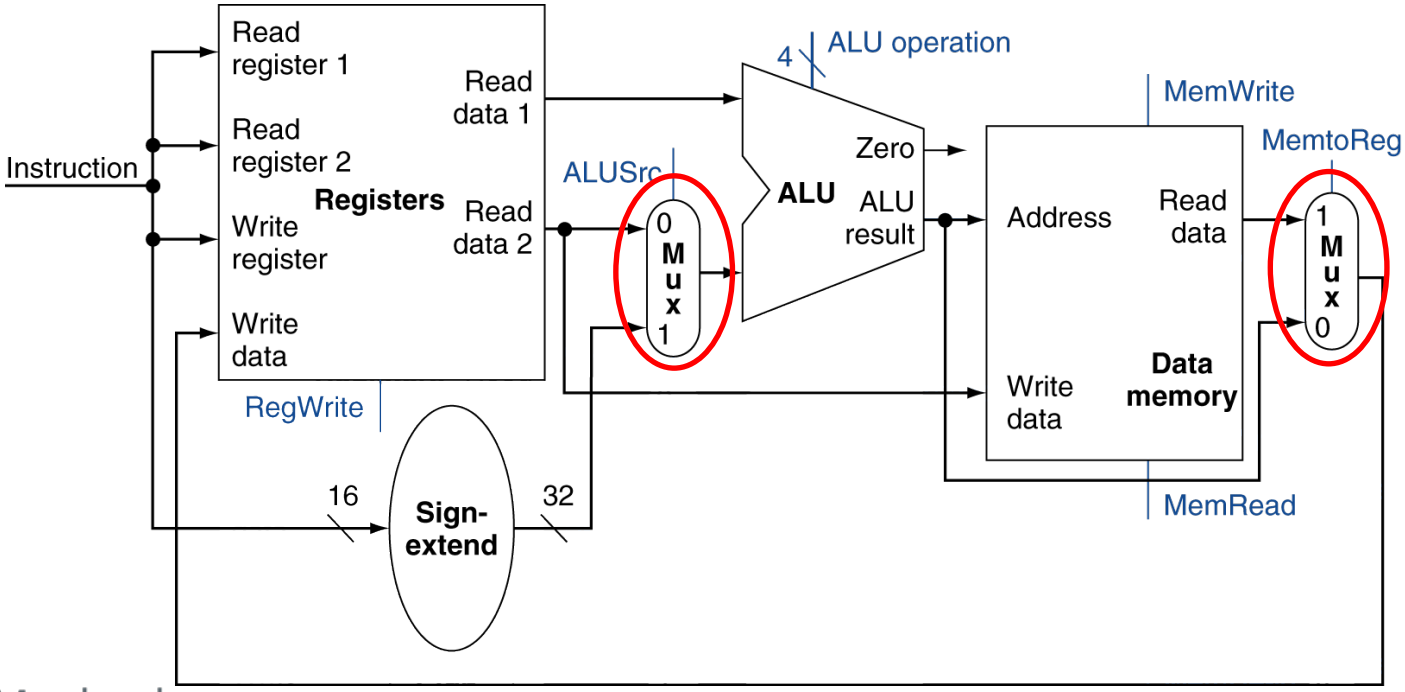


Vollständige Datenpfad-Implementierung (Single Cycle)

- ⇒ seither : nur Datenpfad-Teilentwürfe für die einzelnen Instruktionsklassen
- ⇒ nächste Schritte :
  - ⇒ Zusammenfügen der einzelnen Komponenten
  - ⇒ Aufbau der dazugehörigen Steuerung (Steuerwerk)
- ⇒ zunächst nur für Single-Cycle-Datapath
  (Instruktionsausführung in einem einzigen Taktzyklus)
- ⇒ Single Cycle ist irrelevant für die Praxis, liefert aber einen guten Einblick in die Funktionsweise und bietet somit gute Basis für Multicycle, Pipeline usw.
- ⇒ Ähnliche Datenpfadteile können durch dieselbe Hardware realisiert werden, wenn sie nicht gleichzeitig aktiv sein müssen!
- ⇒ Unterschiedliche Datenquellen für dieselbe Hardware-Einheit können durch Multiplexer implementiert werden.

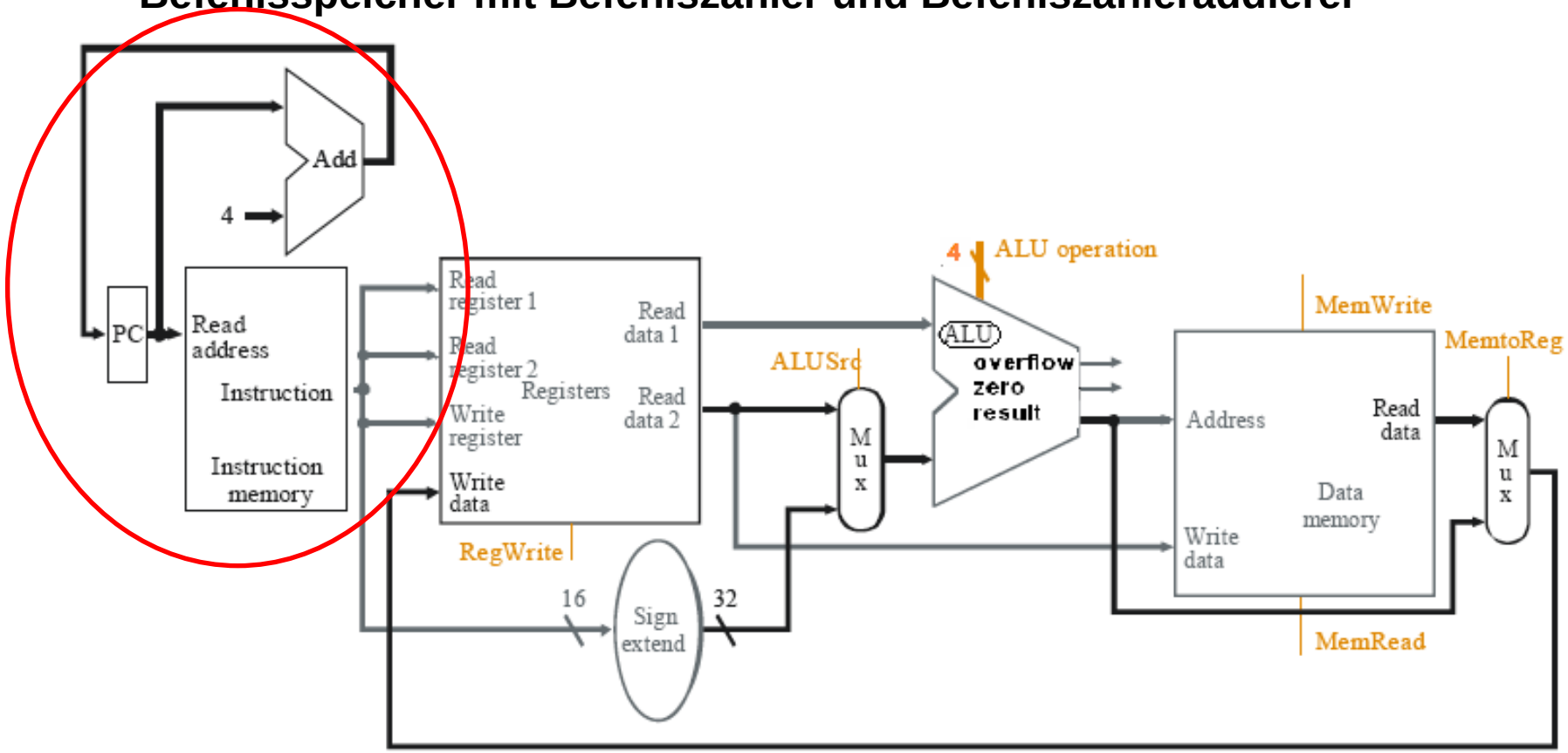
Vollständige Datenpfad-Implementierung (Single Cycle)

- ⇒ Bei R-Type und Speicheroperationen ist der zweite ALU-Eingang entweder das Registerwort oder eine (auf 32 Bit gebrachte) Konstante des Befehlswortes
  - → Multiplexer vor den zweiten ALU-Eingang !
- ⇒ Der Wert, der in ein Zielregister geschrieben wird, ist entweder aus der ALU oder kommt aus dem externen Memory
  - → Multiplexer vor den Registersatz !



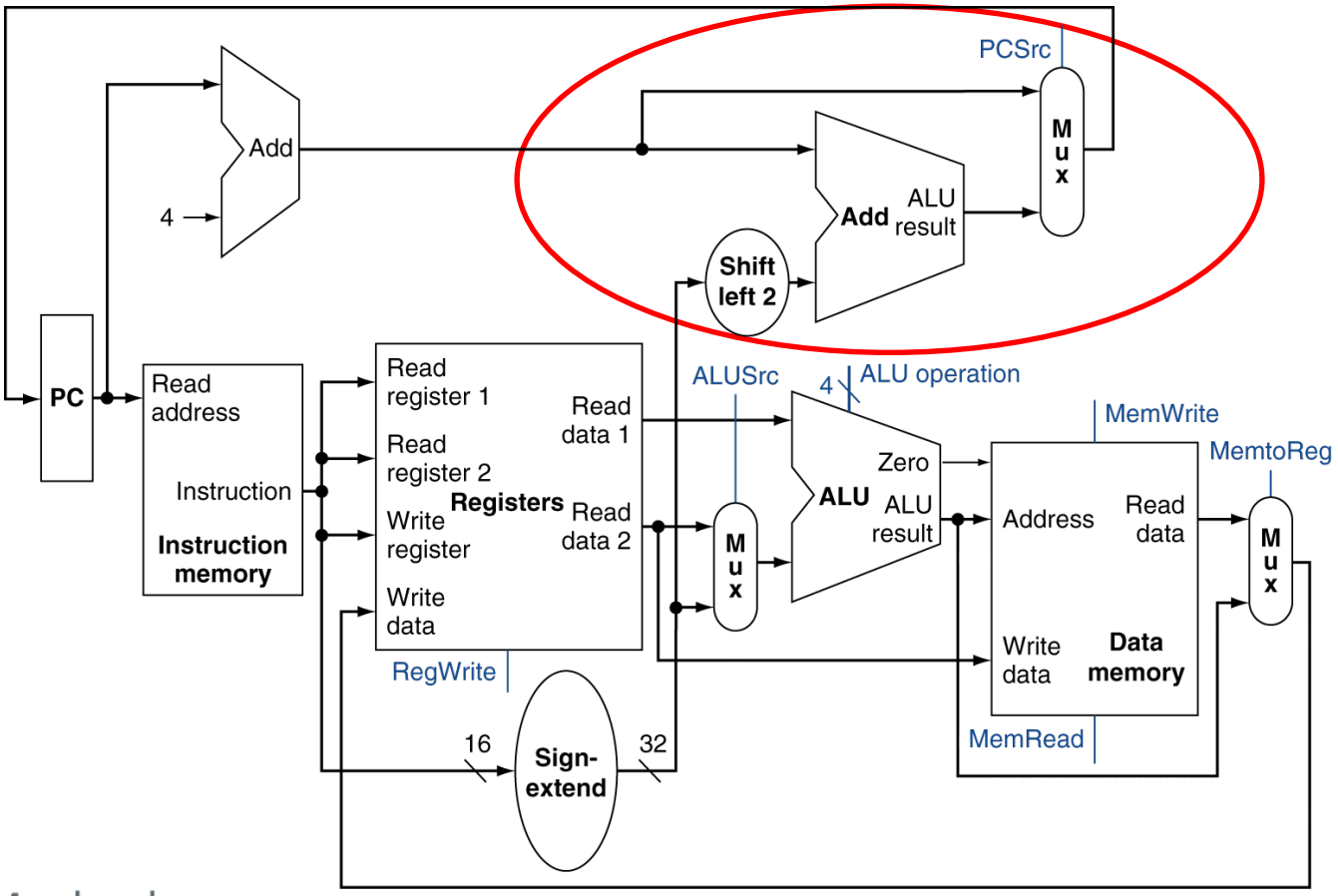
Vollständige Datenpfad-Implementierung (Single Cycle)

- ⇒ **Nun die Befehlsladeeinheit hinzufügen**
- ⇒ **Befehlsspeicher mit Befehlszähler und Befehlszähleraddierer**



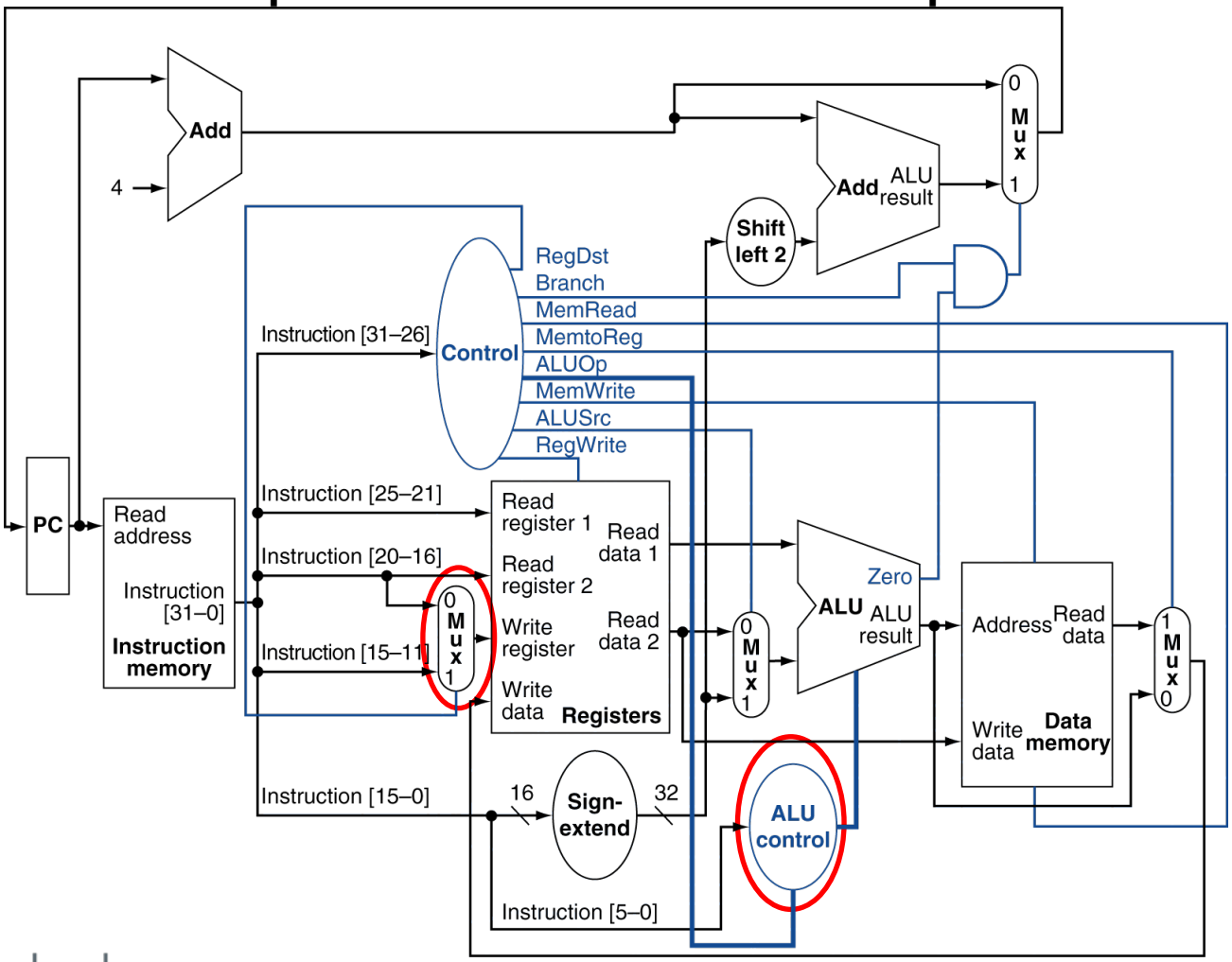
Vollständige Datenpfad-Implementierung (Single Cycle)

- ⇒ **Hinzufügen der BEQ-Einheit**
- ⇒ **Verbinden des Befehlszählers mit der Sprungaddressberechnung**



Vollständige Datenpfad-Implementierung (Single Cycle)

⇒ **Adresse des Zielregisters steht bei R-Type und I-Type an verschiedenen Bitpositionen → MUX für die entsprechenden Bits**



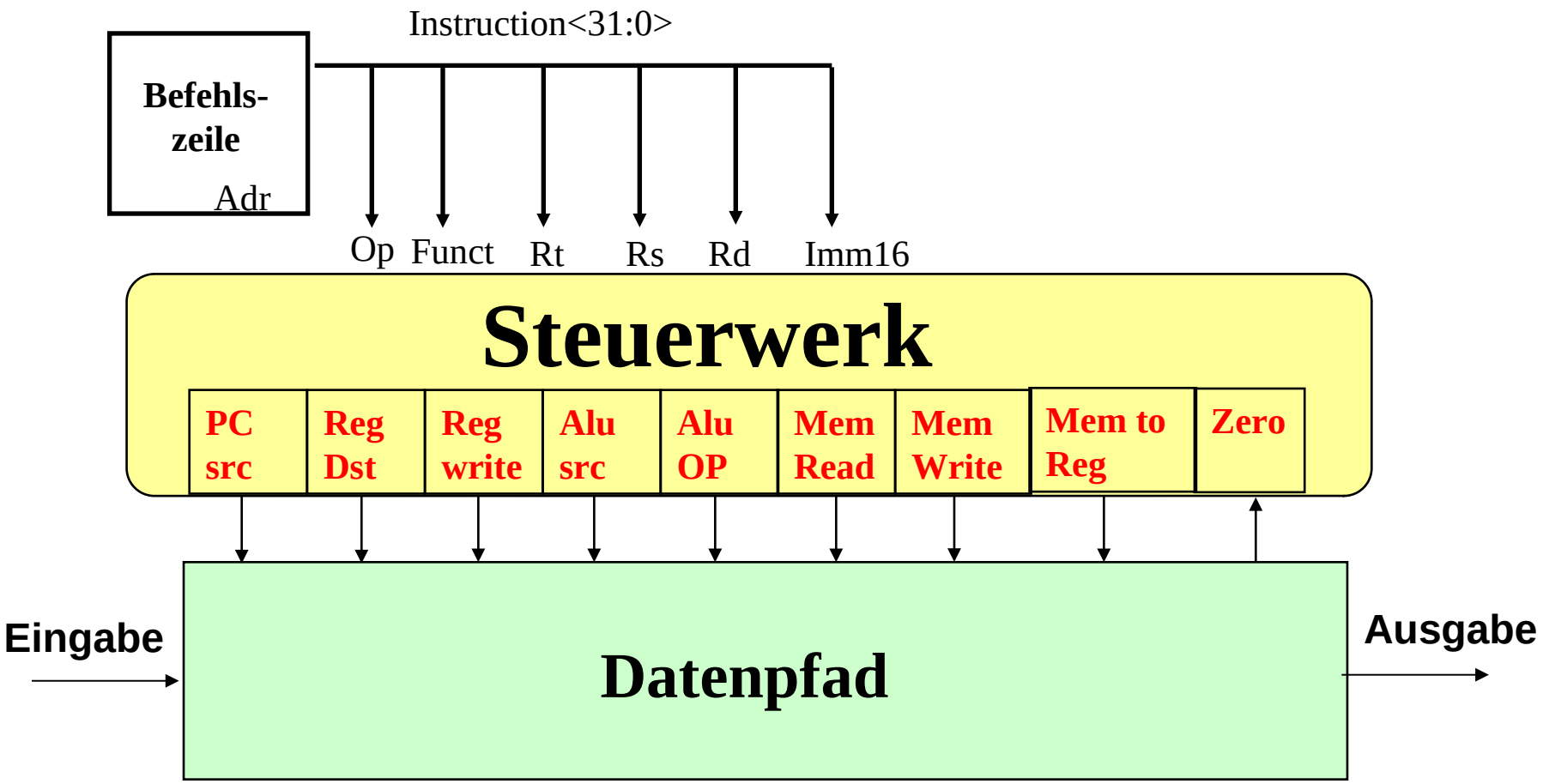
Entwurf des Steuerwerks

Das Steuerwerk hat folgende Aufgaben
⇒ Erzeugen von Steuersignalen für das Schreiben von Speichern und Register
⇒ Selektion von Multiplexereingängen
⇒ Auswahl von ALU-Operationen

Die Steuersignale hängen ab
⇒ nicht nur von der jeweils auszuführenden Instruktion
⇒ sondern auch von Berechnungsergebnissen (also direkt von Daten!)
(z. B. bei der beq, bne Instruktion)

Vorgehensweise
⇒ Beschreibung aufstellen (sprachlich, funktional, Finite State Machine, etc.)
⇒ ggf. Optimierung
⇒ Umsetzung in Zielstruktur
⇒ Hardwarebeschreibungssprache z.B. VHDL
⇒ mikroprogrammierte Steuerung
⇒ festverdrahtete Steuerung (Moore oder Mealy-Schaltwerk)

Datenpfad & Steuerwerk: Einfach-Zyklus



Entwurf des Steuerwerks – ALU Steuersignale

Die ALU benötigt 4 Steuersignale. Die Funktion der Alu und somit die Belegung der ALU-Steuersignale (ALU control) hängt ab vom auszuführenden Befehl :

⇒ Bei R-Type-Befehlen (and, or, add, sub, nor , slt) ist die Funktion der ALU vom "funct"-Feld der Instruktion vorgegeben, der OpCode ist bei diesen Befehlen 0
⇒ Bei lw und sw: ALU muss eine Addition ADD durchführen
⇒ Bei beq: ALU muss eine Subtraktion SUB durchführen

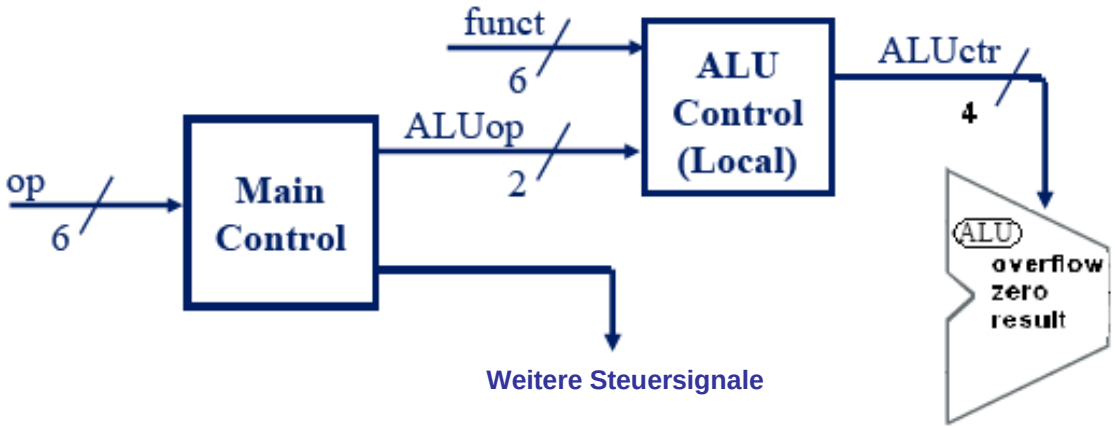
ALU Control Lines	Function
0000	And
0001	Or
0010	Add
0110	Sub
0111	Slt
1100	NOR

Anegate, Bnegate, 2 Bits Operation...

Entwurf des Steuerwerks – ALU Steuersignale

Aufgrund der differenzierten ALU-Ansteuerung empfiehlt sich eine Aufteilung der Funktion des Steuerwerks in 2 Funktionsblöcke

⇒ ALU Steuerung und Hauptsteuerung



Es gibt insgesamt 3 Fälle abzudecken

1. Adressberechnung für LW/SW
2. Vergleich für BEQ BNE
3. R-Type für arithmetrisch-logische Befehle

Entwurf des Steuerwerks – ALU Steuersignale

⇒ ALU-Op-Signal erhält 2 Bit : 00 für LW/SW 01 für BEQ,BNE 10 für R-Type

⇒ ALU Control erhält das o.g. ALU-Op-Signal und das 6 Bit FUNCT-Feld als Eingang

⇒ Die Funktionstabelle für das Ausgangssignal ALU Control lautet dann :

Befehl	ALU Op 1	ALU Op 0	FUNCT	ALU-Funktion	ALU Control
load word	0	0	xxxxxx	add	0010
store word	0	0	xxxxxx	add	0010
BEQ/BNE	0	1	xxxxxx	sub	0110
add	1	0	100000	add	0010
sub	1	0	100010	sub	0110
and	1	0	100100	and	0000
or	1	0	100101	or	0001
slt	1	0	101010	slt	0111
nor	1	0	100111	nor	1100

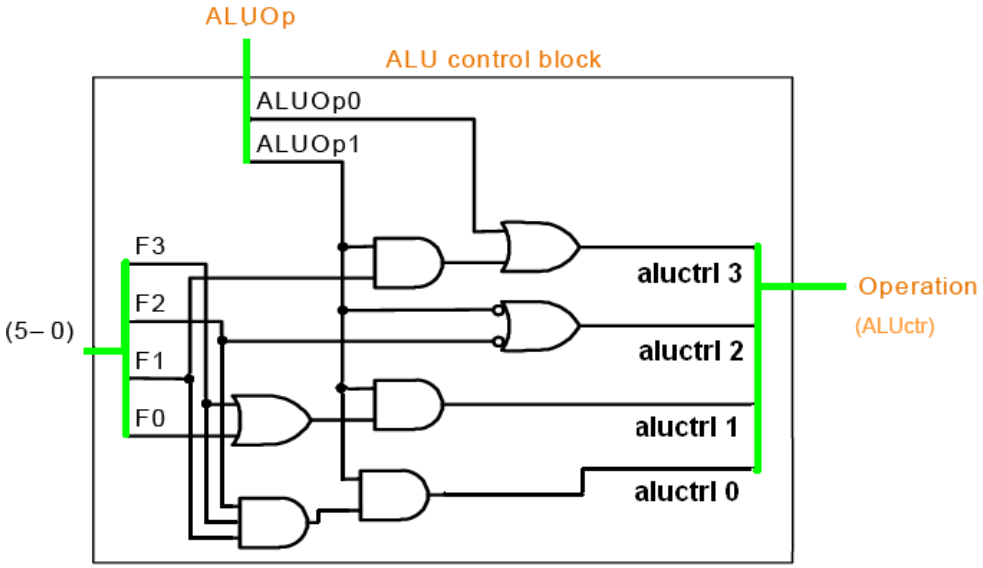
⇒ Aus dieser Tabelle lassen sich Bool‘sche Terme ableiten und vereinfachen.

Entwurf des Steuerwerks – ALU Steuersignale

Vereinfachung :

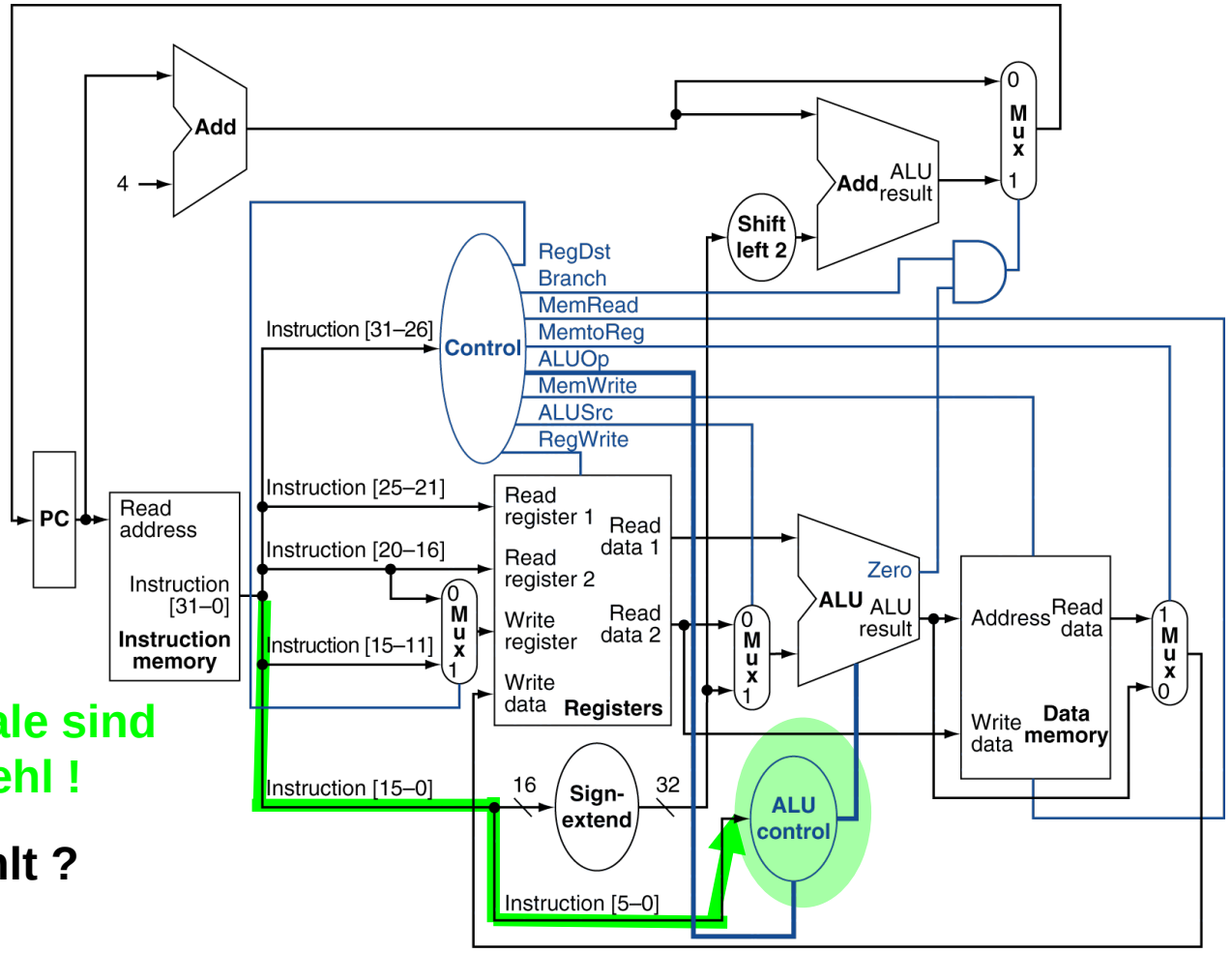
Befehl	ALU Op 1	ALU Op 0	FUNCT	ALU-Funktion	ALU Control
lw/sw	0	0	xxxxxx	add	0010
beq/bne	x	1	xxxxxx	sub	0110
add	1	x	xx0000	add	0010
sub	1	x	xx0010	sub	0110
and	1	x	xx0100	and	0000
or	1	x	xx0101	or	0001
slt	1	x	xx1010	slt	0111
nor	1	x	xx0111	nor	1100

Umsetzung als
festverdrahtetes Schaltwerk :



Der ALU-Control-Block ist eine rein kombinatorische Einheit !

Entwurf des Steuerwerks – Hauptsteuerung



Steuersignale sind Bits im Befehl !

Was fehlt ?

Steuerung der verbleibenden Signale :

⇒ Regdst, Alusrc, Regwrite, Memwrite, Memtoreg, Memread und PCsrc/Branch

Entwurf des Steuerwerks – Hauptsteuerung

Beschreibung der Signale :

⇒ Regdst, Alusrc, Regwrite, Memwrite, Memtoreg, Memread und PCsrc

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20–16).	The register destination number for the Write register comes from the rd field (bits 15–11).
RegWrite	None	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc Branch	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

⇒ Somit lassen sich für jedes Instruktionsformat die Steuerleitungen in Abhängigkeit von den Bits des Opcodes setzen.

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Entwurf des Steuerwerks – Hauptsteuerung

Somit ergibt sich folgende Funktionstabelle in Abhängigkeit der OP_Code-Bits:

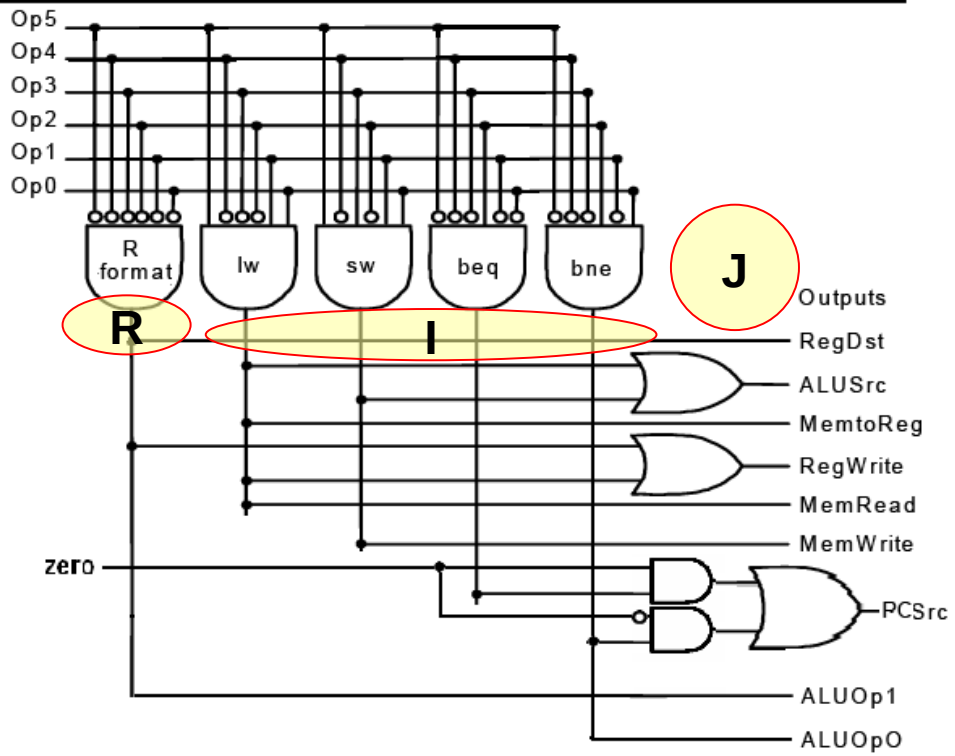
Befehl	OpCode	OPcode5	OPcode4	OPcode3	OPcode2	OPcode1	OPcode0	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	PCsrc	ALUOp1	ALUOp0
R-Type	hex 0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
load word	hex 23	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
store word	hex 2b	1	0	1	0	1	1	x	1	x	0	0	1	0	0	0
beq	hex 4	0	0	0	1	0	0	x	0	x	0	0	0	1	0	1
bne	hex 5	0	0	0	1	0	1	x	0	x	0	0	0	1	0	1

BNE und BEQ Werte sind gleich, hier ist eine Verknüpfung mit dem Zero-Flag nötig

Aus der Tabelle lassen sich wiederum die Bool'schen Funktionen für die Steuer-signale ableiten.

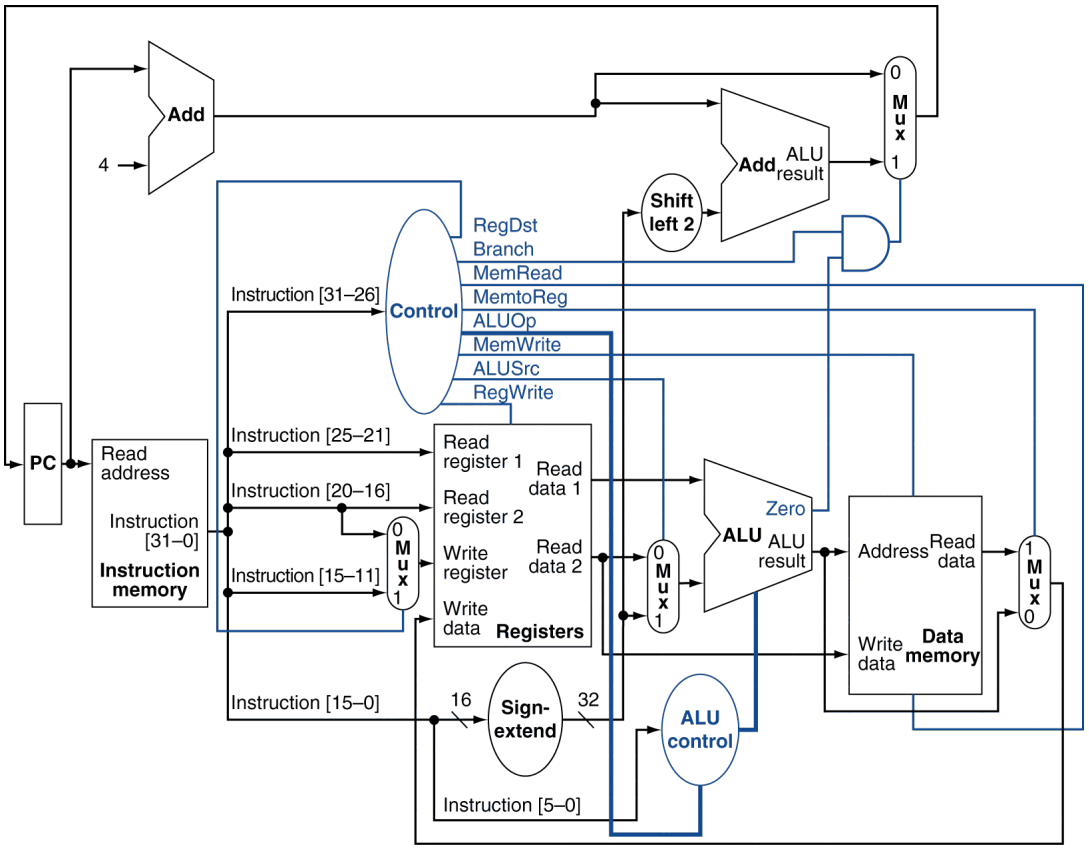
Das gesamte Steuerwerk ist eine rein kombinatorische Einheit !

Auch J-Type lässt sich analog dazu rein kombinatorisch realisieren.
→ kommt gleich ….



Mikrocomputertechnik 1 – Single Cycle Datenpfad und Steuerwerk

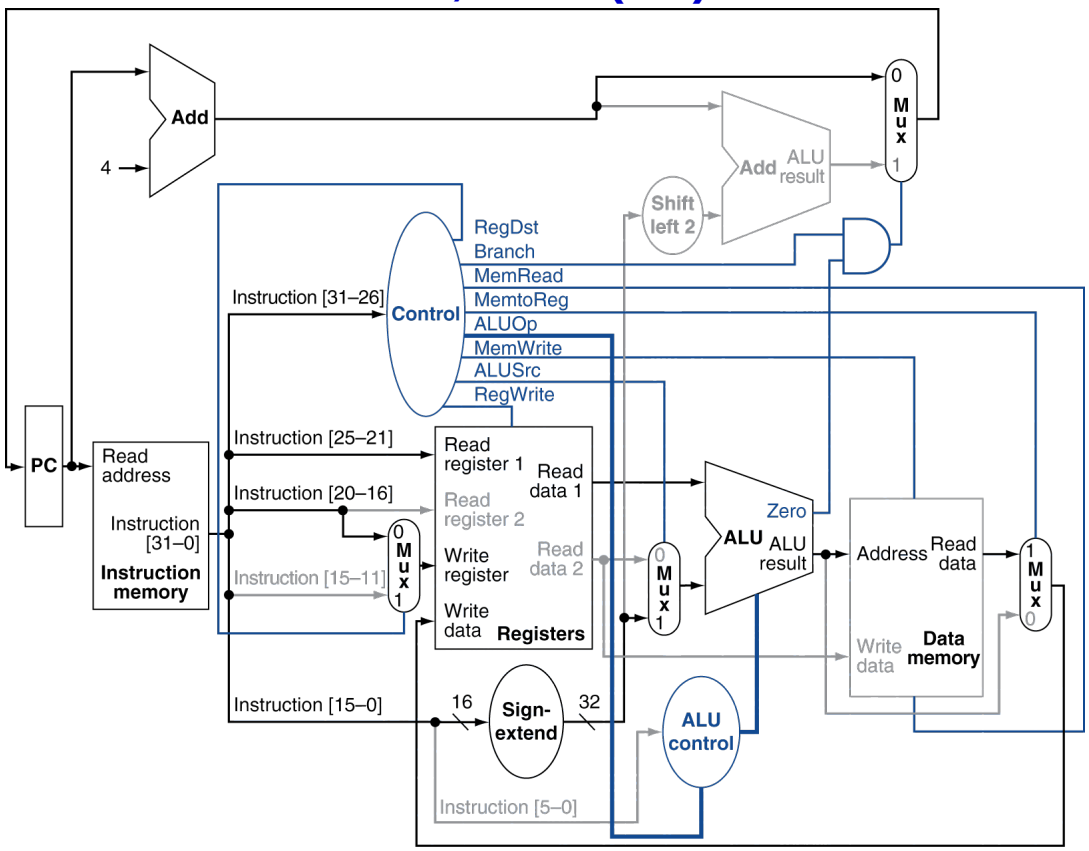
Beispiel : Ablauf des Befehls add $t1, $t2, $t3



⇒ Befehl aus Instruktionsspeicher lesen und Befehlszähler inkrementieren
⇒ Register $t2 und $t3 lesen und Steuersignale der Hauptsteuerung setzen
⇒ ALU bearbeitet die beiden Registerinhalte unter Auswertung des “funct”- Feldes
⇒ Schreiben des ALU-Resultats in Register $t1

Mikrocomputertechnik 1 – Single Cycle Datenpfad und Steuerwerk

Beispiel : Ablauf des Befehls lw $t1,offset ($t2)



⇒ Befehl aus Instruktionsspeicher lesen und Befehlszähler inkrementieren

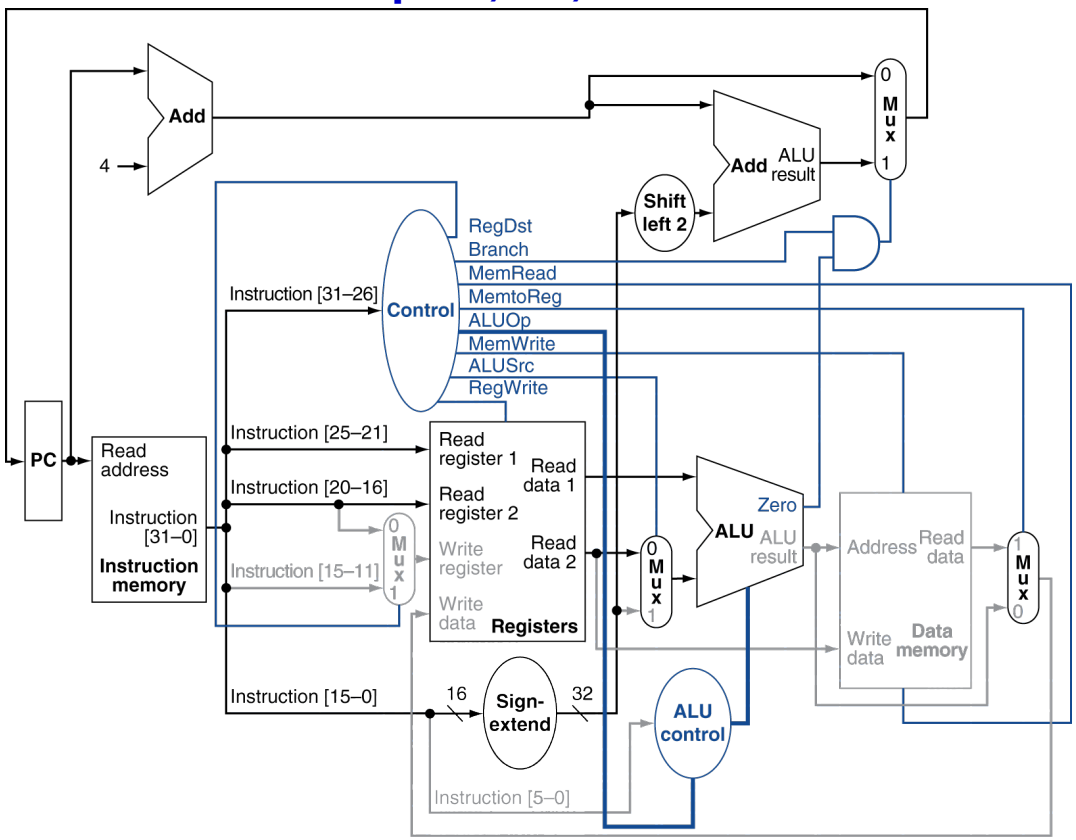
⇒ Register $t2 lesen und Steuersignale der Hauptsteuerung setzen

⇒ ALU errechnet Summe des Wertes in $t2 und des „sign-extended“ Offset-Wertes

⇒ Schreiben des Speicherinhalts in Register $t1

Mikrocomputertechnik 1 – Single Cycle Datenpfad und Steuerwerk

Beispiel : Ablauf des Befehls beq $t1,$t2,offset



⇒ Befehl aus Instruktionsspeicher lesen und Befehlszähler inkrementieren

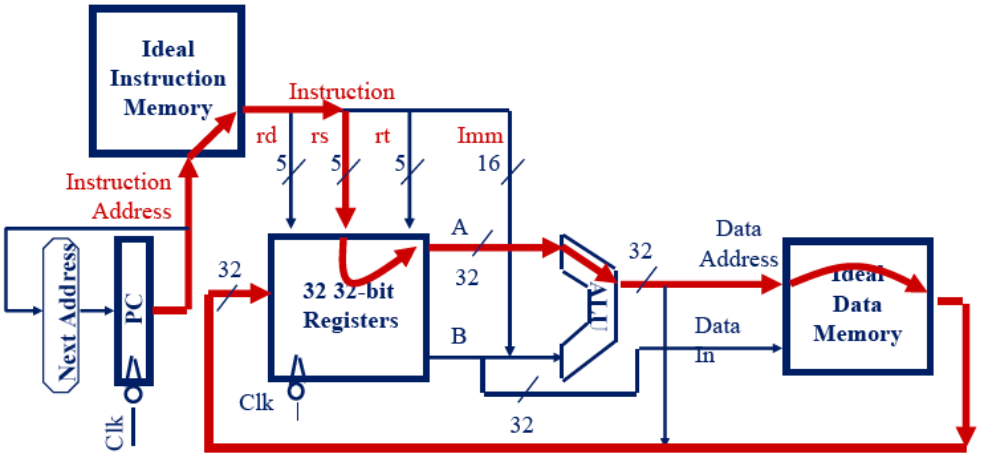
⇒ Register $t1 und $t2 lesen und Steuersignale der Hauptsteuerung setzen

⇒ ALU subrahiert die Inhalte von $t1 und $t2 und addiert den „sign-extended“ Offset-Wert zum aktuellen Befehlszählerwert

⇒ das Zero-Flag entscheidet, welcher Wert in den Befehlszähler übernommen wird

Nachteile der Single Cycle CPU

⇒ Taktzykluszeit muss sich am sog. „kritischen Pfad“ der kombinatorischen Einheiten orientieren, darunter versteht man den längsten Pfad, entlang dessen sich die Gatterlaufzeiten addieren

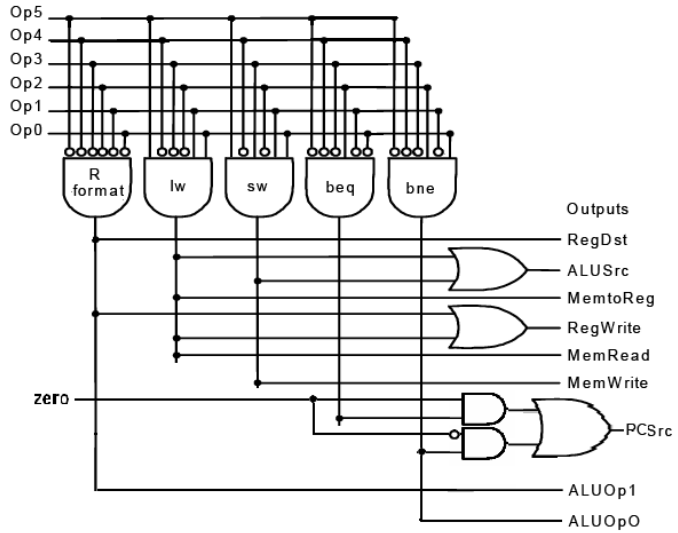


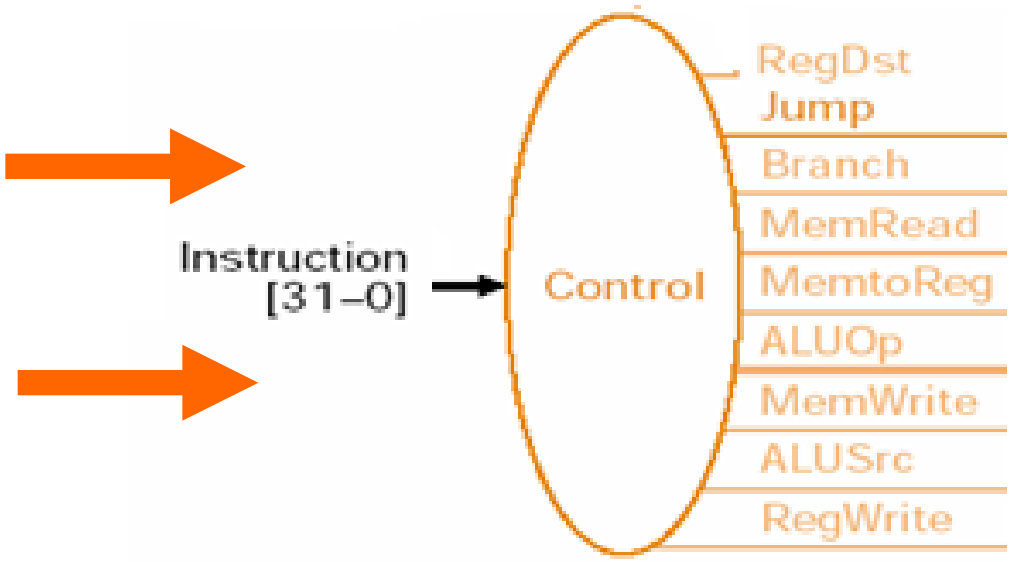
⇒ Kritischer Pfad hier : Gatterlaufzeiten für Load Word sind am längsten

⇒ Load-Befehl verhindert höhere Taktfrequenz

⇒ einige Hardwareblöcke müssen mehrfach vorhanden sein, da jede Einheit in einem Taktzyklus nur einmal verwendet werden kann.

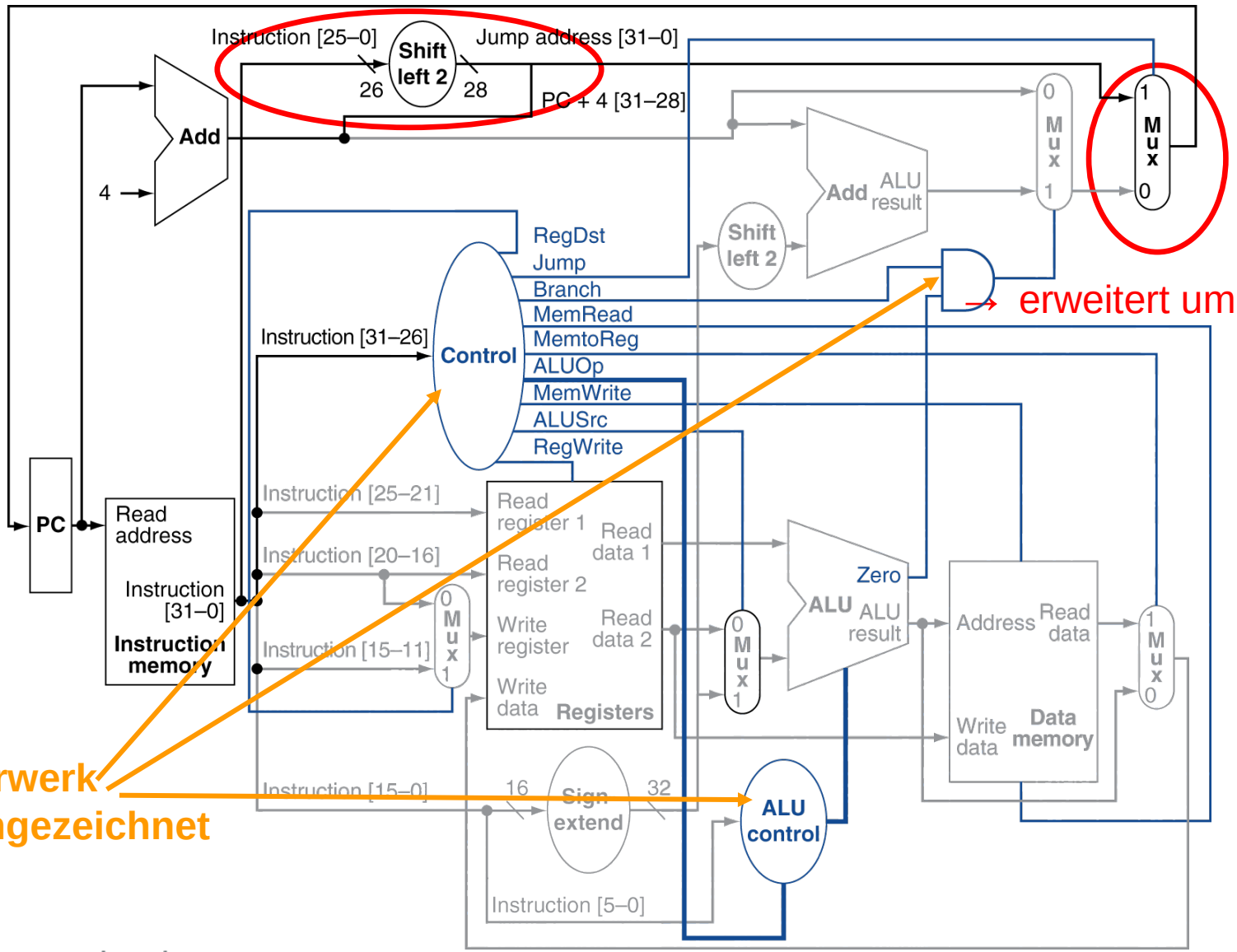
Steuerwerk inklusive Jump als Blockschaltbild





Mikrocomputertechnik 1 – Single Cycle Datenpfad und Steuerwerk

Vollständiger Single-Cycle- Datenpfad mit Steuerwerk



→ erweitert um J-Type

→ **Steuerwerk miteingezeichnet**

Zusammenfassung

⇒ Entwurf des Datenpfads eines Single Cycle-Prozessors ist rein kombinatorisch
⇒ Vorteil : nur ein Takt pro Instruktion
⇒ Nachteil : Taktzeit wird wegen lw-Befehl sehr lang

Gesamtzusammenfassung

Wir sind die folgenden Schritte durchlaufen, um einen Prozessor zu entwerfen :
⇒ Erstellung eines möglichst einfachen ISA-Befehlssatz
⇒ Zusammenstellung der Hardwarekomponenten,
die diesen Befehlssatz umsetzen können
⇒ Zusammensetzen der Hardwarekomponten zum Datenpfad
⇒ Implementierung der einzelnen Steuerwerksignale anhand der einzelnen Befehle
⇒ Zusammensetzen zur Gesamtsteuerung

⇒ Die MIPS-ISA hat dieses Vorhaben erleichtert
→ Befehle sind alle gleich lang →Ausgangsregister sind immer an derselben Stelle
→ Immediate-Variablen sind immer an derselben Stelle und gleich lang
→ Operationen erfolgen immer mit Registern oder Immediate-Variablen

Referenzen

- Patterson, D; Hennessy , J; 2017; „Rechnerorganisation und Rechnerentwurf - Die Hardware/Software Schnittstelle”; Spektrum Akademischer Verlag; ISBN 9783110446050
- Wikipedia - The Free Enzyklopädie; wikipedia.com
- Wikimedia Commons; commons.wikimedia.org
- Google LLC; google.com
- ARM®; 2016; „ARM® Cortex®-A72 MPCore Processor - Technical Reference Manual“
- ARM®; 2015; „ARM® Cortex®-A Series - Programmer’s Guide for ARMv8-A“