Kapitel 8 (3. Teil MIPS): Steuerwerk für die Eintakt-Implementierung

Technische Grundlagen der Informatik 2
(Rechnertechnologie 2)
SS 2006

Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen

Auf Basis von Material von

Rolf Hoffmann

FG Rechnerarchitektur

Technische Universität Darmstadt

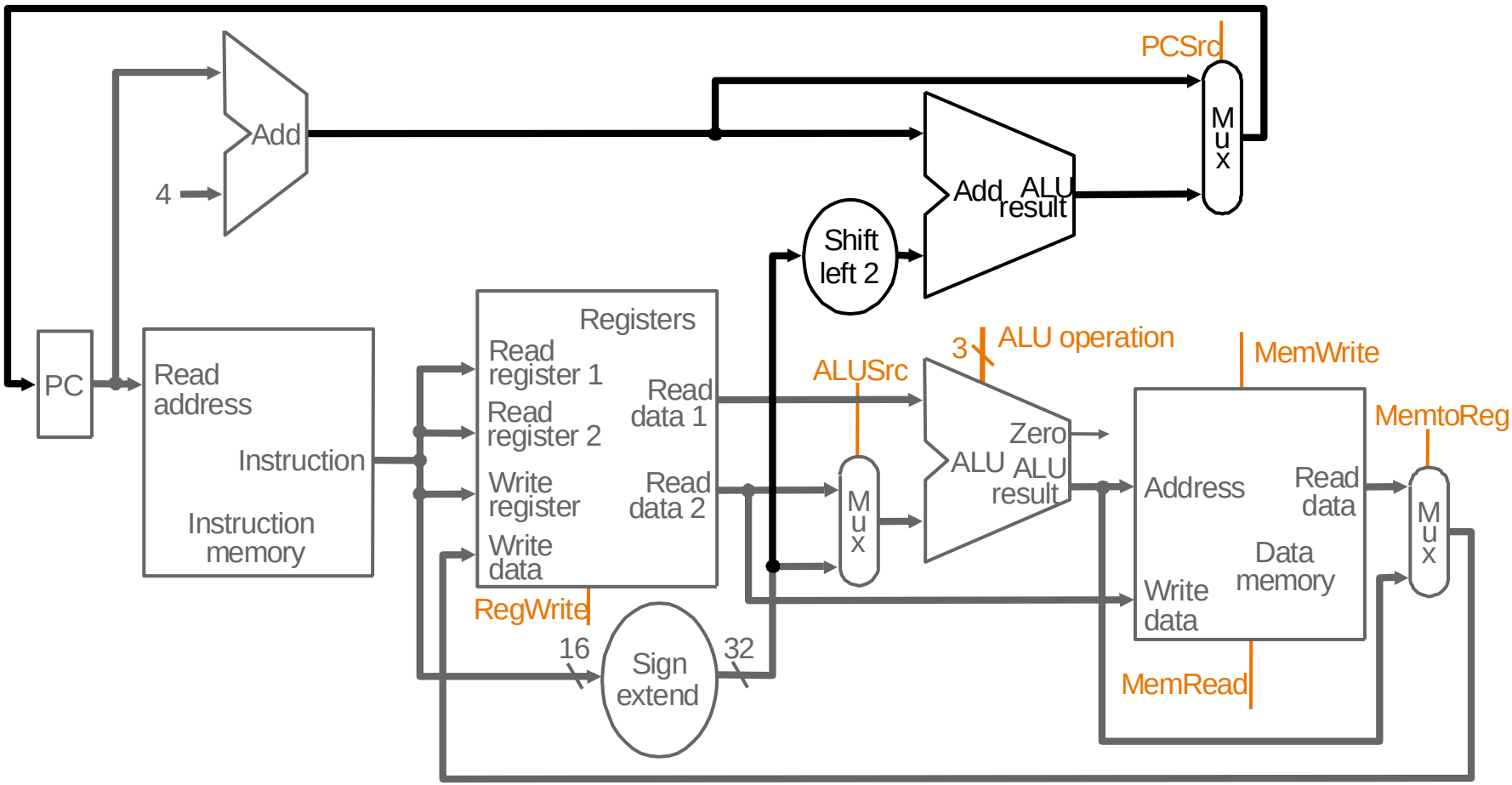
In Anlehnung an das Patterson/Hennessy: Computer Organization & Design, 2nd Edition, Chapter 3, 5

Es sind auch die Folien von Dr. M. G. Wahl (Univ. Siegen, Inst. Mikrosystemtechnik) und ähnliche aus den Grundzügen der Informatik II, SS03, von Prof. Dr. Oskar von Stryk verwendet worden.

Inhalt

- Thema: Entwicklung des Steuerwerks für die Eintakt-Implementierung
- Aufgabe des Steuerwerks
- Betrachtung der Steuersignale zur Steuerung von
  - ALU
  - Lesen und Schreiben der Speicherelemente
  - Multiplexer zur Durchschaltung der aktuellen Wege
  - Auswahl der nächsten Befehlsadresse PCnext
- Der Kritische Pfad

„Single Cycle“-Operationswerk (Wdh.)



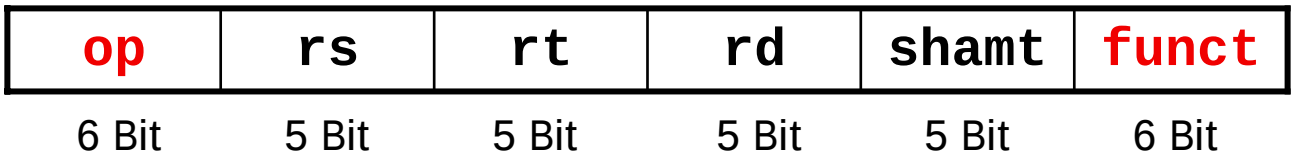
Was noch fehlt:

das Steuerwerk (Control Unit), wird jetzt schrittweise entwickelt

Aufgabe des Steuerwerks

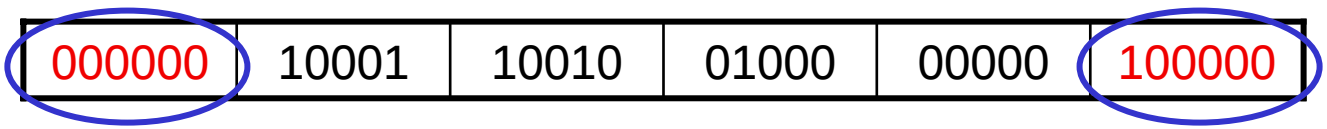
- Erzeugen von Steuersignalen für
  - das Schreiben von Speicherelementen (Speicher und Register), Write-Enable, Clock-Enable
  - die Selektion von Multiplexereingängen
  - die Auswahl von ALU-Operationen
- Die Steuersignale hängen ab von
  - dem jeweils auszuführenden Befehl sowie
  - teilweise auch von Berechnungsergebnissen (d. h. von Daten!) (z. B. bei **beq**-Befehl).
  - teilweise auch von dem Status der Maschine (beeinflusst durch vorhergehende Befehle, Eingangssignale (z. B. Interrupts))

Wdh.: MIPS-Befehlsformat R



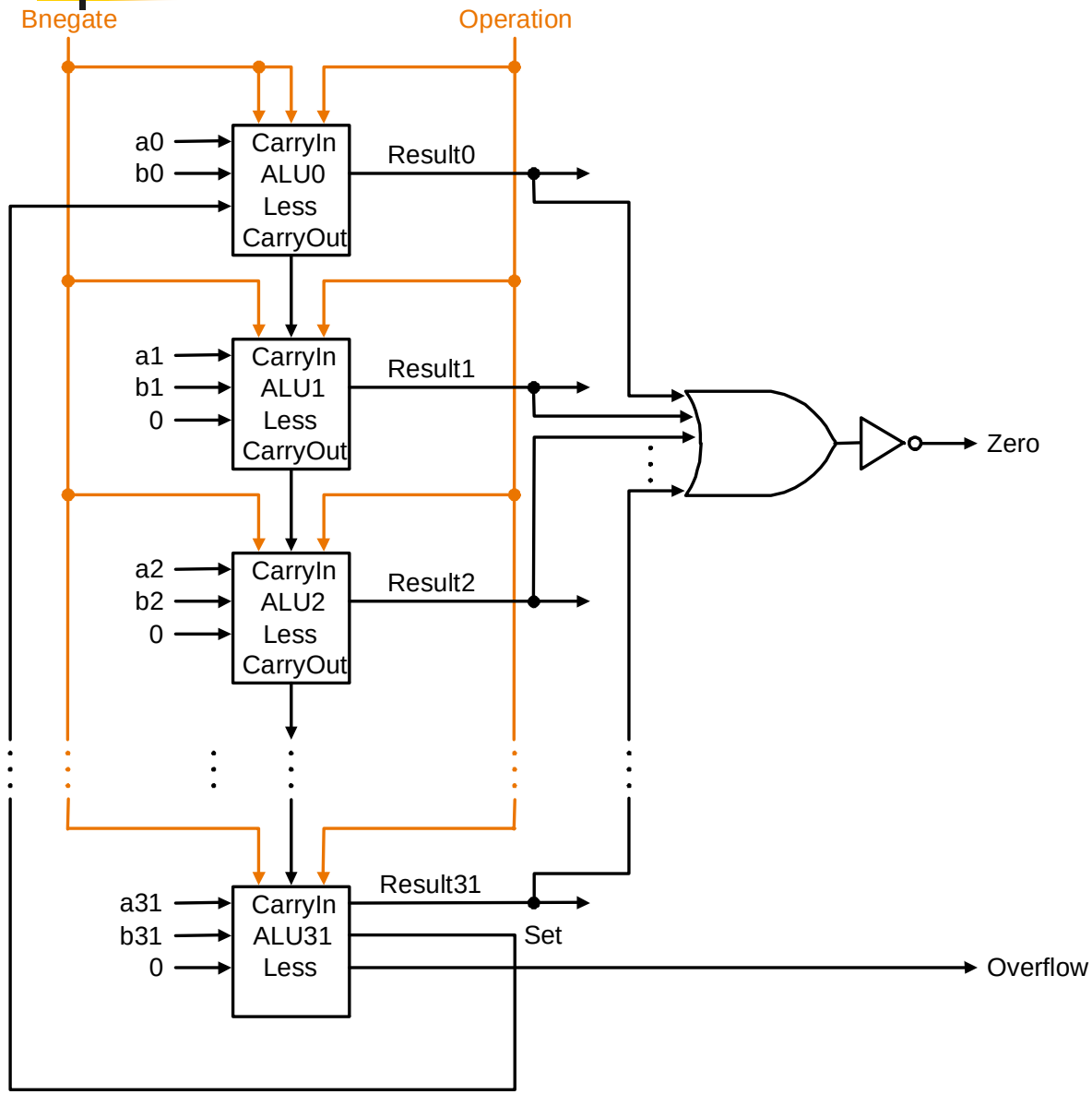
- **op**: „opcode“; Grundtyp des Befehls (instruction)
- **rs**: erster Quelloperand (Register)
- **rt**: zweiter Quelloperand (Register)
- **rd**: Ziel (Register)
- **shamt**: „shift amount“, für Schiebeoperationen
- **funct**: „Funktionscode“; Ergänzung zum **op**-Feld: Spezifikation der durchzuführenden Operation (and, or, add, subtract, slt, ...)

- Beispiel: **`add $t0, $s1, $s2`**

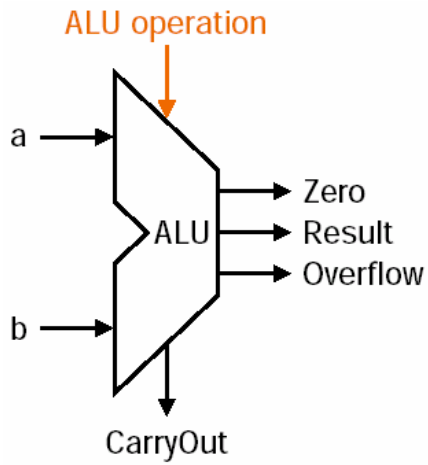


- Die auszuführende ALU-Operation ist durch Befehlstyp **op** und Funktionscode **`funct = add`** bestimmt.

Vollständige 32-Bit ALU (Wdh.)



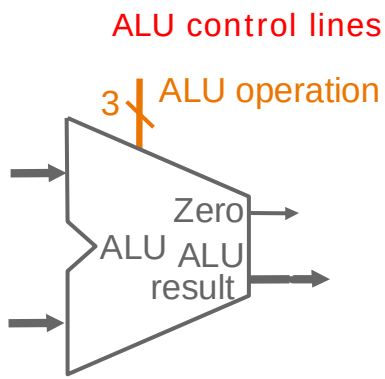
000 = and
001 = or
010 = add
110 = subtract
111 = slt



ALU-Steuersignale

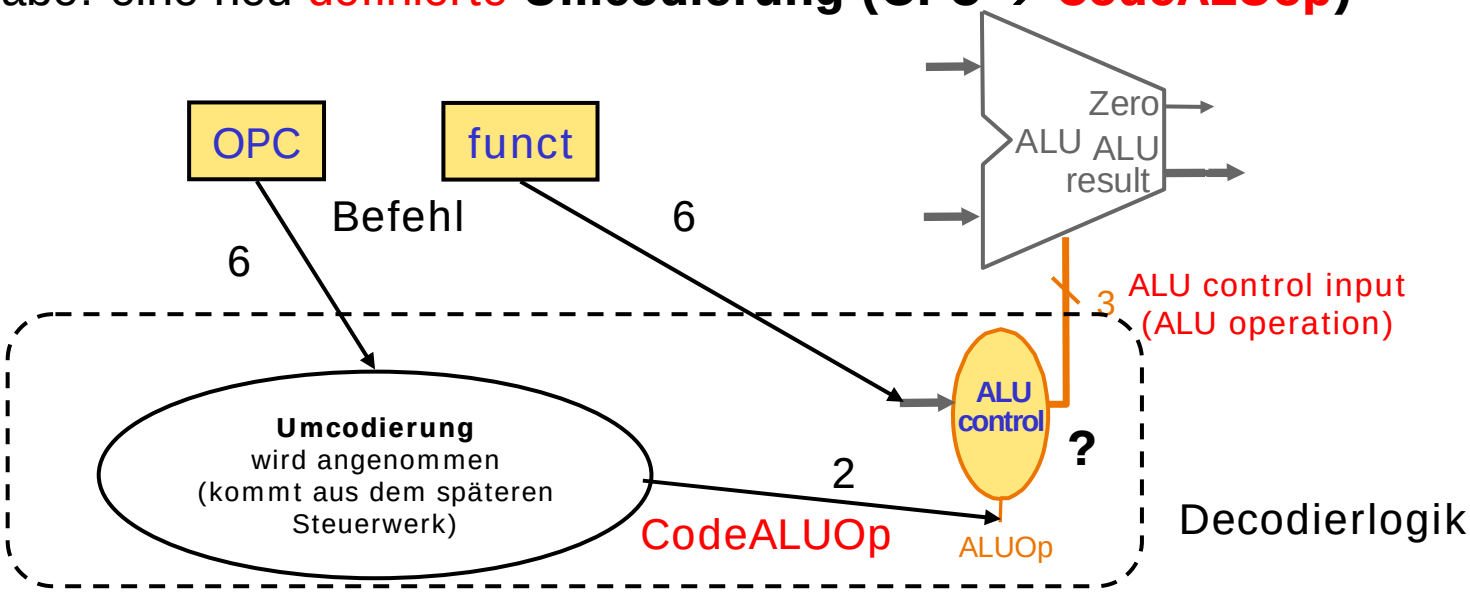
- Die ALU benötigt 3 Bits als Steuersignale zur Auswahl der ALU operation (aber nur 5 der 8 möglichen Kombinationen werden benutzt).
- Die konkrete Belegung der Steuersignale hängt von dem jeweils auszuführenden Befehl ab.
  - Für **`lw`** und **`sw`** (I-Format): Die ALU wird zur Berechnung der effektiven Datenadresse (Basisregister + Offset) benutzt.
    - ALUoperation = Add
  - Für Befehle vom R-Format: AND, OR, Add, Subtract oder Set-on-less-than und in Abhängigkeit vom „funct“-Feld des Befehls wird die zugeordnete Rechenoperation verlangt:
    - ALUoperation = And, Or, Subtract, Add, Set-on-less-than

ALU Control lines	Function
000	And
001	Or
010	Add
110	Subtract
111	Set-on-less-than

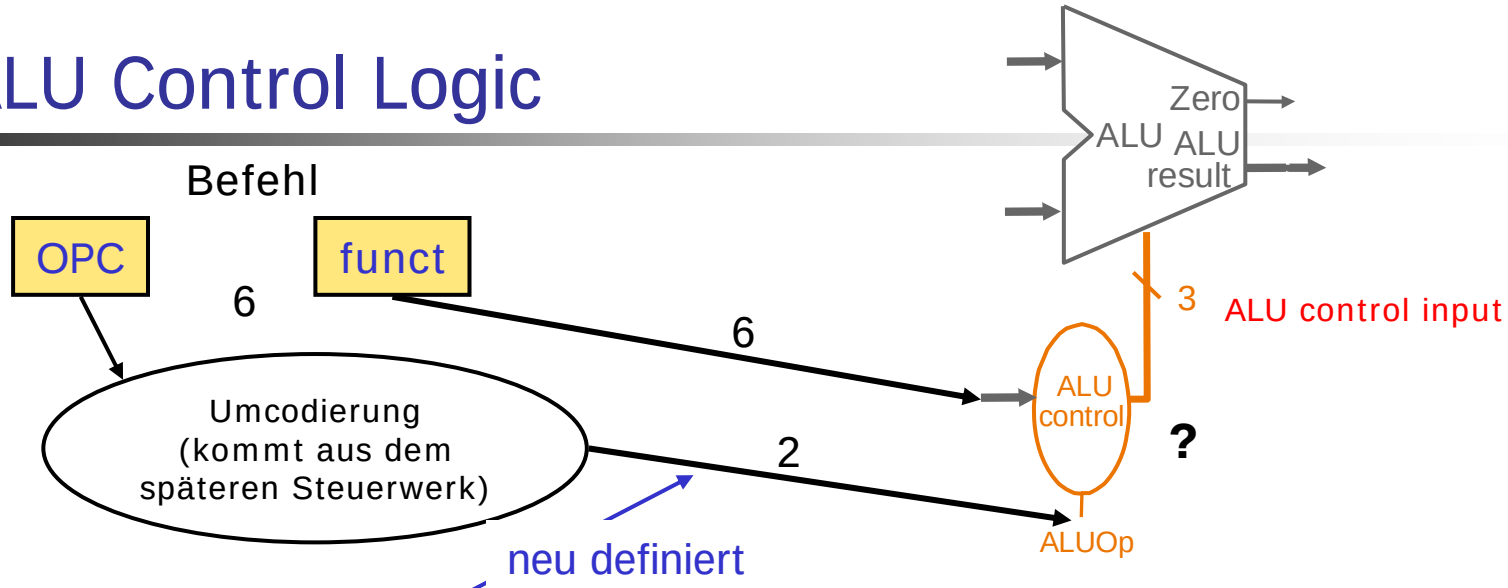


ALU Control Logic

- **Entwurf einer kleinen Logik (ALU Control Logic) für die ALU.**
- Diese Logik ist nicht besonders sinnvoll, da sie zu einer zusätzlichen Zeitverzögerung führt. Besser sollte nur ein Decodierlogik (Eingänge OPC und funct, Ausgang ALU control) benutzt werden. Wir wollen aber dem Buch folgen.
- Eingaben: „funct“-Feld des Befehls und 2-Bit Hilfssignal ALUOp (wird aus Opcode-Feld des Befehls berechnet)
- Ausgabe: 3-Bit ALU control input (bestimmt den ALU Operator)
- Vorgabe: eine neu definierte **Umcodierung (OPC → CodeALUop)**



ALU Control Logic



OPC

Instruction opcode	ALUOp	Instruction operation	funct Function code	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	010
SW	00	store word	XXXXXX	add	010
Branch equal	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set-on-less-than	101010	set-on-less-than	111

10 0011
10 1011
00 0100
00 0000
00 0000
00 0000
00 0000
00 0000

Aus dieser Tabelle ergeben sich die 3 Bit-Steuersignale der ALU in Abhängigkeit vom Befehl (ALUOp und funct)

ALU Control Logic, Minimierung

- Boolesche Funktionen mittel Wahrheitstafel beschreibbar (die wiederum mit Logik-Synthese-System automatisch in Gatter umgesetzt werden kann):
- Die „X“-Einträge stellen „don‘t cares“ dar. Sie werden zur vereinfachten Implementierung der booleschen Funktionen verwendet

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	010
SW	00	store word	XXXXXX	add	010
Branch equal	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set-on-less-than	101010	set-on-less-than	111

Umcodierung
mit Don't Cares

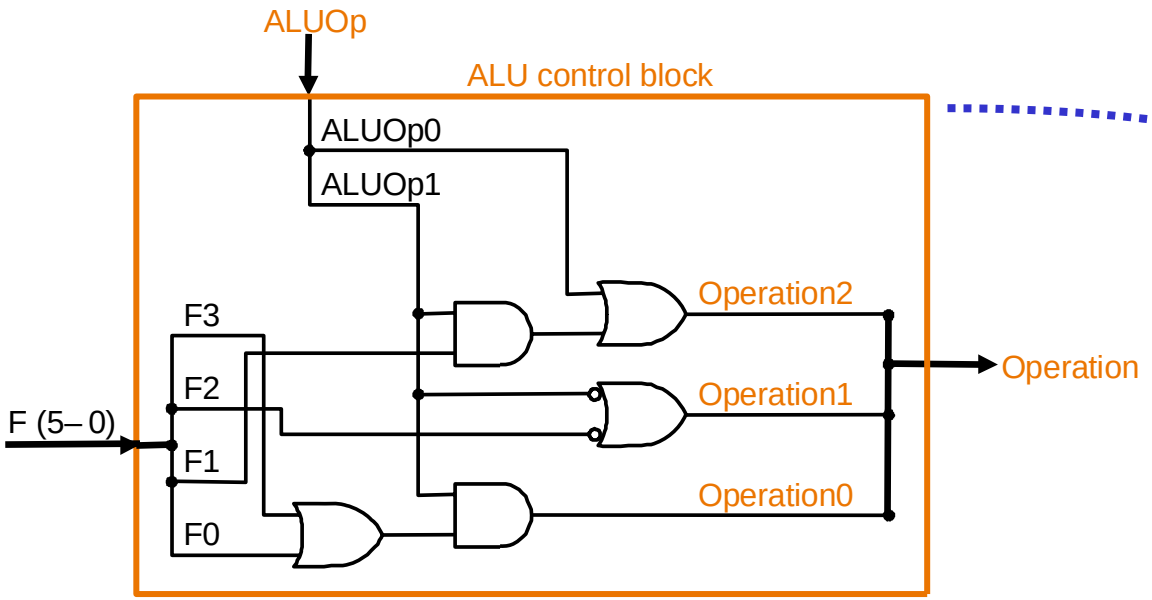
ALUOp		Function code						ALU control input
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

ALU Control Logic

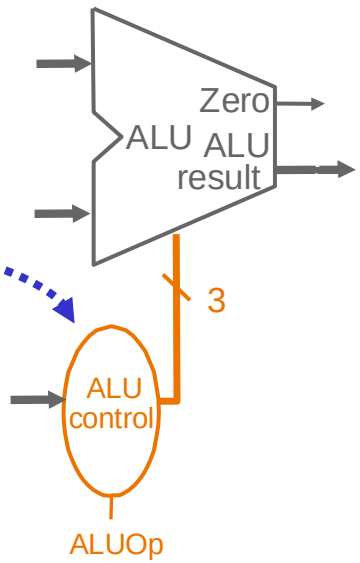
Wahrheitstafel für Funktion der ALU-Steuersignale

ALUOp		Function code						ALU control input
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

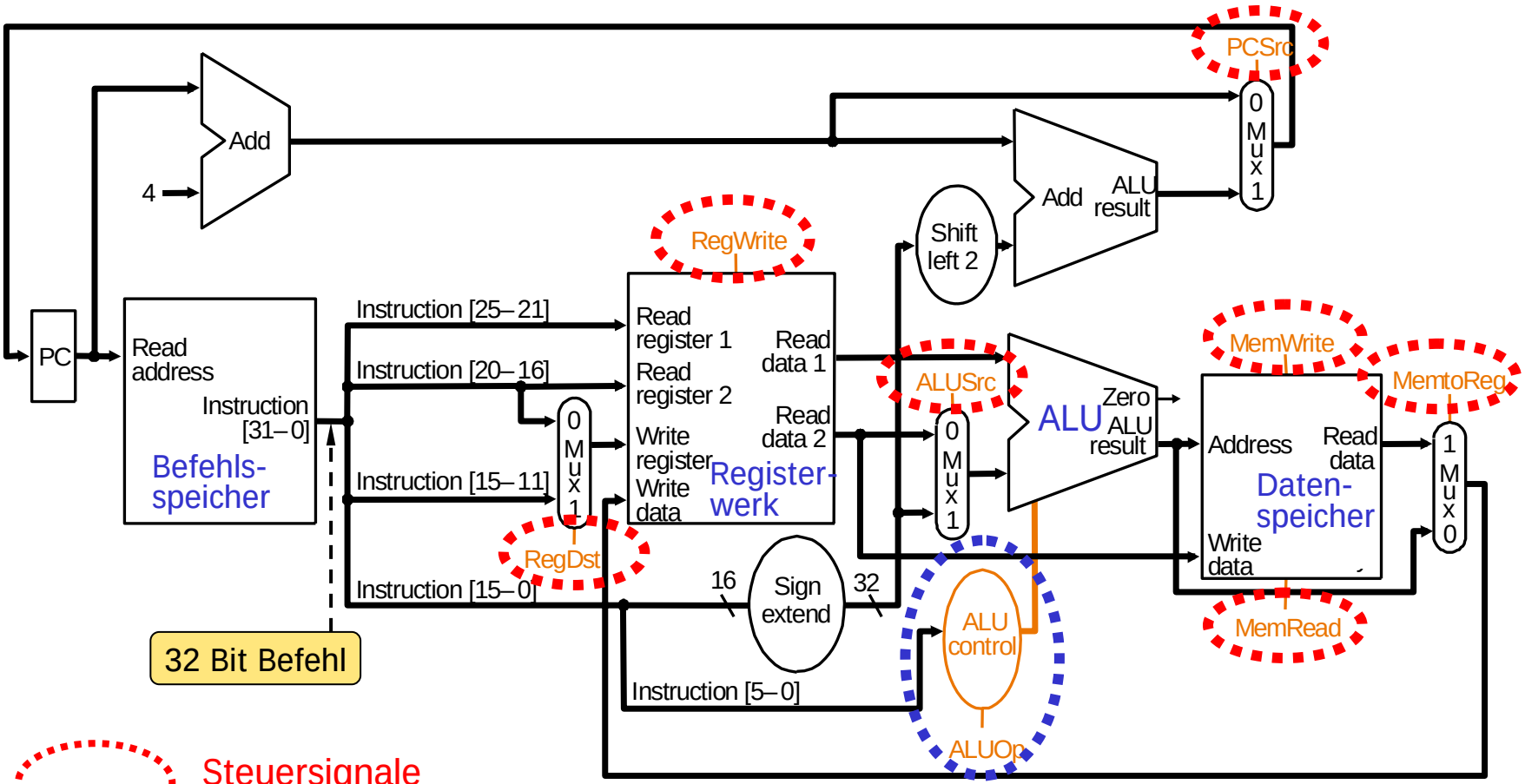
Umsetzung in Schaltung für ALU-Steuersignale



Symbolische Darstellung



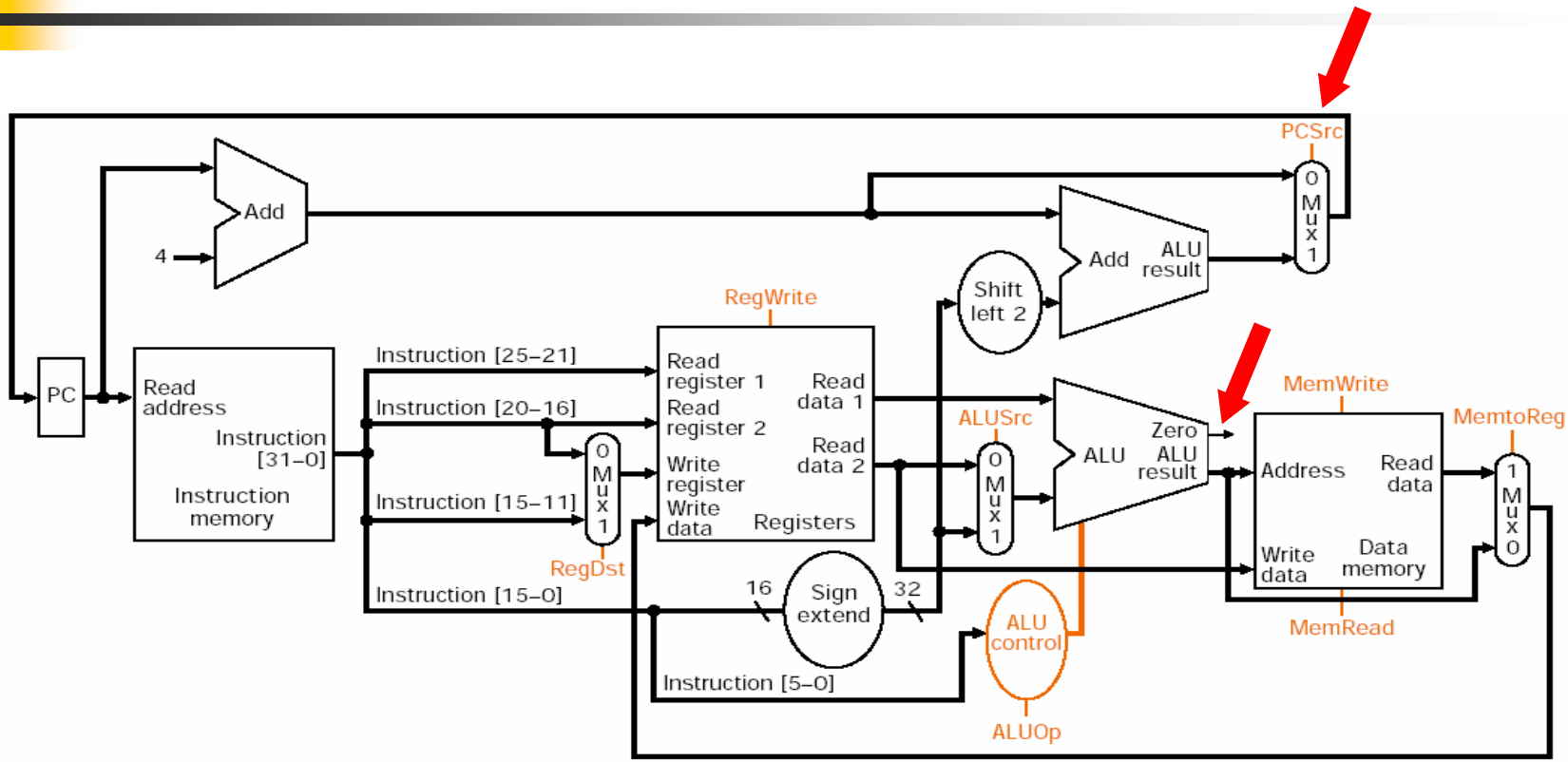
Operationswerk mit Steuersignalen



Bedeutung der 7 Steuersignale

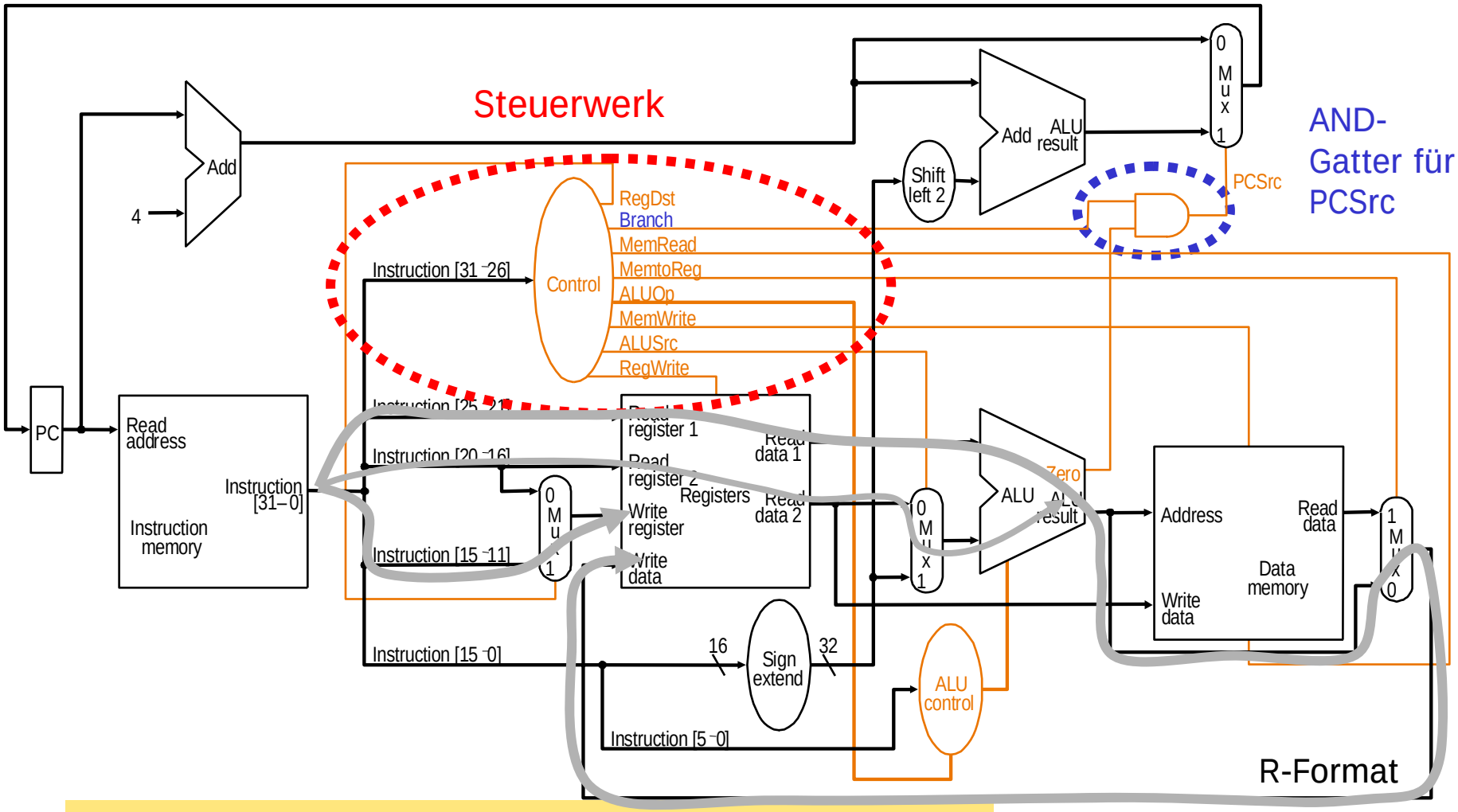
Signal name	Wirkung, wenn nicht gesetzt (0)	Wirkung, wenn gesetzt (1)
RegDst	The register destination number for the Write register comes from the rt field (bits 20–16).	The register destination number for the Write register comes from the rd field (bits 15–11).
RegWrite	None	The register on the Write 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	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.

Laden der neuen Befehlsadresse



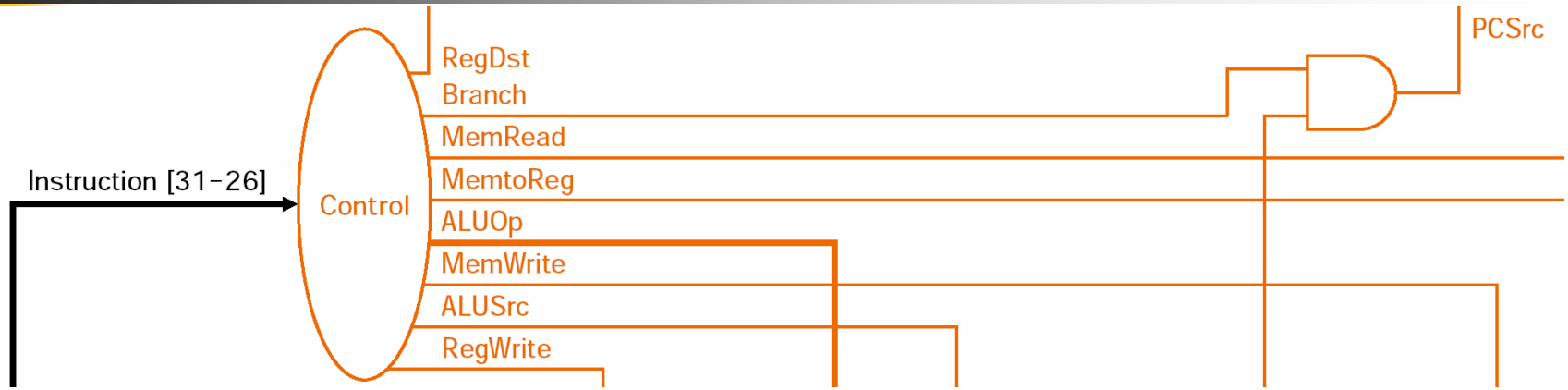
- (Bei bedingtem Sprung) PCSrc ist das einzige Steuersignal das nicht nur vom Befehl abhängt, sondern auch vom Ergebnis einer Berechnung (Zero-Ausgang der ersten ALU)!
- Es bestimmt, ob die Folgeadresse PC+4 oder die Sprungadresse benutzt wird.

Operationswerk mit Steuerwerk und -leitungen



Das Steuerwerk besitzt bei der Eintakt-Implementierung keine Zustandsregister, ist also nur ein Schaltnetz

Erzeugung der Steuersignale



- Die 9 Steuersignale (7 plus 2 für ALUOp) sind in Abhängigkeit von den 6 Opcode-Bits zu setzen.
- Vorgehen: Wahrheitstabelle → Logische Schaltung

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

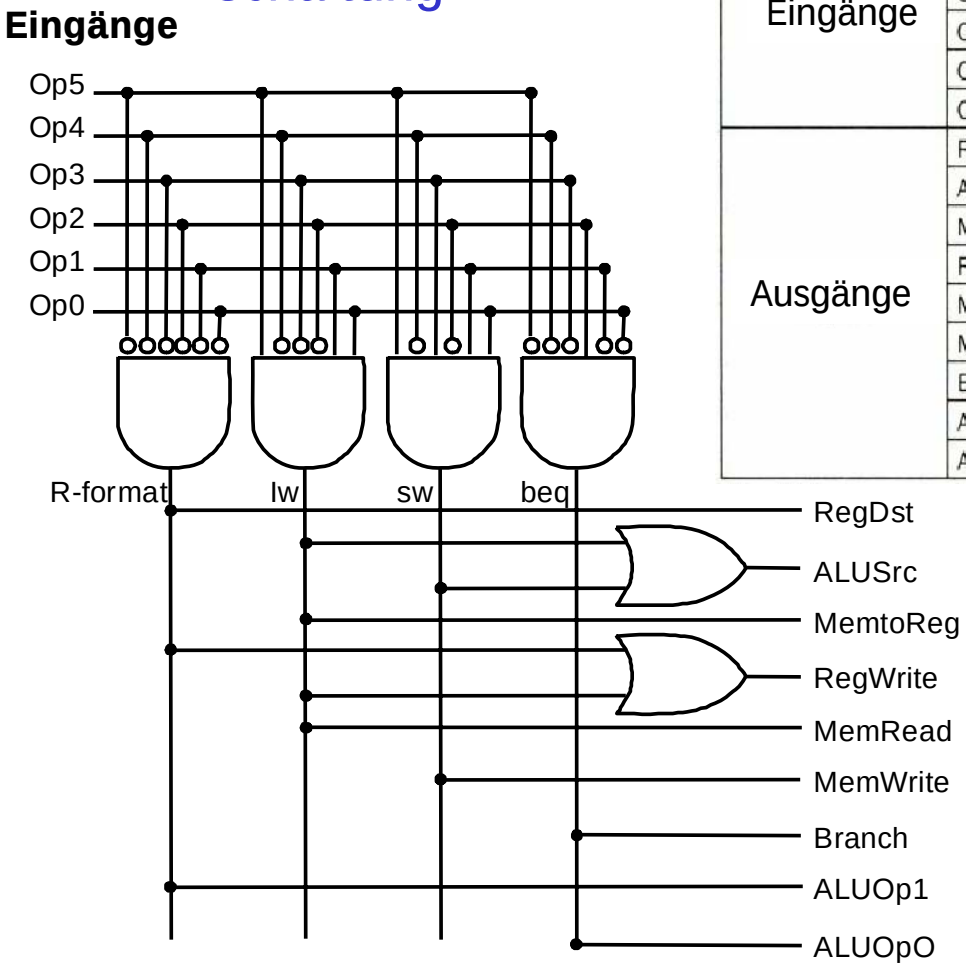
Steuersignale zur Aktivierung der zugeordneten Mikrooperation, R-Format siehe 8-15

Erzeugung der Steuersignale

Umcodierung des OPC in die Steuersignale

		R-format	lw	sw	beq
Eingänge	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Ausgänge	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Schaltung



Ausführung R-Format

z. B. add $t1, $t2, $t3

1. Ein Befehl wird aus dem Programmspeicher geladen und der Befehlszähler wird inkrementiert (PC+4).
2. Zwei Register $t2 und $t3 werden gelesen. Gleichzeitig werden die Steuersignale berechnet.
3. Die ALU bearbeitet die beiden Registerinhalte unter Auswertung des “funct”-Feldes der Befehl.
4. Das Ergebnis der ALU-Operation wird in das Register $t1 geschrieben.

ACHTUNG! Wir haben es hier mit einer Eintakt-Steuerung zu tun, d. h. ein Befehl wird vollständig in einem Taktzyklus ausgeführt.
Korrektes Timing wird einzig durch den Datenfluss im Operationswerk garantiert, der grob von links nach rechts (gem. Abb.) erfolgt.

Ausführung Load/Store

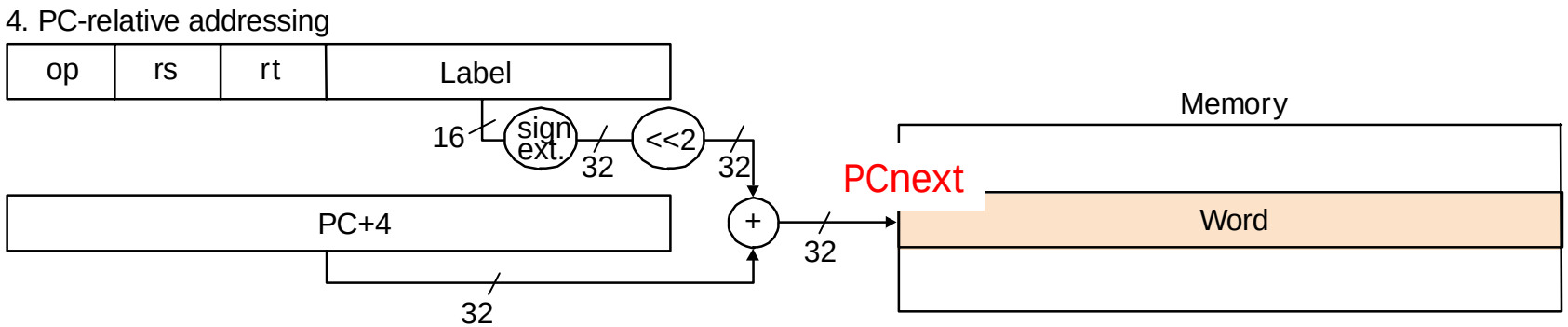
z. B. lw $t1, offset($t2)

1. Ein Befehl wird aus dem Programmspeicher geholt und der Befehlszähler wird inkrementiert (PC+4).
2. Das Register $t2 wird gelesen.
3. (Berechnung der effektiven Adresse) Die ALU berechnet die Summe des Wertes in $t2 und des vorzeichenerweiterten offset-Wertes.
4. Das ALU-Resultat wird zur Adressierung des Datenspeichers verwendet.
5. Die Daten aus dem Speicher werden in das Register $t1 geschrieben.

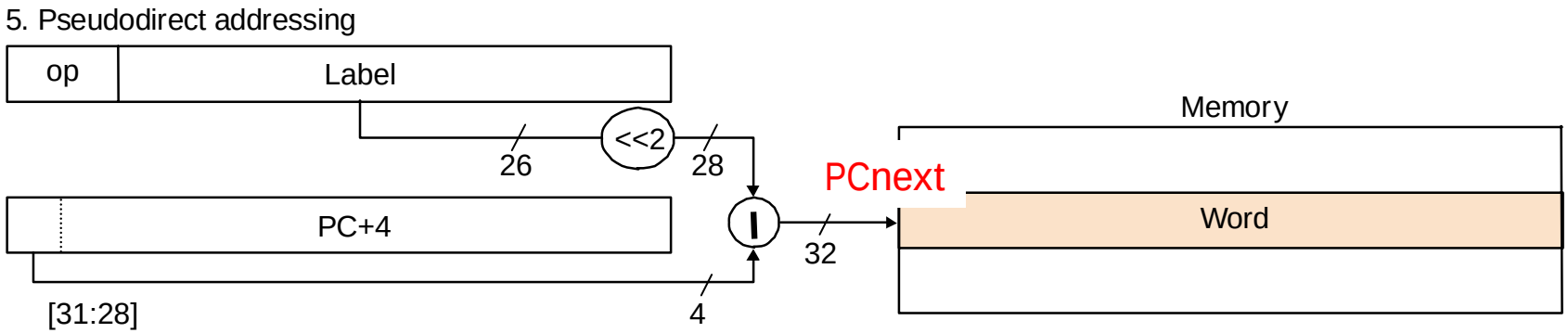
Ausführung Verzweigung

beq $t1, $t2, Label

1. Ein Befehl wird aus dem Programmspeicher geholt und der Befehlszähler wird inkrementiert (PC+4).
2. Zwei Register $t1 und $t2 werden gelesen.
3. Die ALU subtrahiert die Inhalte der beiden Register.
4. (Sprungzieladresse berechnen) Gleichzeitig werden die niederwertigen 16 Bit des Befehlswortes vorzeichenerweitert, um zwei Bits nach links geschoben auf (PC+4) addiert.
5. Der Wert der Zero-Leitung aus der ALU entscheidet, ob PCnext:=PC+4 oder PCnext:=(PC+4)+shl2(signext(Label)).



Ausführung Jump, Format J

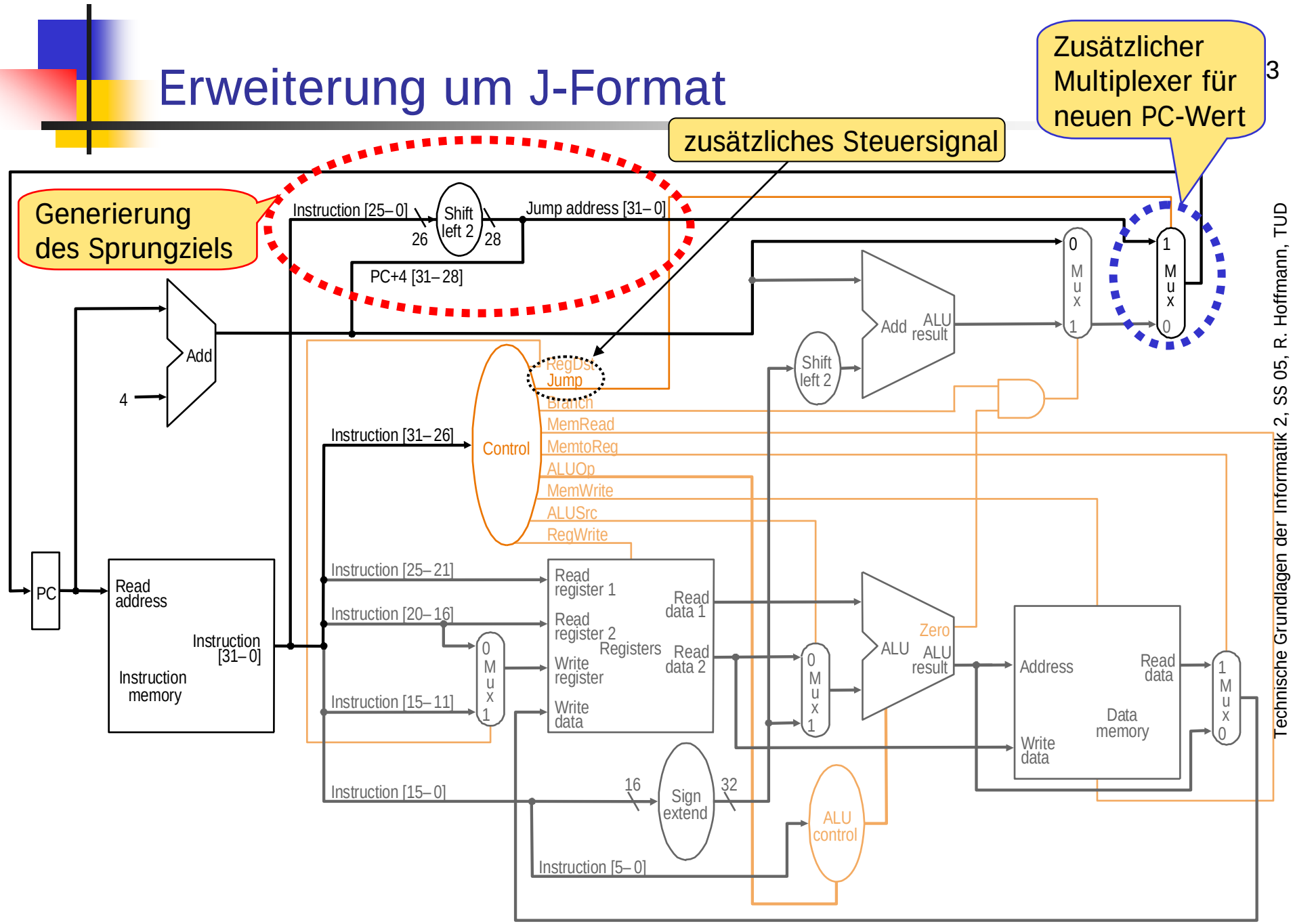


- „Jump“-Befehl (opcode = 2) ist ähnlich zur Verzweigung, aber mit anderer Berechnung des Sprungziels ohne Bedingung
- Sprungziel = höchstwertigen 4 Bits von (PC+4) konkateniert mit (L shift left 2)

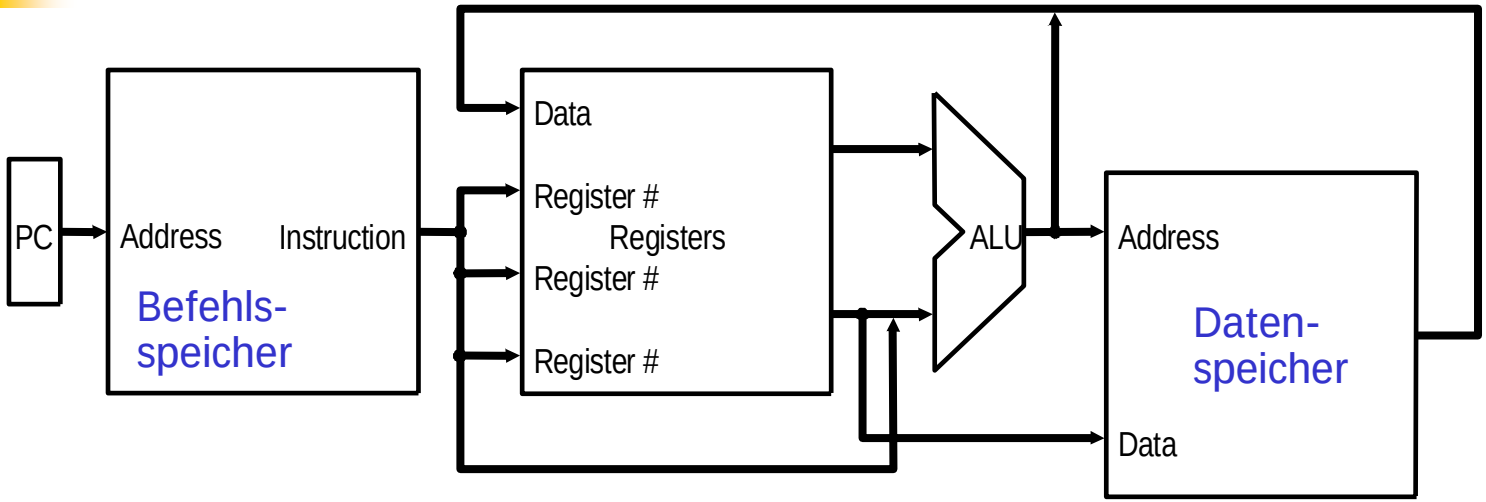
Program-Counter Next

		Berechnung des nächsten PC	Kommentar
Kein Sprungbefehl		PC ← PC+4	
Bedingter Sprung z. B. beq Label	if not Zero:	PC ← PC+4	
	if Zero:	PC ← (PC+4)+shl2(signext(Label))	relativer Sprung
Unbedingter Sprung j Label		PC ← (PC+4)[31:28] , shl2(Label)	Label = Instruction[25:0], absoluter Sprung innerhalb der Seite

Erweiterung um J-Format



Zeitverhalten der Eintakt-Implementierung



- Ausführungszeit wird für alle Befehle gleich gesetzt.
- Ausführungszeit wird durch die am längsten dauernde Operation bestimmt (insbesondere durch die von uns noch gar nicht betrachteten Gleitkomma-Befehle).
- Taktzykluszeit wird durch den kritischen Pfad (maximale Gesamtverzögerungszeit durch die Komponenten)
- Hardware-Einheiten müssen möglicherweise mehrfach vorhanden sein, da jede Einheit in einem Taktzyklus nur einmal verwendet werden kann.

→ Eintakt-Implementierung ist ineffizient!

Der Kritische Pfad

- bestimmt die kleinste mögliche Dauer des Taktes.
- Berechnung
  - Betrachte alle Wege von den Ausgängen der getakteten Speicherelemente zu allen Eingängen.
  - Die Länge eines Weges ist durch die Summe der Verzögerungszeiten durch die Schaltnetze definiert.
  - Bestimme den längsten Weg. Diese Zeit muß zwischen zwei positiven Taktflanken abgewartet werden.

- Der neue Wert am Ausgang eines Registers ist nach der positiven Flanke nach der Zeit $t_{cto} = t_{clocktoOutput}$ stabil. Dazu kommt ggf. noch eine Lesezeit $t_{regread}$, die durch die Leselogik (z. B. Multiplexer) bestimmt wird.
- Am Eingang vor einem Register muß der neue Wert mindestens vor der Set-Up-Zeit t_{setup} stabil sein.
- Die Zeit zum Lesen aus dem Datenspeicher bezeichnen wir mit $t_{readdata}$.
- Die Set-Up-Zeit zum Schreiben in den Datenspeicher bezeichnen wir mit $t_{writedata}$

Verzögerungszeiten

http://edascript.ims.uni-hannover.de/260_Emulation/folie_06.html

- Jedes Schaltungselement und jede Leitung verzögert die Signale, die sie durchlaufen. Besonders groß ist die zusätzliche Verzögerung, wenn das Signal von einem Chip zu einem anderen geführt wird. Die Leitungen außerhalb der Chips sind im Vergleich zu den internen Leitungen sehr lang und haben eine sehr große Kapazität. Beides erhöht die Verzögerungszeiten.
- Als einen Pfad bezeichnet man den Weg eines Signals durch eine Schaltung. Addiert man die Verzögerungen der einzelnen Schaltungskomponenten auf dem Pfad auf, erhält man die Gesamtverzögerung des Pfads. Derjenige mit der größten Verzögerungszeit wird als kritischer Pfad bezeichnet, weil durch ihn die maximale Taktfrequenz festgelegt wird, mit der man die Schaltung betreiben kann. Ist nämlich die Signalverzögerung auf einem Pfad bei gegebener Taktfrequenz größer als eine Taktperiode, kommt es sehr wahrscheinlich zu falschen Ergebnissen. Die Gefahr dafür ist beim kritischen Pfad am höchsten. Entsprechend muss darauf geachtet werden, dass der kritische Pfad und andere Pfade mit ähnlich großer Verzögerungszeit möglichst nicht über mehrere Chips führen, da sonst durch die zusätzliche Verzögerung die mögliche Taktfrequenz der Schaltung deutlich herabgesetzt wird.

Graph zur Ermittlung der kritischen Pfade

- Die folgenden Graphen wurden aus der Schaltung für die Eintakt-Implementierung gewonnen. Sie enthalten alle relevanten Schaltnetze zwischen den Speicherelementen.
- Befehlszähler **pc**, Registersatz **REGS** und der Datenspeicher **DataMem** sind jeweils zweimal gezeichnet, aber nur einmal vorhanden.
- Alle Schaltnetze (Programmspeicher **Pmem**, Recheneinheiten **plus4**, **ALU**, Multiplexer **mux**) besitzen Verzögerungszeiten.
- Shift und Konkatenation sind nur Verdrahtungen und sollen keine (signifikante) Verzögerung verursachen.
- Vergleichen Sie den Graphen (8-30) mit der bekannten Schaltung (8-29).
- Bemerkung: Für den Registerspeicher und Datenspeicher müssen asynchrone Schreibimpulse erzeugt werden, wenn diese Speicher geschrieben werden sollen.

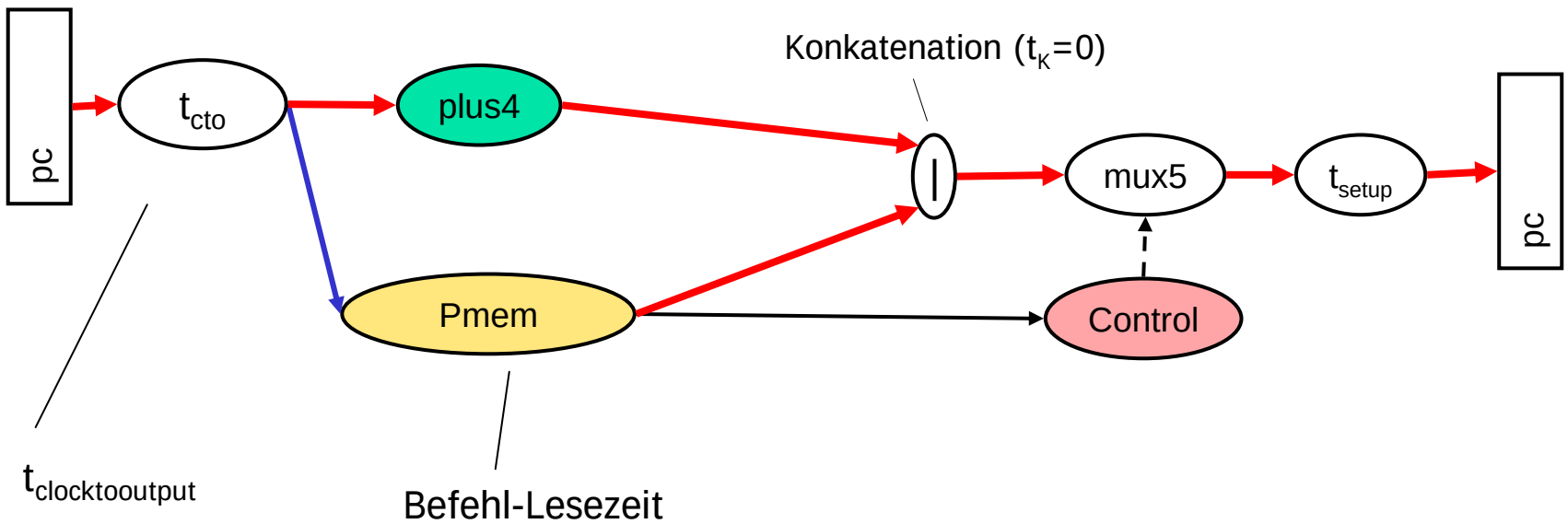
Kritischer Pfad für den Jump-Befehl

Bei einem Jump-Befehl sind die Wege durch den speziellen Graphen gegeben.

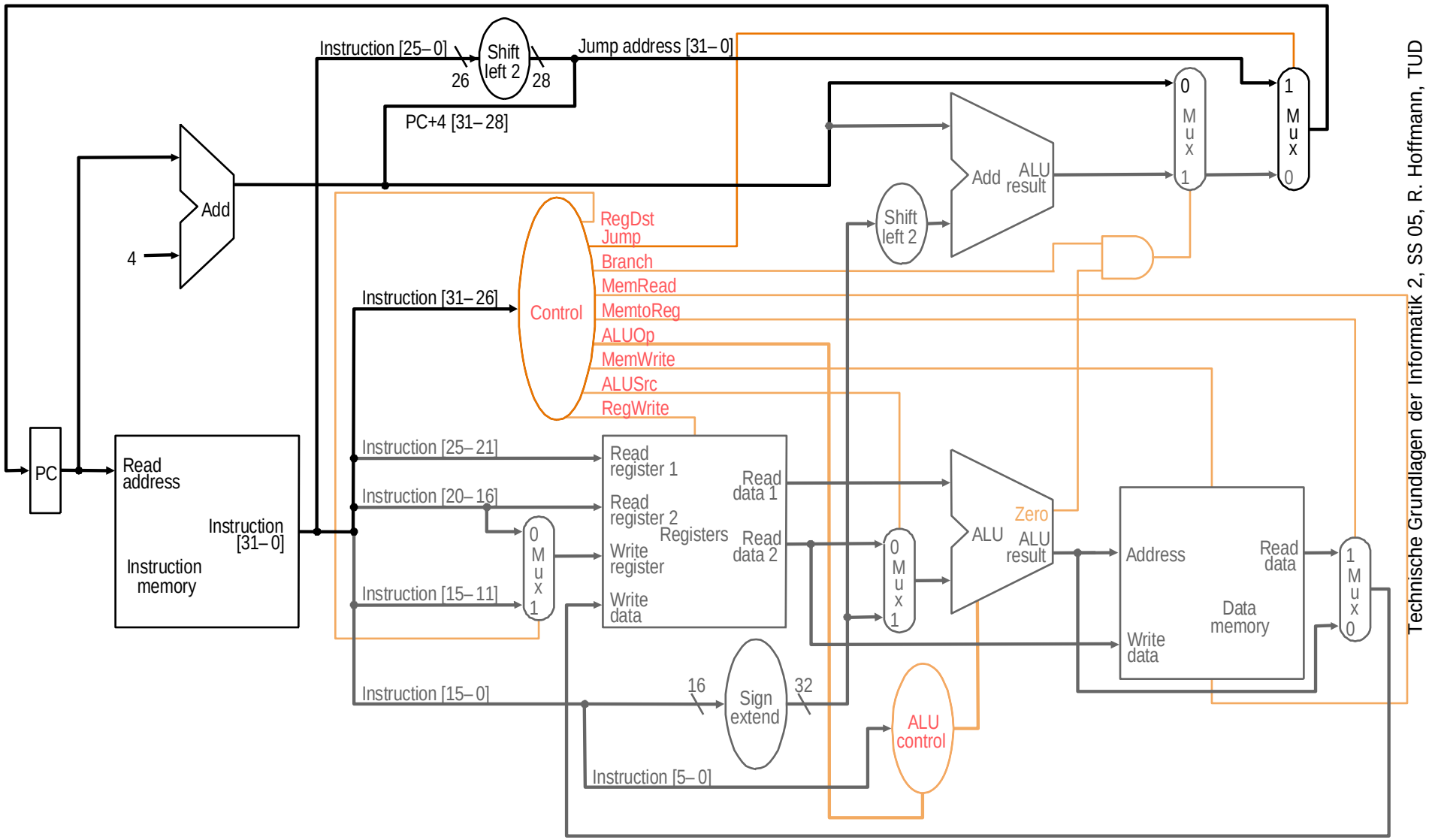
$T = t_{cto} + \max(t_{plus4}, t_{Pmem}, t_{Pmem} + t_{Control}) + t_{mux5} + t_{setup}$

$T = t_{cto} + \max(t_{plus4}, t_{Pmem} + t_{control}) + t_{mux5} + t_{setup}$

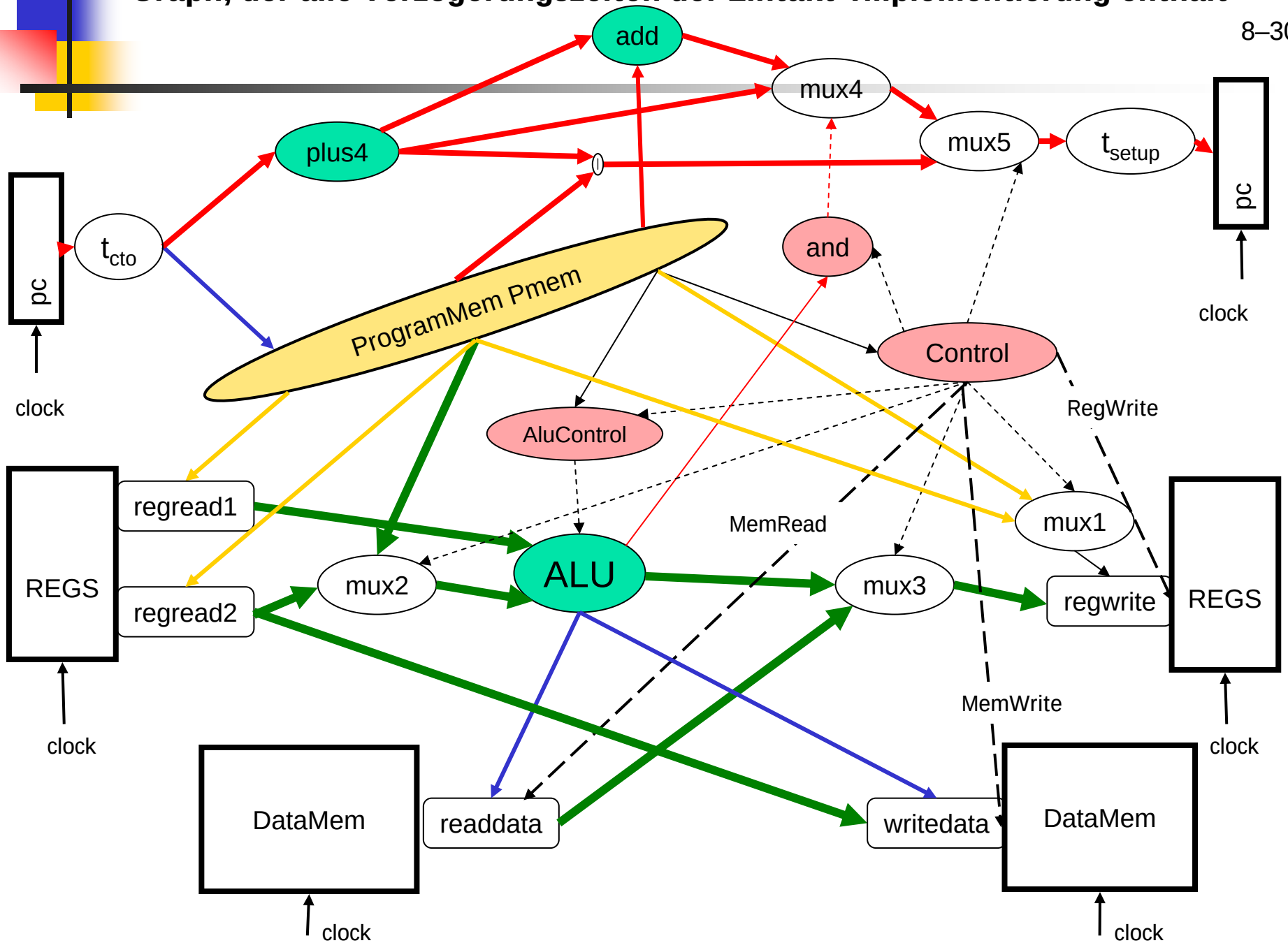
für $t_{cto}=2$ ns, $t_{setup}=1$ ns, $t_{plus4}=4$ ns, $t_{Pmem}=5$ ns, $t_{mux}=2$ ns, $t_{Control}=3$ ns
ergibt sich $T = 2 + \max(4, 5 + 3) + 2 + 1 = 13$ ns → $1/T = 76{,}9$ MHz



Eintakt-Implementierung (zum Vergleich)



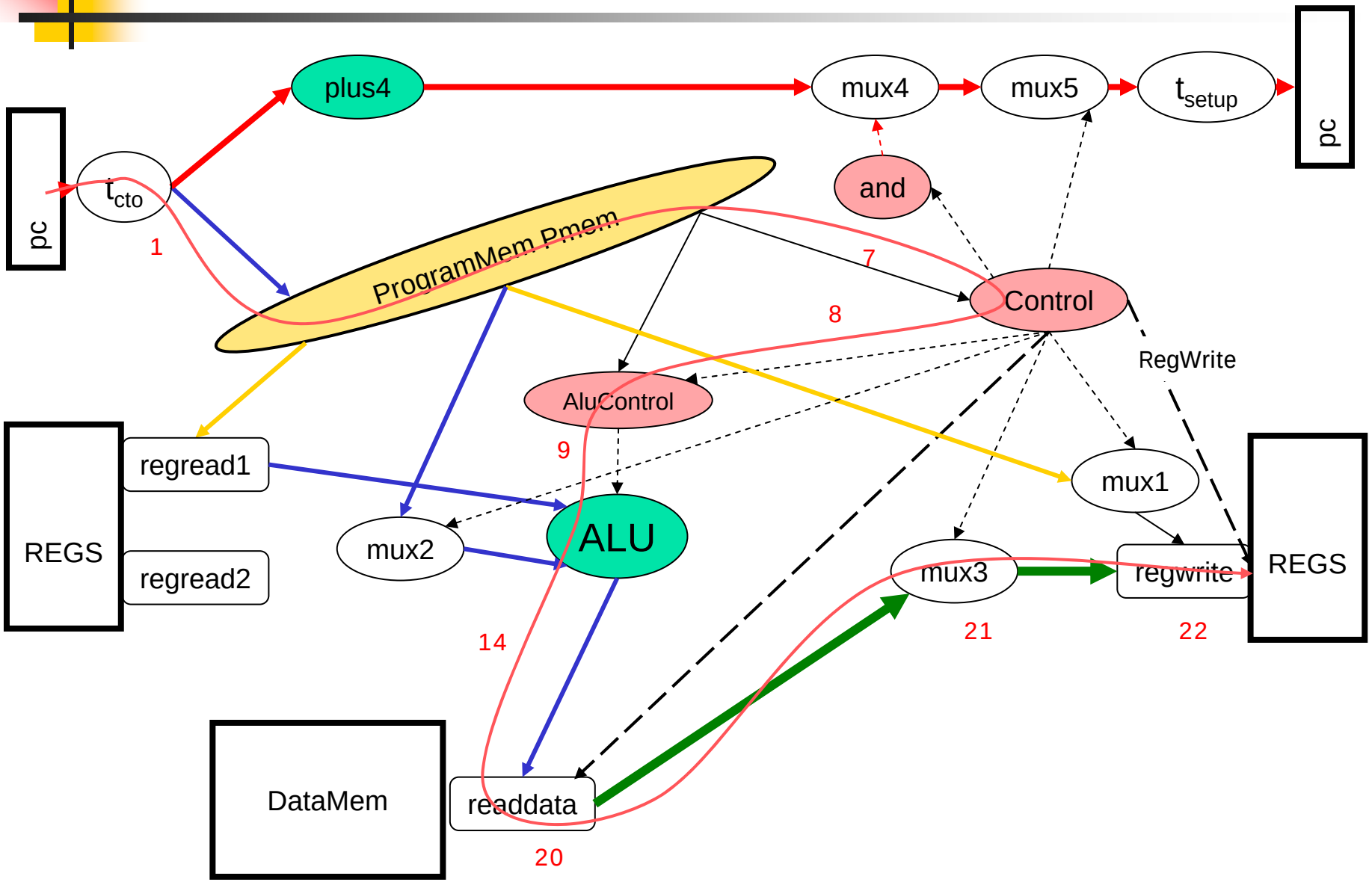
Graph, der alle Verzögerungszeiten der Eintakt-Implementierung enthält



Kritischer Pfad für Load

- Die folgende Folie zeigt für den Befehl **Load** die aktiven Speicherelemente und Schaltnetze. Dieser Graph ist nur der Teil des allgemeinen Graphen (8-30), der gerade aktiv ist.
- Daraus kann dann der kritische Pfad abgelesen werden.
- Beispiel
  - t_{cto}, t_{setup}, $t_{regread}$, $t_{regwrite}$, t_{mux}, $t_{control}$, $t_{AluControl}$, t_{and} =1
  - t_{Pmem}=6, $t_{readdata}$=6, t_{ALU}=5, t_{plus4}=2
  - Ermittlung des Kritischen Pfades durch Addition der Verzögerungszeiten und Markierung an den Kanten.
  - T = 22

Kritischer Pfad für Load

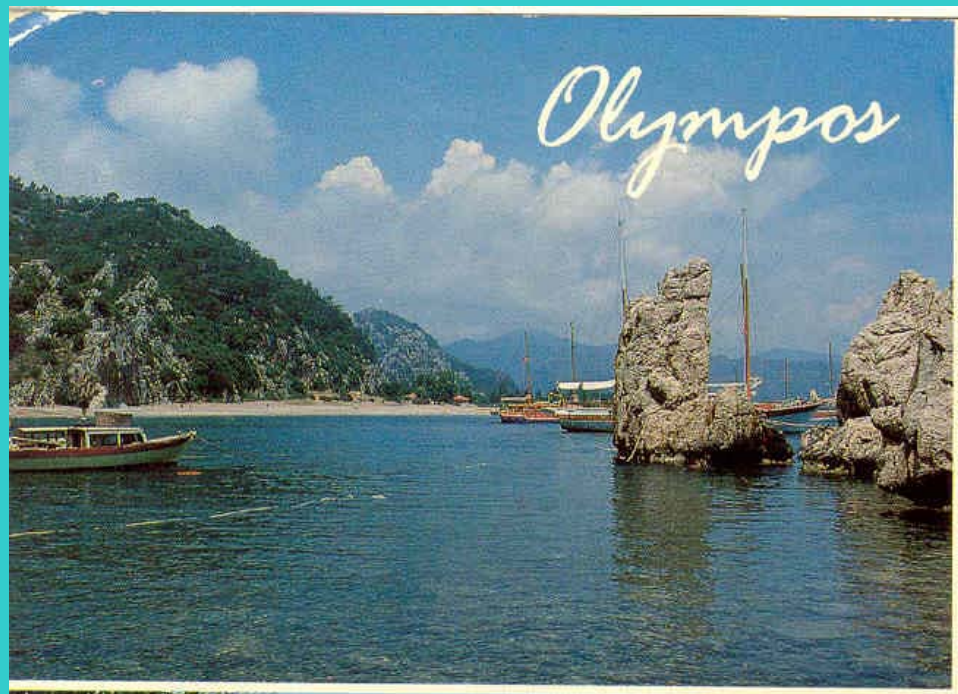


Ergebnis

- Der Kritische Pfad hängt ab von
  - dem Befehlstyp
  - kann sogar von den Daten abhängen (z. B. von der Zero-Bedingung bei dem Befehl beq)
  - ist insgesamt bei der Eintakt-Implementierung sehr lang
- Möglichkeiten zur Reduzierung der kritischen Pfadlänge
  - Mehrtakt-Implementierung (wie bei Dinatos)
  - Pipelining

Zusammenfassung

- Bei der Eintakt-Implementierung besteht das Steuerwerk nur aus einem Steuer-Schaltnetz
  - Inputsignale: Befehl und ALU-Output-Condition (Zero)
- Steuersignale
  - ALU-Steuersignale werden aus dem **OPC** und **funct** abgeleitet
  - Steuersignale zum Lesen und Schreiben der Register und des Datenspeichers
  - Steuersignale für die Multiplexer
- PCnext =
  - PC + 4
  - PC + relativeSprungadresse (if beq and cond true)
  - absoluteSprungadresse (bei Jump)
- Kritische Pfad
  - Summe der Verzögerungszeiten für den längsten Weg
  - abhängig vom Befehlstyp


Olympos