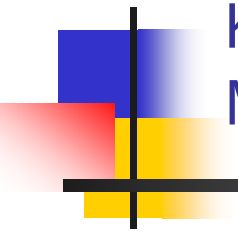


Kapitel 9 (4. Teil MIPS) : Mehrtakt-Implementierung

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: Die MIPS-Befehle sollen in mehreren Takten interpretiert werden, so wie schon im DINATOS
- Modifikation des Operationswerks
- Zustandsdiagramm zur Interpretation der Befehle
- Implementierungen
  - Hardware-Steuerung mit PLA
  - Mikroprogramm-Steuerwerk
    - Mikrogramm zur Interpretation
    - Mikroprogramm-Steuerwerk: Sequencing
- Ausnahmebehandlung

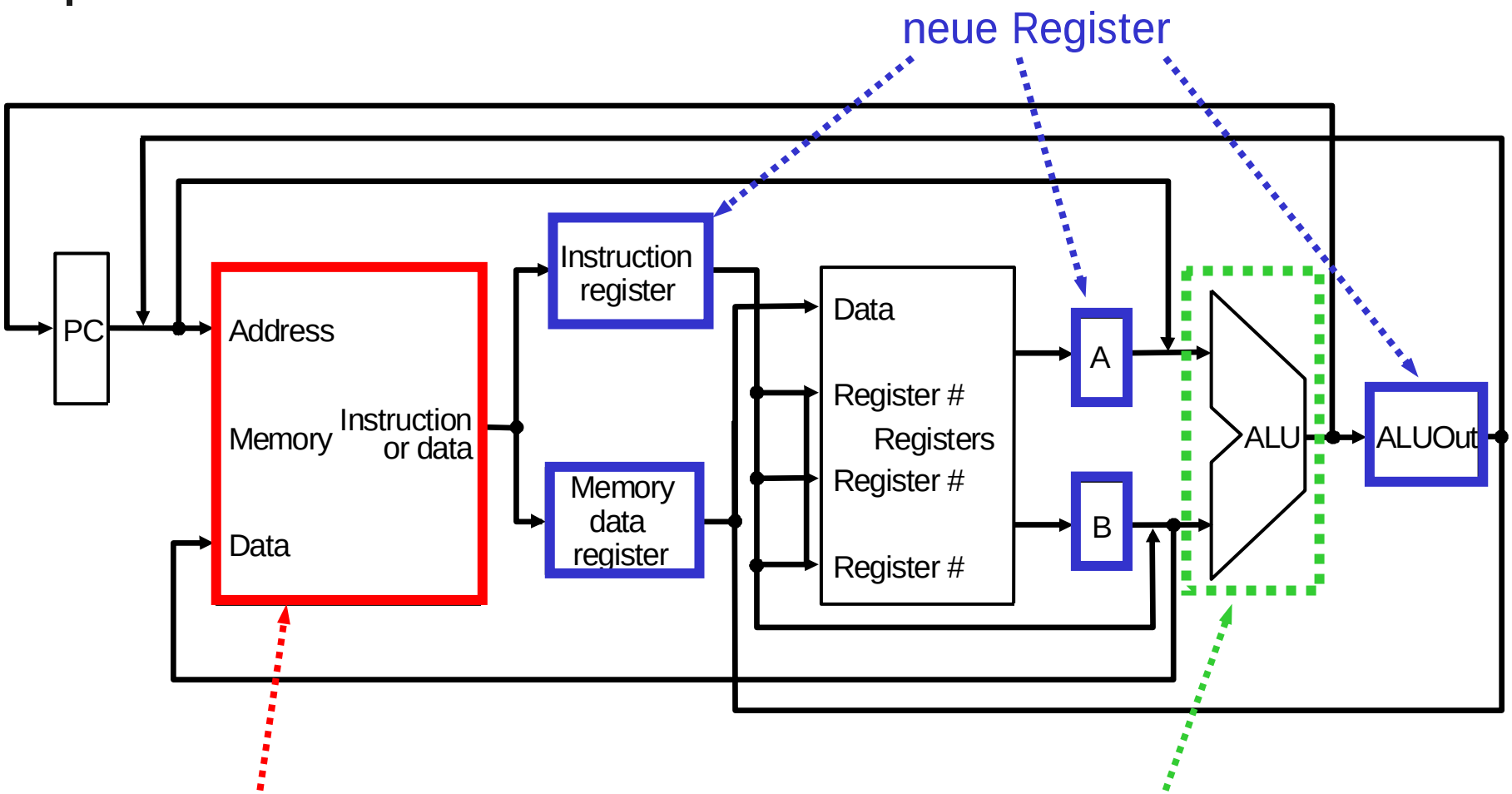
Mehrtakt-Operationswerk (multi-cycle datapath)

- In jedem Taktzyklus werden statt des ganzen Befehls nur Teile durchgeführt
- Konsequenz: Ein Befehl benötigt dann mehrere, aber kürzere (da kürzere kritische Pfade) Taktzyklen.
  - Hardware-Einheiten, die von einem Befehl mehrfach benötigt werden, müssen nicht unbedingt mehrfach vorhanden sein.
  - In einem Taktzyklus ist in der Regel nur ein Teil des Operationswerks aktiv.
  - Es müssen Mechanismen vorgesehen werden, um Informationen von einem Taktzyklus zum nächsten zu „retten“.

Mehrtakt-Operationswerk

- Ausgangspunkt ist die Aufteilung jedes Befehl in eine Folge von Mikrooperationen.
- Arbeitshypothese: In einem Taktzyklus können ausgeführt werden
  - ein Speicherzugriff oder
  - ein Zugriff auf das Registerwerk (entweder 2x Lesen oder 1x Schreiben) oder
  - eine ALU-Operation.
- Notwendige Hardware-Maßnahmen für Mehrtakt-Operationswerk:
  - Es müssen interne Register (Hilfsregister, Pufferregister, Pipelineregister) vorgesehen werden, die Informationen zwischen den Taktzyklen zwischenspeichern.
  - Wenn Hardware-Einheiten für mehr als eine Aufgabe innerhalb einer Befehlsausführung verwendet werden sollen, müssen Multiplexer an die Eingänge gesetzt werden.
- Steuersignale werden vom Befehl und Steuerzustand bestimmt.

Geändertes Operationswerk



Erklärung siehe Folie +1

Hilfsregister

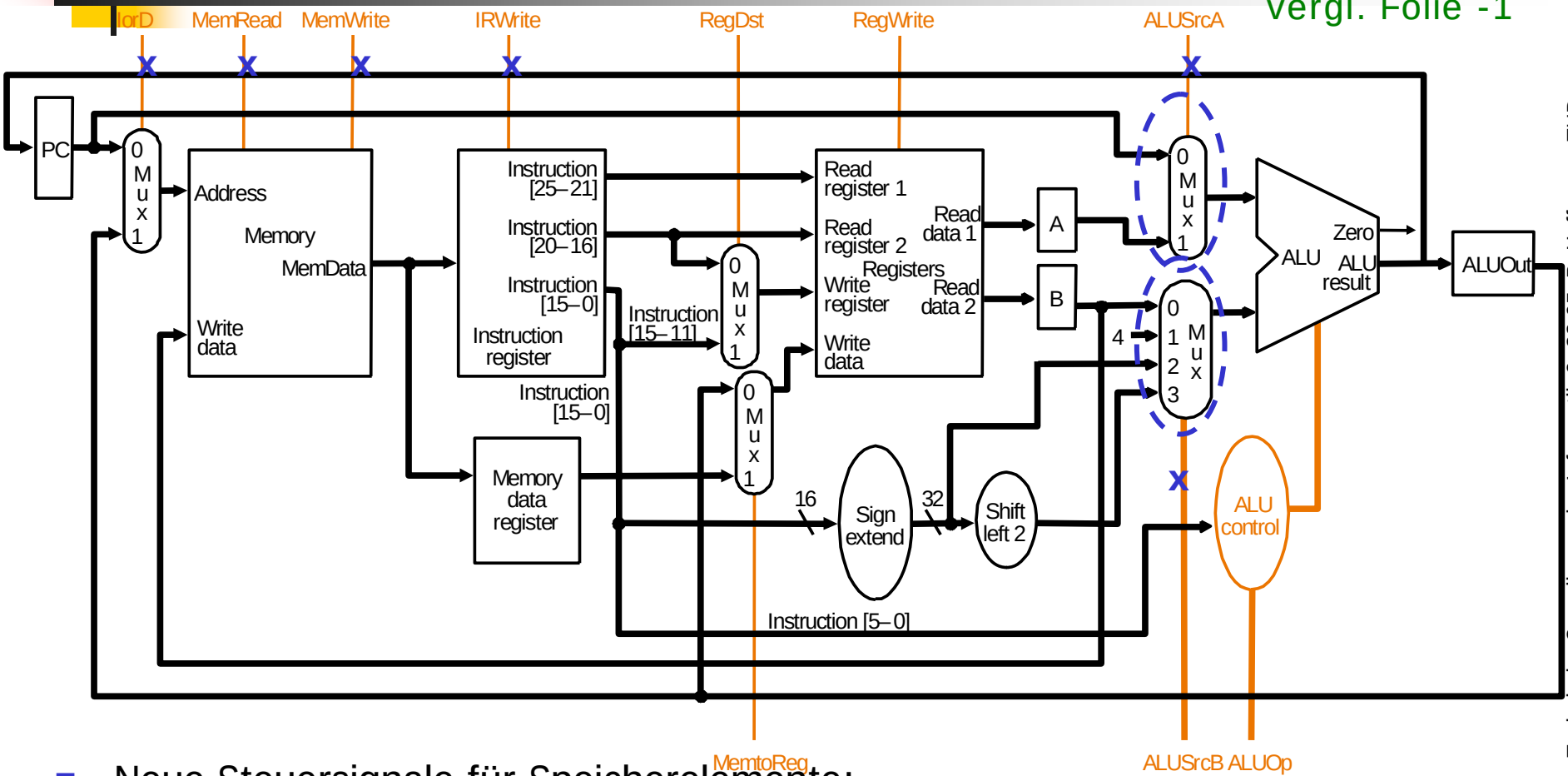
- Instruction Register (IR) speichert den geholten Befehl für dessen gesamte Ausführungszeit.
- Memory Data Register puffert die Daten, die aus dem Speicher gelesen werden.
- A- und B-Register puffern die aus den Registern gelesenen Operandenwerte.
- ALUOut-Register puffert die Ausgabe der ALU.
- Alle Register (außer IR) speichern Daten immer zwischen zwei aufeinander folgenden Takten, d. h. kein explizites „Write“-Steuersignal nötig.
- siehe Folie -1

Zusätzliche Multiplexer

- ALU
  - **Grund**: Sowohl die Adreßberechnungen als auch die eigentliche Datenoperation laufen jetzt über dieselbe ALU.
  - Multiplexer vor dem ersten ALU-Eingang, Auswahl von
    - (0) PC (um PC+4 zu bilden)
    - (1) A-Register (erster Registeroperand)
  - Multiplexer vor dem zweiten ALU-Eingang von zwei auf vier Eingänge erweitert
    - (0) B-Register (zweiter Registeroperand)
    - (1) Konstante 4, um PC+4 berechnen zu können
    - (2) signext(16-Bit-Offset) (für Basisadressierung bei Load und Store)
    - (3) signext(16-Bit-Offset) shift left 2 (für PC-relativen bedingten Sprung)
- Speicheradresseingang, Auswahl von
  - (0) PC (enthält die nächste Befehlsadresse)
  - (1) ALUOut (enthält berechnete Sprungzieladresse)
- Einsparungen gegenüber Eintakt-Operationswerk: zwei Addierer, ein Speicher
- siehe Folie +1

Mehrtakt-Operationswerk: Steuersignale

vergl. Folie -1



- Neue Steuersignale für Speicherelemente:
  - Schreib-Signale für PC, Speicher und Register / Lese-Signal für Speicher
- Neue Steuersignale für Multiplexer
  - 1-Bit-Signale für Speichereingangs- und 1. ALU-Eingangsmultiplexer
  - 2-Bit-Signal für den 2. ALU-Eingangsmultiplexer
- ALU-Steuerlogik wie vom Eintakt-Operationswerk her bekannt

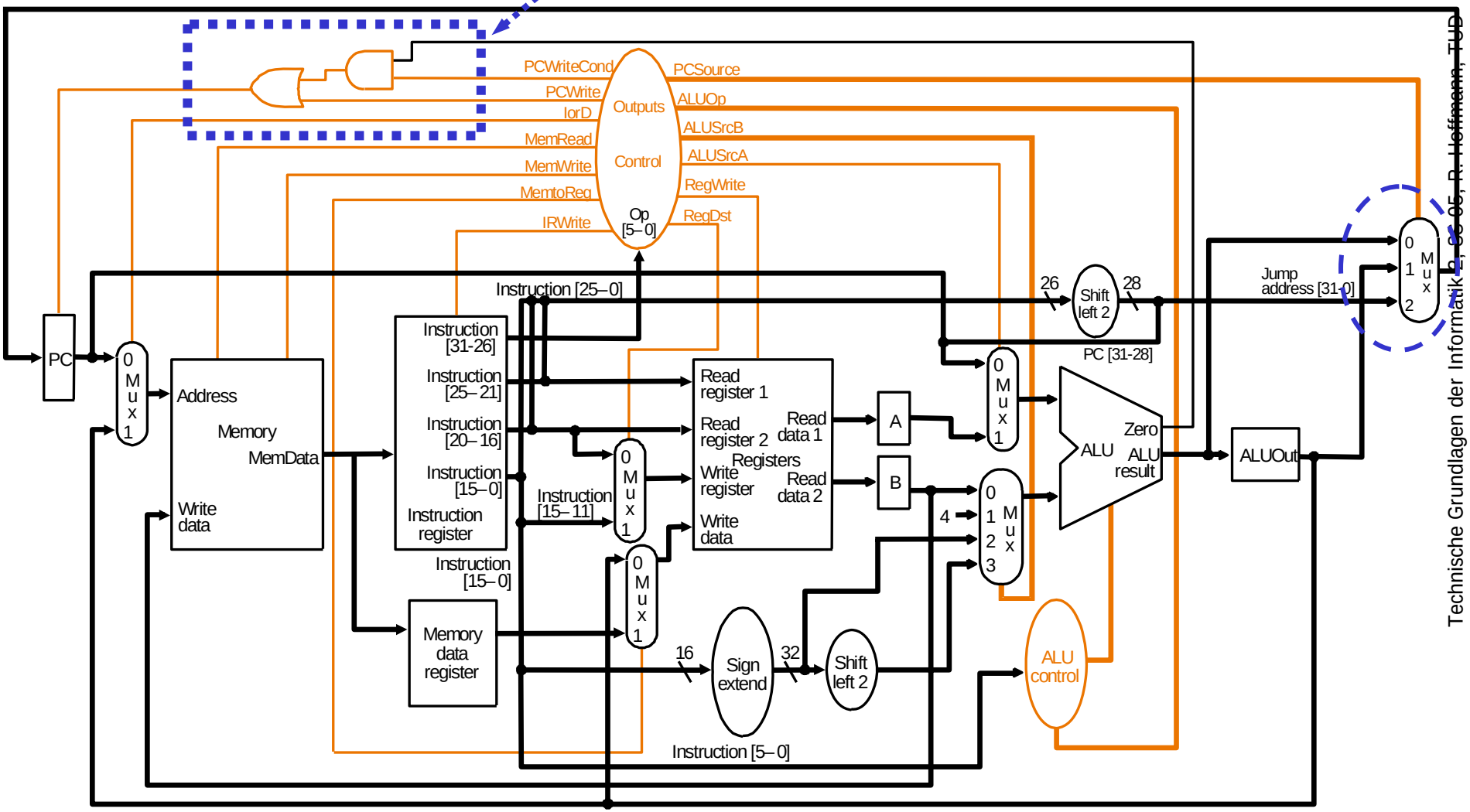
Erweiterungen für Verzweigungsbefehle

- Der neue Wert des Befehlszählers PC kann von drei Quellen stammen:
  - (0) PC ← ALUAusgang (= PC+4)
  - (1) PC ← RegisterALUOut
    (enthält berechnete Sprungzieladresse
    PC+(sext(IR[15:0])<<2) bei beq)
  - (2) PC<={PC[31:28], IR[25:0]<<2} bei jump
- Zusatzlogik
  - Wenn PC unbedingt geändert werden soll, dann sendet das Steuerwerk PCWrite
  - Wenn PC nur in Abhängigkeit vom ALU-Zero-Ausgang geändert werden soll, dann sendet das Steuerwerk PCWriteCond.
- s. Folie +1

Mehrtakt-Operationswerk mit Steuerwerk

Zusatzlogik

s. Folie -1



Bedeutung der 1-Bit Steuersignale

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None	The output of the memory is written into the IR.
PCWrite	None	The PC is written; the source is controlled by PCSource.
PCWriteCond	None	The PC is written if the Zero output from the ALU is also active.

Bedeutung der 2-Bit Steuersignale

Signal name	Value	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25–0] shifted left 2 bits and concatenated with PC + 4[31–28]) is sent to the PC for writing.

Implementierung einer Mehrtakt-Steuerung

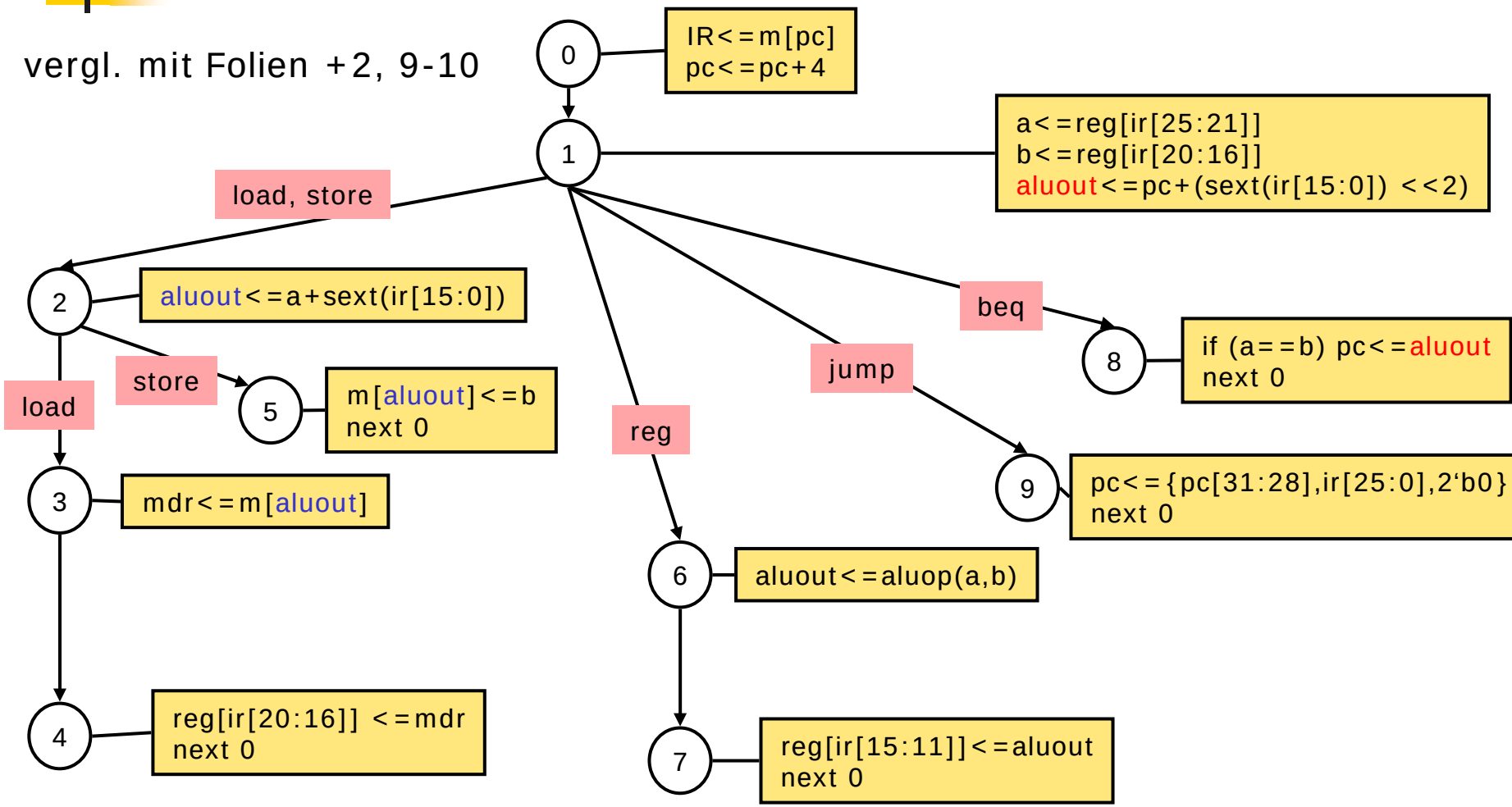
- Zerlegung jeder Befehlsausführung in eine Folge von Mikrooperationen
  - Ziel dabei: Ausbalancierung des Aufwands zur Ausführung einer Aktion in einem Taktzyklus (dadurch Minimierung der Taktzyklusdauer)
- Entwurf eines Steueralgorithmus, welcher die Mikrooperationen in der gewünschten Reihenfolge aktiviert.
- Zuordnung von Steuersignalen zu den einzelnen Mikrooperationen.

Steueralgorithmus

- Was ist zu tun? Es ist ein Steueralgorithmus zu entwerfen (am besten in Form eines **Zustandsdiagramms**), das
  - die Interpretation der Befehle durch Mikrooperationen beschreibt, wie beim DINATOS
- Der Steueralgorithmus wird anschließend implementiert (siehe Kap. 4) durch ein
  - Hardware-Steuerwerk (z. B. Schritt-Steuerwerk) oder durch ein
  - Mikroprogramm-Steuerwerk
- Das folgende Zustandsdiagramm interpretiert die Befehle
  - R(egister)
  - Load
  - Store
  - Jump
  - beq

Zustandsdiagramm

vergl. mit Folien +2, 9-10



Durchzuführende Mikrooperationen

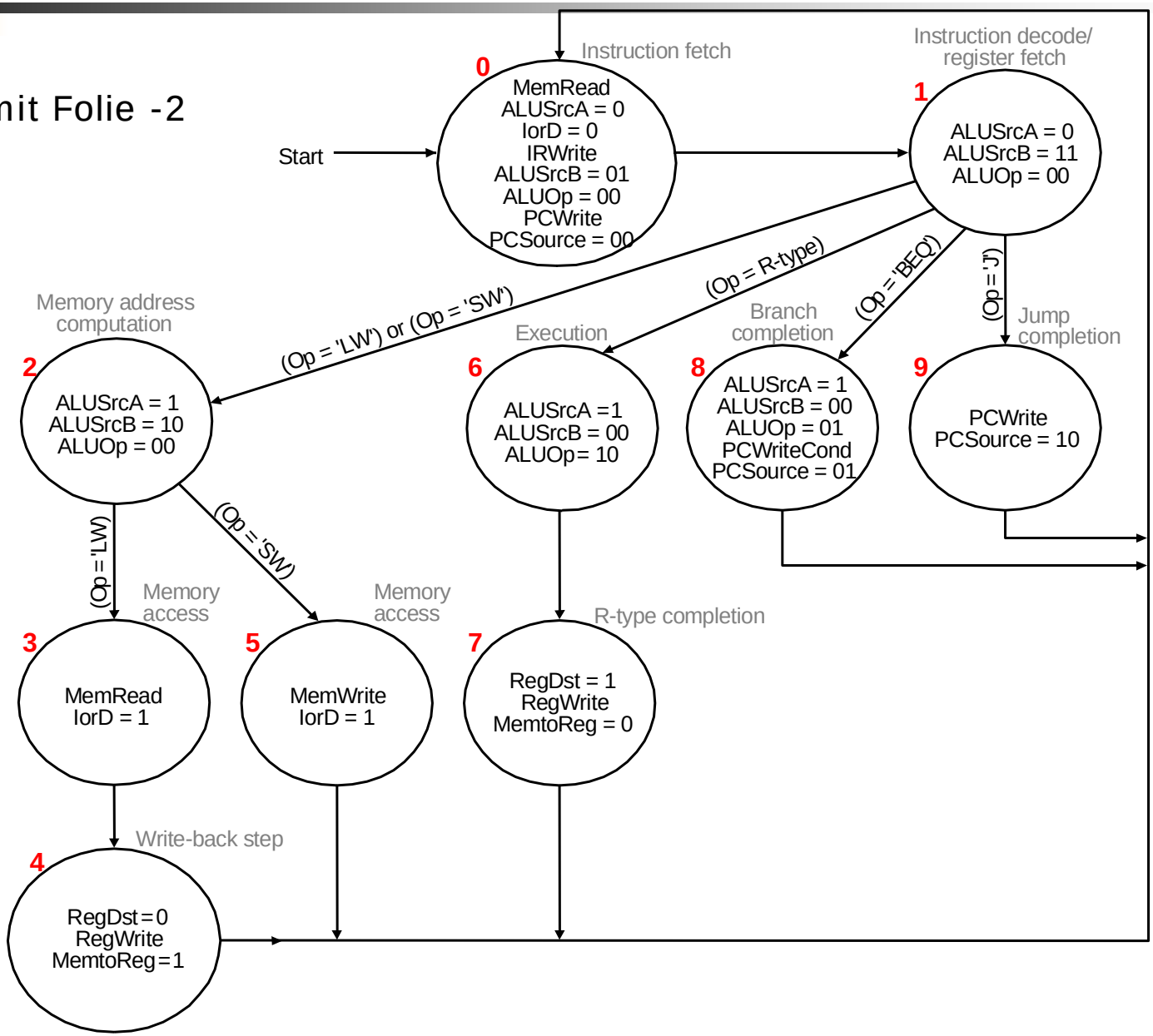
- Befehle benötigen zwischen 3 und 5 Zyklen.

Schrittname	Aktion für R-Typ Instruktionen	Aktion für Speicher-zugriffsinstruktionen	Aktion für Verzweigung	Aktion für Sprünge
Instruktionszugriff	IR = Memory[PC]			
	PC = PC + 4			
Instruktion Dekodierung/ Registerzugriff	A = Reg [IR[25-21]]			
	B = Reg [IR[20-16]]			
	ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Ausführung, Adress-berechnung, Verzweigung/ Sprungabschluß	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
Speicherzugriff oder R-Typ Abschluß	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] oder Store: Memory [ALUOut] = B		
Speicherzugriffabschluß		Load: Reg[IR[20-16]] = MDR		

vergl. Zustandsdiagramm

Ablauf mit Steuersignalen

vergl. mit Folie -2



Zuordnung von Steuersignalen zu den Mikrooperationen

vergl. Folie 9-10

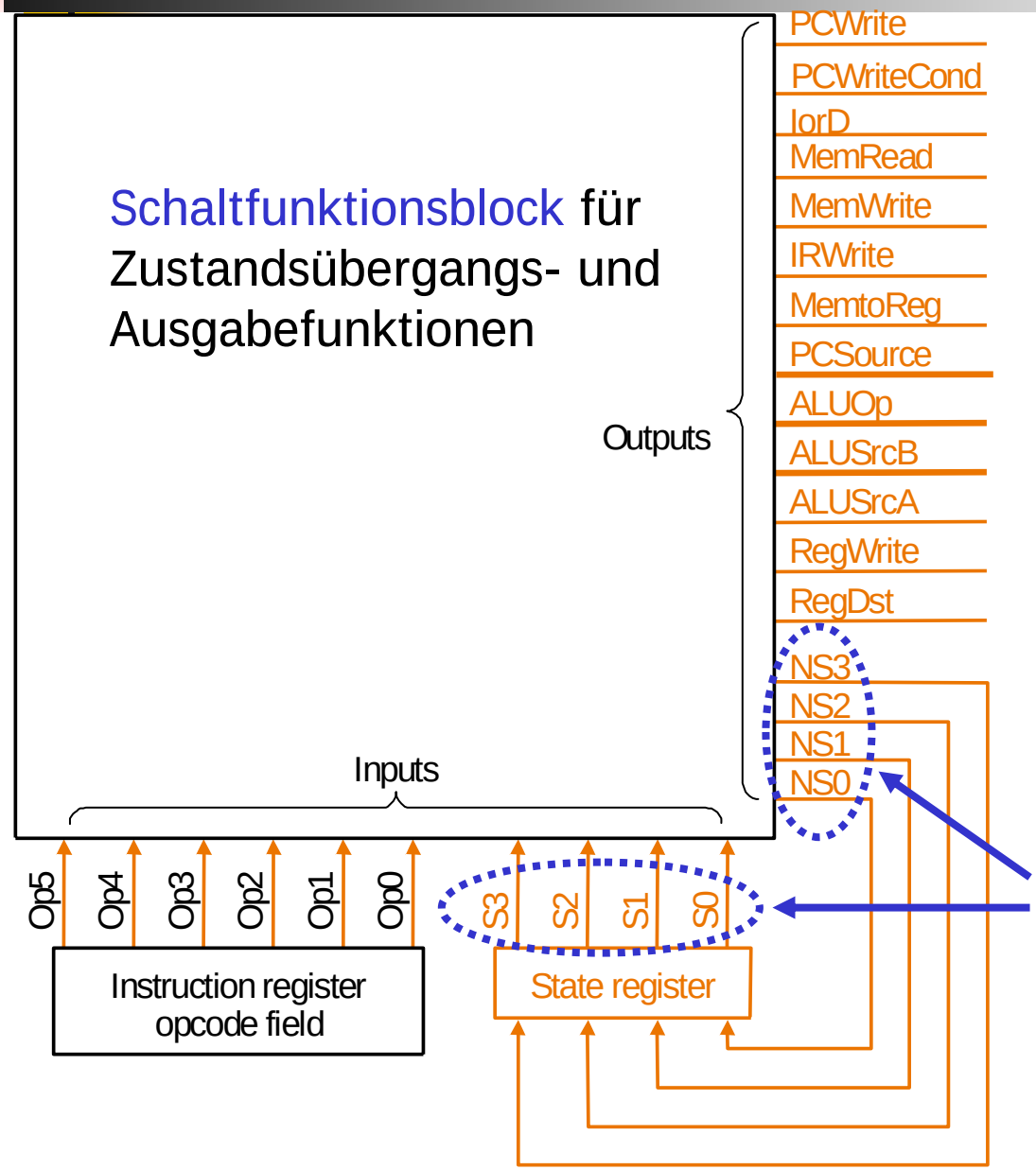
```
IR<=m[pc]
pc<=pc+4
```

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

```
a<=reg[ir[25:21]]
b<=reg[ir[20:16]]
aluout<=pc+(sext(ir[15:0]) <<2)
```

ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00

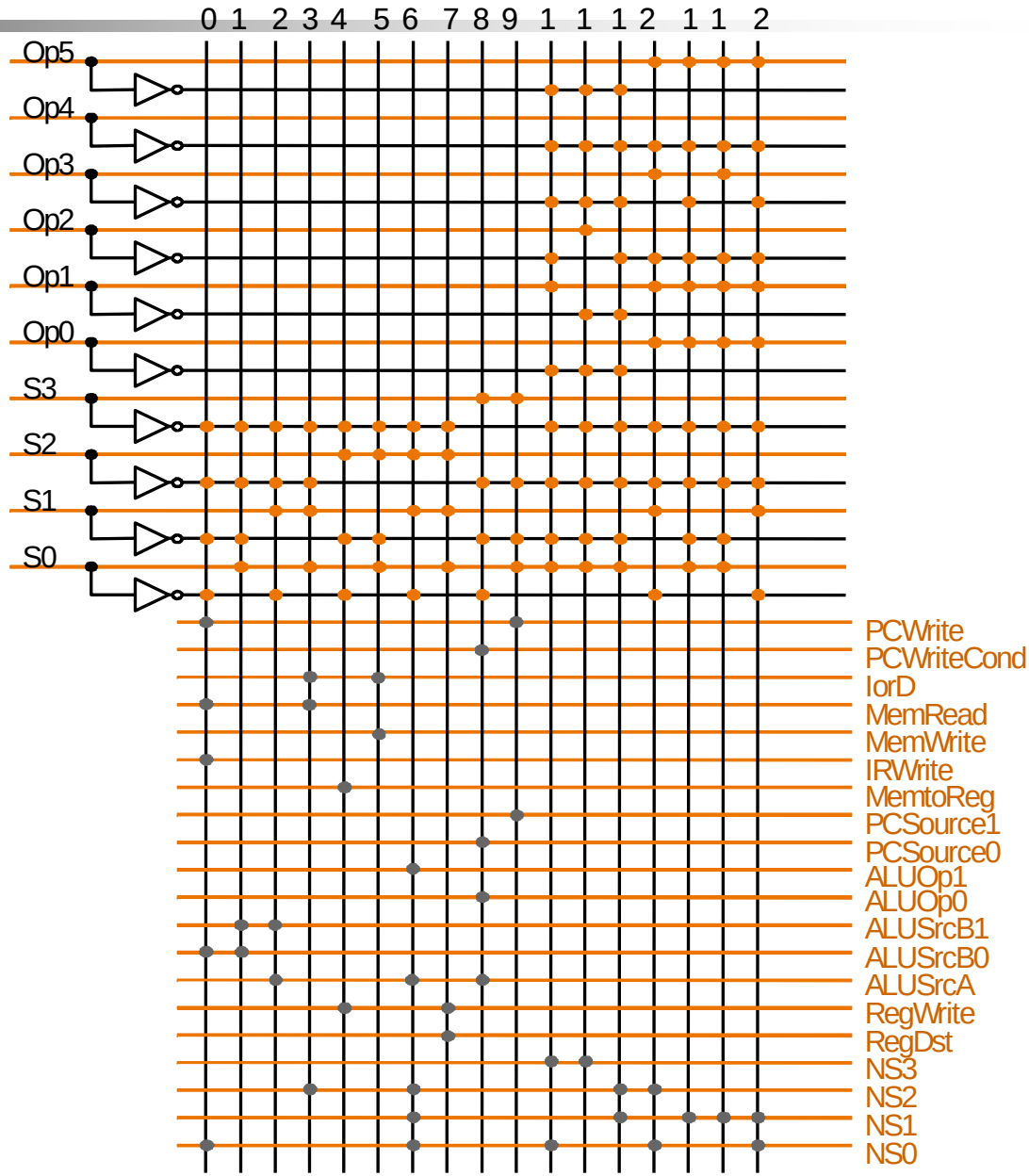
Prinzipielle Implementierung



4 Bits zur Codierung der 10 Zustandsnummern (NS: next state)

Implementierung des Funktionsblocks

- durch ein PLA (Programmable Logic Array)



UND

ODER

Diskussion

- Die Implementierung der Mehrtakt-Steuerung ist für eine einfache Teilmenge der MIPS-Instruktionen einfach und direkt möglich.
- Aber: Voller MIPS-Befehlssatz umfaßt mehr als 100 Instruktionen, die 1 bis über 20 Taktzyklen erfordern.
- Andere Instruktionssätze (z. B. Intel 80x86) haben weit mehr Adressierungsmodi und Opcodes, so daß ein Zustandsdiagramm mit entsprechend vielen Zuständen nötig wäre.

Alternative: Mikroprogramm-Steuerwerk

- Wir fassen die Steuersignale zu einem Steuerbefehl zusammen, das vom Steuerwerk zum Operationswerk gesendet wird und die auszuführenden Mikrooperationen bestimmt.
- Ein Mikrobefehl besteht aus dem Steuerbefehl und Informationen für den nächsten Zustand des Automaten, die zur Berechnung des Folgebefehls dienen (sequencing, Ablaufkontrolle).
- Die Mikrobefehle könne symbolisch dargestellt werden, ähnlich wie Maschinenbefehle.
- Eine Folge von Mikrobefehlen ergibt ein Mikroprogramm. Es kann – wie hier – zur Interpretation der Maschinenbefehle benutzt werden.

Definition eines Mikrobefehlsformats

- Definition von
  - Anzahl, Größe und Anordnung der Felder eines Mikrobefehls und
  - welche Steuersignale von welchem Feld beeinflußt werden.
- Anforderung: Das Format sollte die Auslösung nicht verträglicher Mikrooperationen verhindern oder erschweren (z. B. gleichzeitig pc ← pc+4 und pc ← Label).
- Ansatz: Jedes Feld des Mikrobefehls ist für eine Menge von alternativen Operationen verantwortlich oder steuert einen Multiplexer.
- Bei der Definition der Felder müssen die Gegebenheiten des Operationswerks berücksichtigt werden.

Definition eines Mikrobefehlsformats

- Ergebnis: Jeder Mikrobefehl enthält diese 7 Felder:

Field name	Function of field
ALU control	Specify the operation being done by the ALU during this clock; the result is always written in ALUOut.
SRC1	Specify the source for the first ALU operand.
SRC2	Specify the source for the second ALU operand.
Register control	Specify read or write for the register file, and the source of the value for a write.
Memory	Specify read or write, and the source for the memory. For a read, specify the destination register.
PCWrite control	Specify the writing of the PC.
Sequencing	Specify how to choose the next microinstruction to be executed.

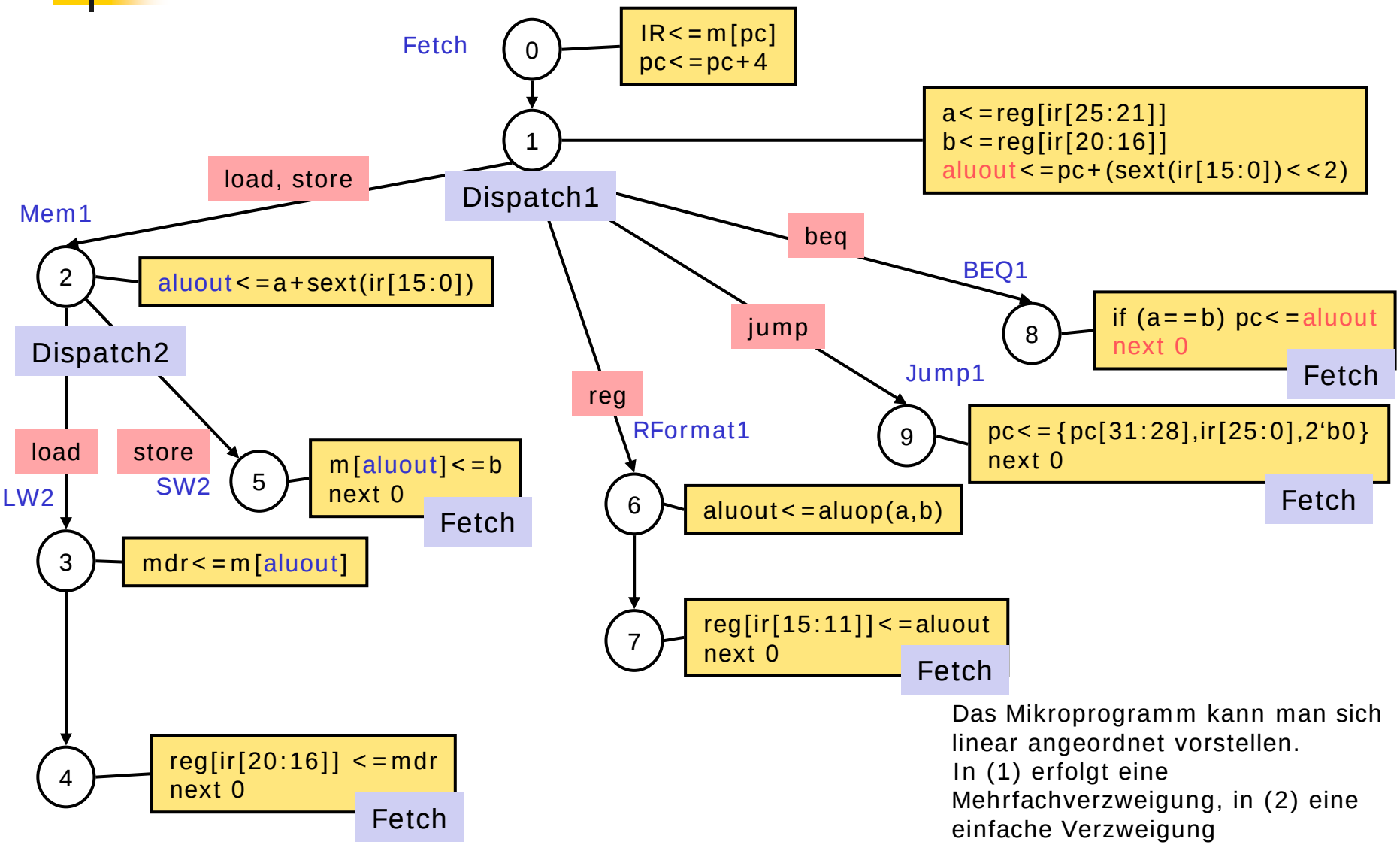
Die ersten 6 Felder steuern das Operations-werk.

- Das siebte Feld steuert den Ablauf der Mikrobefehle.
- Mikrobefehle werden in ROM realisiert und sind somit adressierbar.
- Bei „Sequencing“ drei Möglichkeiten zur Auswahl des Folgebefehls:
  - **`Seq`**: Der nächste Mikrobefehl ist die im Speicher folgende (default).
  - **`Fetch`**: Der nächste Mikrobefehl beginnt einen neuen Ausführungszyklus für eine MIPS-Instruktion.
  - **`Dispatch i`**: Der nächste Mikrobefehl wird durch die Steuereingänge definiert. Zwei Dispatch-Tabellen legen die konkreten Sprungziele ausgehend vom Zustand 1 bzw. vom Zustand 2 fest.

Werte der 7 Felder und Auswirkungen

Feldname	Wert	aktive Signale	Kommentare
	Add	ALUOp = 00	Veranlasse ALU zu addieren.
ALU control	Subt	ALUOp = 01	Veranlasse ALU zu subtrahieren; implementiert Vergleich für
			Verzweigungen.
	Func code	ALUOp = 10	Bestimme ALU Steuerung mit Funktionskod der Instruktion.
SRC1	PC	ALUSrcA = 0	Nimm PC als erste ALU Eingabe.
	A	ALUSrcA = 1	Register A ist erste ALU Eingabe.
	B	ALUSrcB = 00	Register B ist zweite ALU Eingabe.
SRC2	4	ALUSrcB = 01	Nimm 4 als zweite ALU Eingabe.
	Extend	ALUSrcB = 10	Nimm Ausgabe der Vorzeicheneinheit als zweite ALU Eingabe.
	Extshft	ALUSrcB = 11	Verwende Ausgabe der verschiebe-um-zwei Einheit als zweite ALU Eingabe.
	Read		Lies zwei Register mit den rs und rt Feldern von IR als Registernummern
			und speichere die Daten in die Register A und B.
	Write ALU	RegWrite,	Schreibe in Register mit dem rd Feld des IR als Registernummer und
Register		RegDst = 1,	dem Inhalt von ALUOut als Daten.
control		MemtoReg = 0	
	Write MDR	RegWrite,	Schreibe in Register mit dem rt Feld des IR als Registernummer und
		RegDst = 0,	dem Inhalt von MDR als Daten.
		MemtoReg = 1	
	Read PC	MemRead,	Lies den Speicher mit dem PC als Adresse; schreibe das Ergebnis ins IR (und
		IorD = 0	das MDR).
Memory	Read ALU	MemRead,	Lies den Speicher mit der ALUOut als Adresse; schreibe das Ergebnis ins MDR.
		IorD = 1	
	Write ALU	MemWrite,	Schreibe in den Speicher mit der ALUOut als Adresse, Inhalt von B als
		IorD = 1	Daten.
	ALU	PCSource = 00	Schreibe die Ausgabe der ALU in den PC.
		PCWrite	
PC write control	ALUOut-cond	PCSource = 01,	Bei gesetzter Zero Ausgabe der ALU schreibe den Inhalt des Registers
		PCWriteCond	ALUOut in den PC.
	jump address	PCSource = 10,	Schreibe die Sprungadresse aus der Instruktion in den PC.
		PCWrite	
	Seq	AddrCtl = 11	Wähle die nächste Mikroinstruktion sequentiell.
Sequencing	Fetch	AddrCtl = 00	Gehe zur ersten Mikroinstruktion, um eine neue Instruktion zu beginnen.
	Dispatch 1	AddrCtl = 01	Verteile mittels des ROM 1.
	Dispatch 2	AddrCtl = 10	Verteile mittels des ROM 2.

Mikroprogramm als Graph



Das Mikroprogramm kann man sich linear angeordnet vorstellen.
In (1) erfolgt eine Mehrfachverzweigung, in (2) eine einfache Verzweigung

Mikroprogramm, symbolisch

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Erste Zeile:

- ALU control, SRC1, SCR2: Berechnung von ALUout=PC + 4
- Memory: Laden des Befehls ins Befehlsregister IR<=m[PC]
- PCWrite control: bewirkt, daß PC<=ALUout
- Sequencing: Gehe zum nächsten Mikrobefehl

Zweite Zeile:

- ALU control, SRC1, SCR2: (PC + Vorzeichenerweiterung(IR[15-0]) <<2) in ALUOut speichern
- Register control: Lesen von $rs und $rt und Platzieren der Daten in A und B
- Sequencing: Verwenden von Dispatch-Tabelle 1 zur Wahl des nächsten Mikrobefehlsadresse (Sprung zu Mem1, Rformat1, BEQ1 oder JUMP1)

Mikroprogramm

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch

Dritte Zeile:

- ALU control, SRC1, SCR2: Speicheradresse (Register(rs)+Vorzeichenerweiterung(IR[15-0])) errechnen und in ALUOut speichern
- Sequencing: Verwenden von Dispatch-Tabelle 2 zur Wahl der nächsten Mikrobefehlsadresse (Sprung zu LW2 oder SW2)

Vierte Zeile:

- Memory: Speicher mit Adresse aus ALUOut auslesen und in Memory Data Register schreiben.
- Sequencing: Gehen zum nächsten Mikrobefehl.

Fünfte Zeile:

- Register control: Inhalt des MDR in Registerwerk (Adresse aus rt) schreiben
- Sequencing: Gehen zum Mikrobefehl mit Markierung **`Fetch`**

Sechste Zeile:

- Memory: Speicherinhalt aus Register A in Speicher mit Adresse aus ALUOut schreiben
- Sequencing: Gehen zum Mikrobefehl mit Markierung **`Fetch`**

Mikroprogramm

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Siebte Zeile:

- ALU control, SRC1, SCR2: ALU operiert auf Inhalten der Register A und B, Funktionsfeld spezifiziert Operationstyp
- Sequencing: Gehen zum nächsten Mikrobefehl.

Achte Zeile:

- Register control: Inhalt von ALUOut wird ins Registerwerk geschrieben (Registernummer in rd)
- Sequencing: Gehen zum Mikrobefehl mit Markierung **`Fetch`**

Mikroprogramm

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

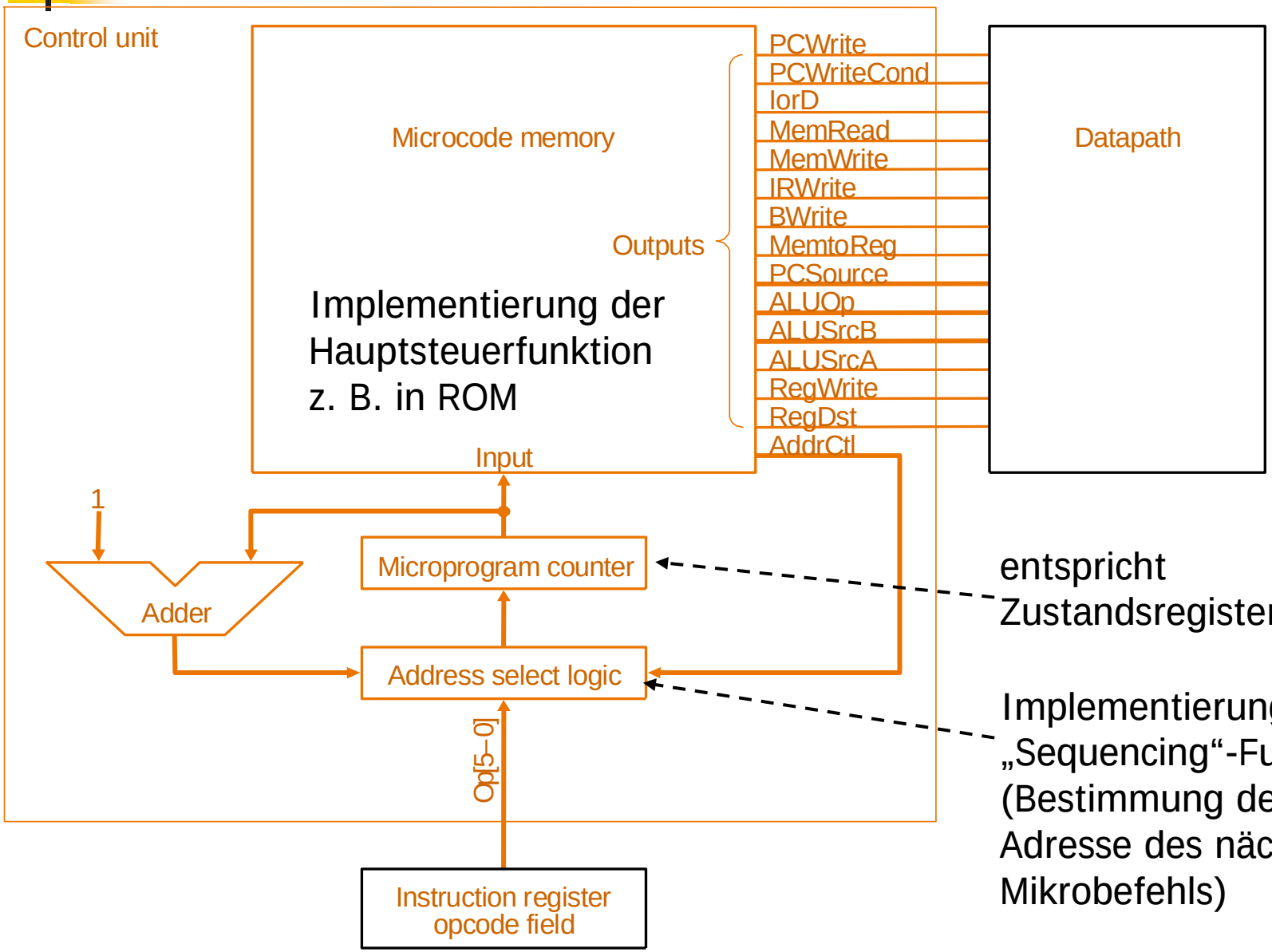
Neunte Zeile:

- ALU control, SRC1, SCR2: Subtraktion von A und B mit ALU, Ergebnis in Zero Output
- PCWrite control: PC wird mit Wert in ALUOut beschrieben, falls Zero Output von ALU „true“ ist.
- Sequencing: Gehen zum Mikrobefehl mit Markierung **`Fetch`**

Zehnte Zeile:

- PCWrite control: PC wird mit Wert der Sprungzieladresse beschrieben
- Sequencing: Gehe zum Mikrobefehl mit Markierung **`Fetch`**

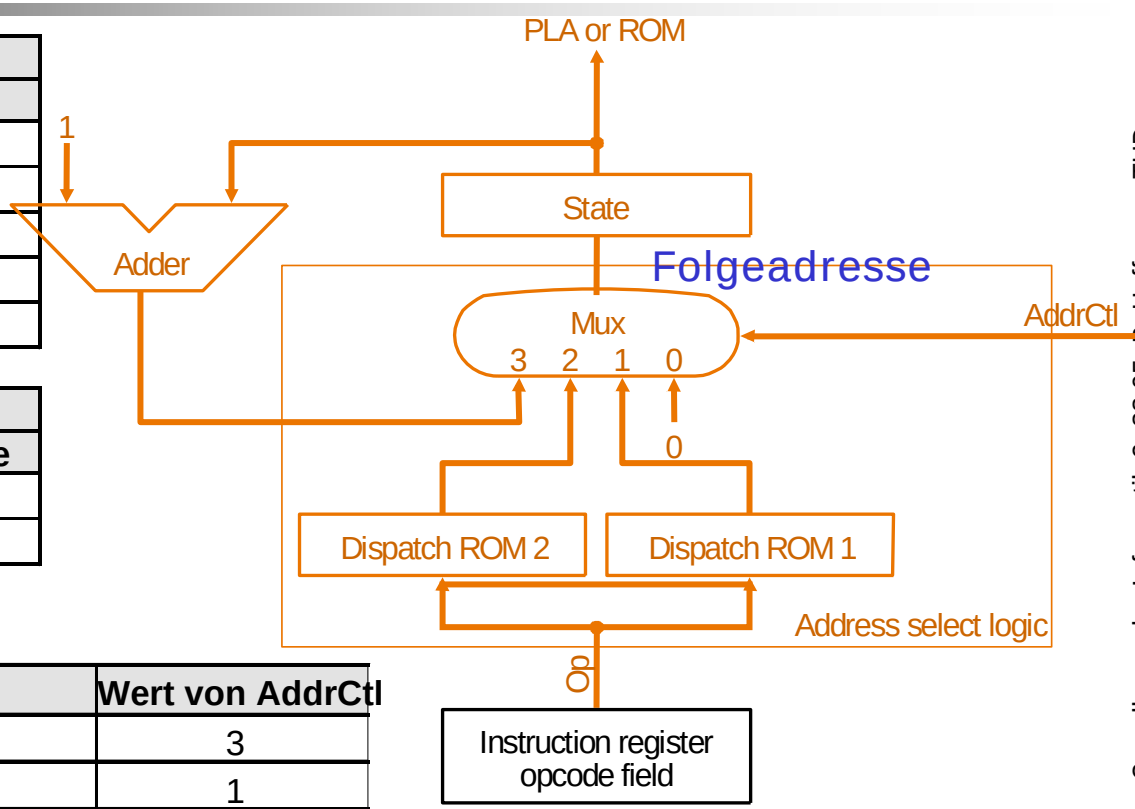
Mikroprogramm-Steuerwerk: Sequencing



Details: Auswahl der Folgeadresse

Dispatch ROM 1		
Op	**Opcode Name**	**Wert**
000000	R-format	0110
000010	`jmp`	1001
000100	`beq`	1000
100011	`lw`	0010
101011	`sw`	0010

Dispatch ROM 2		
Op	**Opcode Name**	**Value**
100011	`lw`	0011
101011	`sw`	0101



Zustandsnr	Adressierungsaktion	Wert von AddrCtl
0	Verwende erhöhten Zustand	3
1	Verwende Dispatch ROM 1	1
2	Verwende Dispatch ROM 2	2
3	Verwende erhöhten Zustand	3
4	Ersetze Zustandsnummer durch 0	0
5	Ersetze Zustandsnummer durch 0	0
6	Verwende erhöhten Zustand	3
7	Ersetze Zustandsnummer durch 0	0
8	Ersetze Zustandsnummer durch 0	0
9	Ersetze Zustandsnummer durch 0	0

Erkenntnisse

- Wie wir schon im Kapitel 4 gelernt haben, kann anstelle einer direkten Hardware-Steuerung (entsprechend dem Zustandsdiagramm) eine Mikroprogramm-Steuerung verwendet werden.
- Das soeben vorgestellte MIPS-Mikroprogramm besteht aus 10 Mikrobefehlen, die den 10 Zuständen des Zustandsdiagramms entsprechen.
- In der Praxis bestehen die Mikroprogramme meistens aus mehr Mikrobefehlen als die Anzahl der Zustände des Zustandsdiagramms.
- Das MP-STW ist für den MIPS optimiert.
- Das Mikroprogramm-STW hat den Vorteil, das es mit programmtechnischen Mittel leicht modifiziert oder erweitert werden kann. Somit lassen sich auch ältere Maschinenbefehlssätze emulieren oder andere implementieren.

Ausnahmebehandlung

- Ausnahmen sind unerwartete Ereignisse, die den normalen Kontrollfluß des Maschinenprogramms unmittelbar verändern.
- Man unterscheidet zwischen
  - „Exceptions“ (Arithmetik-Over/Underflows, ungültiger OPC, Zugriffsverletzung, SupervisorCall ...). Interne Ereignisse im Prozessor.
  - „Interrupts“: Programmunterbrechungen durch asynchrone externe Ereignisse (z. B. Ein-/Ausgabe-Anforderungen)

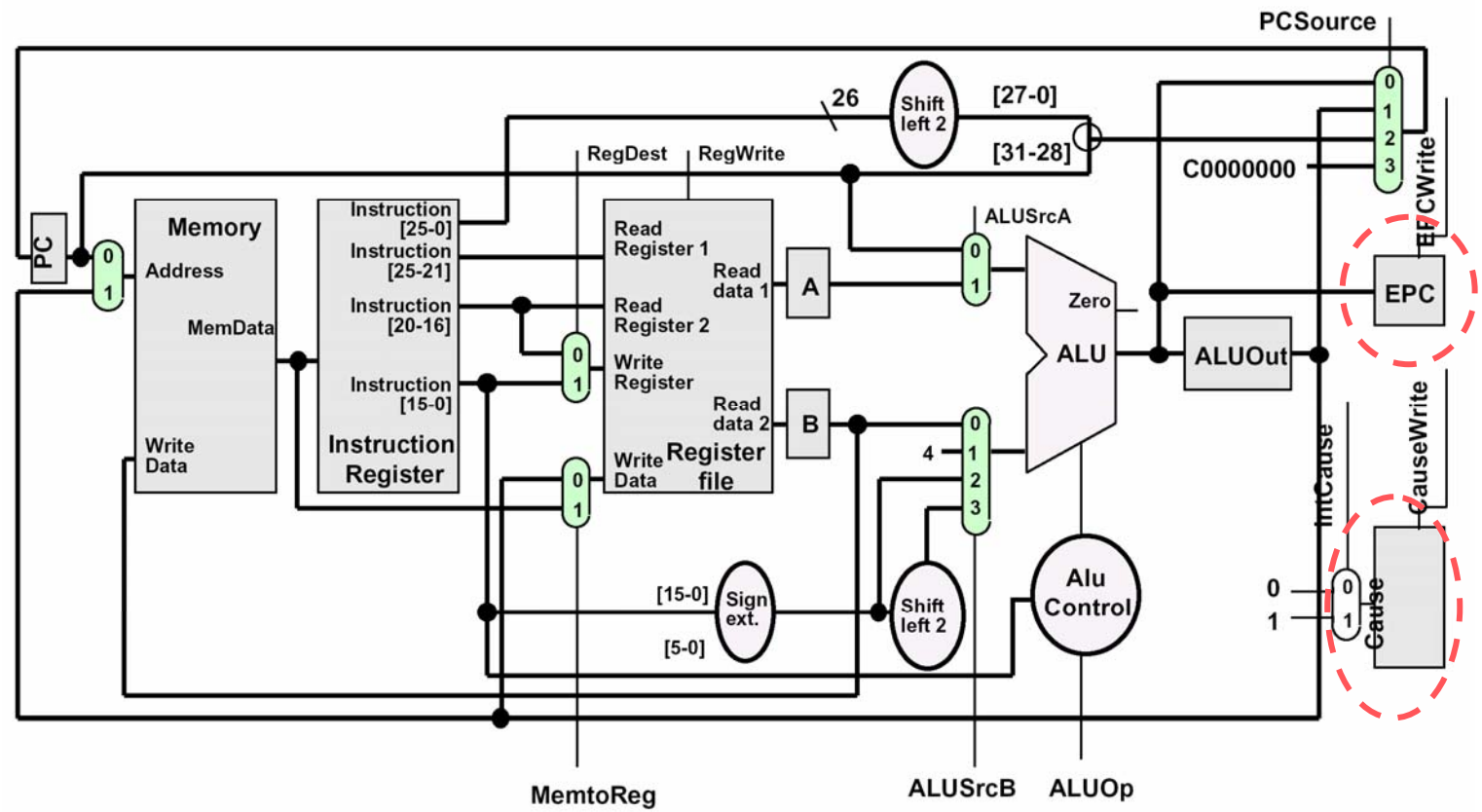
Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

Interrupt-Service-Routine

- Beim Erkennen einer Unterbrechung wird der Status der Maschine gerettet (hier zunächst nur PC) und es wird zu einer festgelegten Speicheradresse (Beginn der Interrupt-Service-Routine, Exception-Handler) verzweigt.
- hier wird die Adresse hex C0000000 benutzt.
- Dort steht eine Routine des Betriebssystems, die weitere Status-Sicherungsmaßnahmen ergreift, die Ursache der Unterbrechung feststellt und dann geeignet reagiert.
- Der Rücksprung aus der Routine erfolgt meist durch einen speziellen Befehl, der nicht nur den geretteten PC zurücklädt (wie beim Unterprogramm-Rücksprung) sondern zusätzlich den Status der Maschine vor der Unterbrechung wieder herstellt.

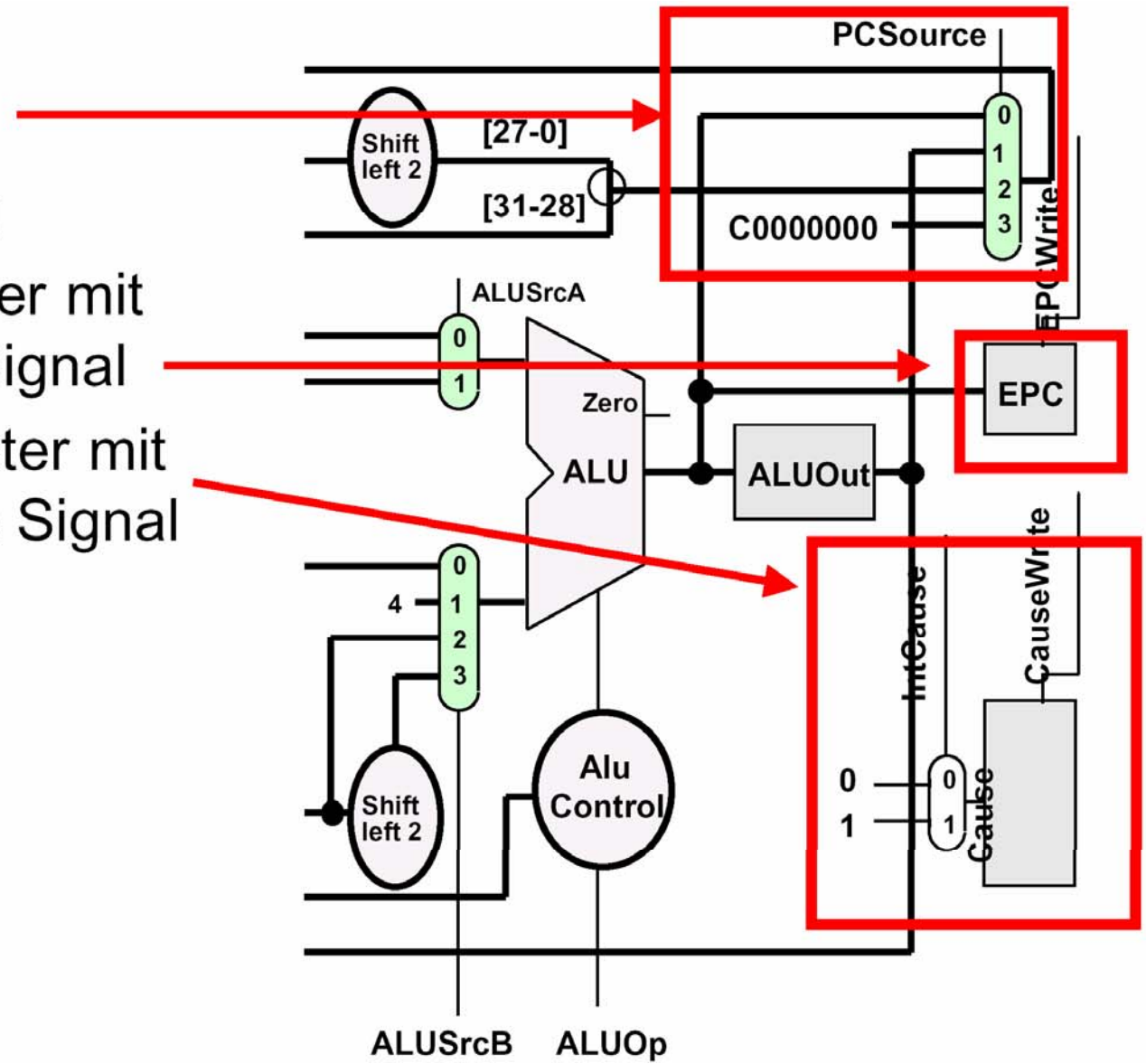
Hardware-Erweiterung

- „Exception Program Counter“ (EPC) zur Sicherung des PC-Wertes, bei dem die Ausnahme auftrat.
- „Cause Register“ zur Unterscheidung der beiden Fälle
  - undefinierter Befehl (0)
  - arithmetischer Überlauf (1)

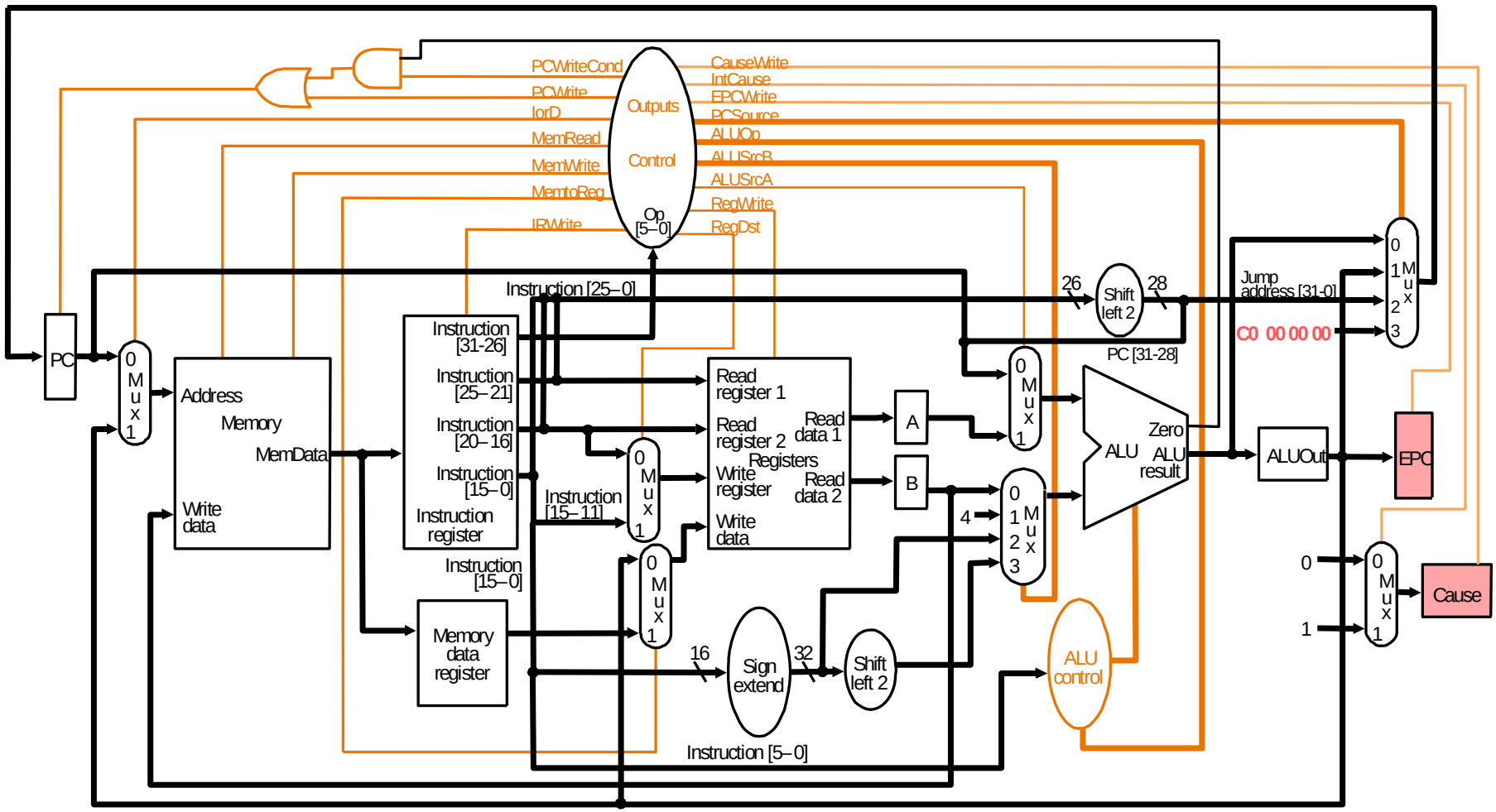


Ausnahmebehandlung im Detail

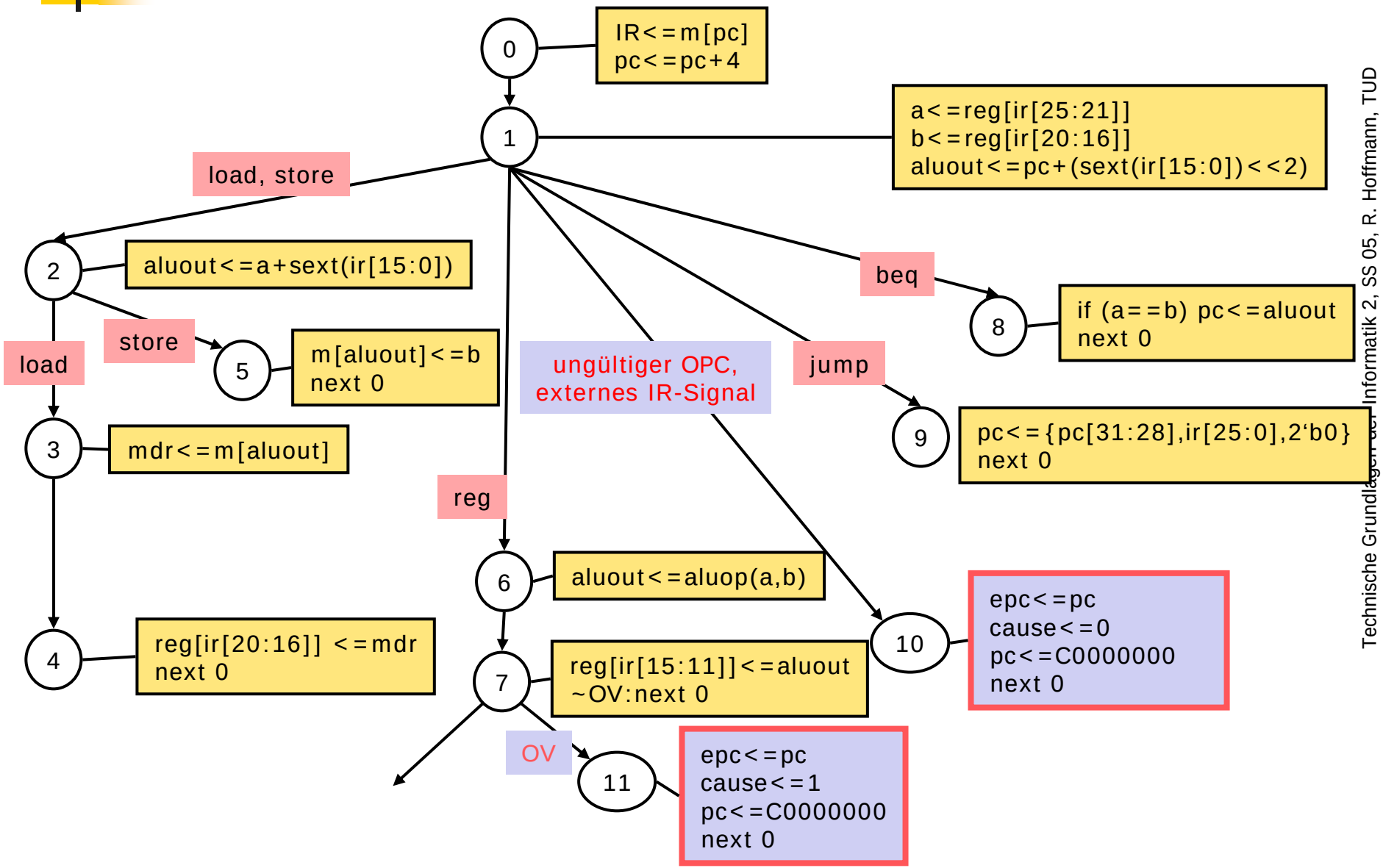
- PCSource Multiplexer Erweiterung
- EPC-Register mit EPCWrite Signal
- Cause register mit Cause write Signal



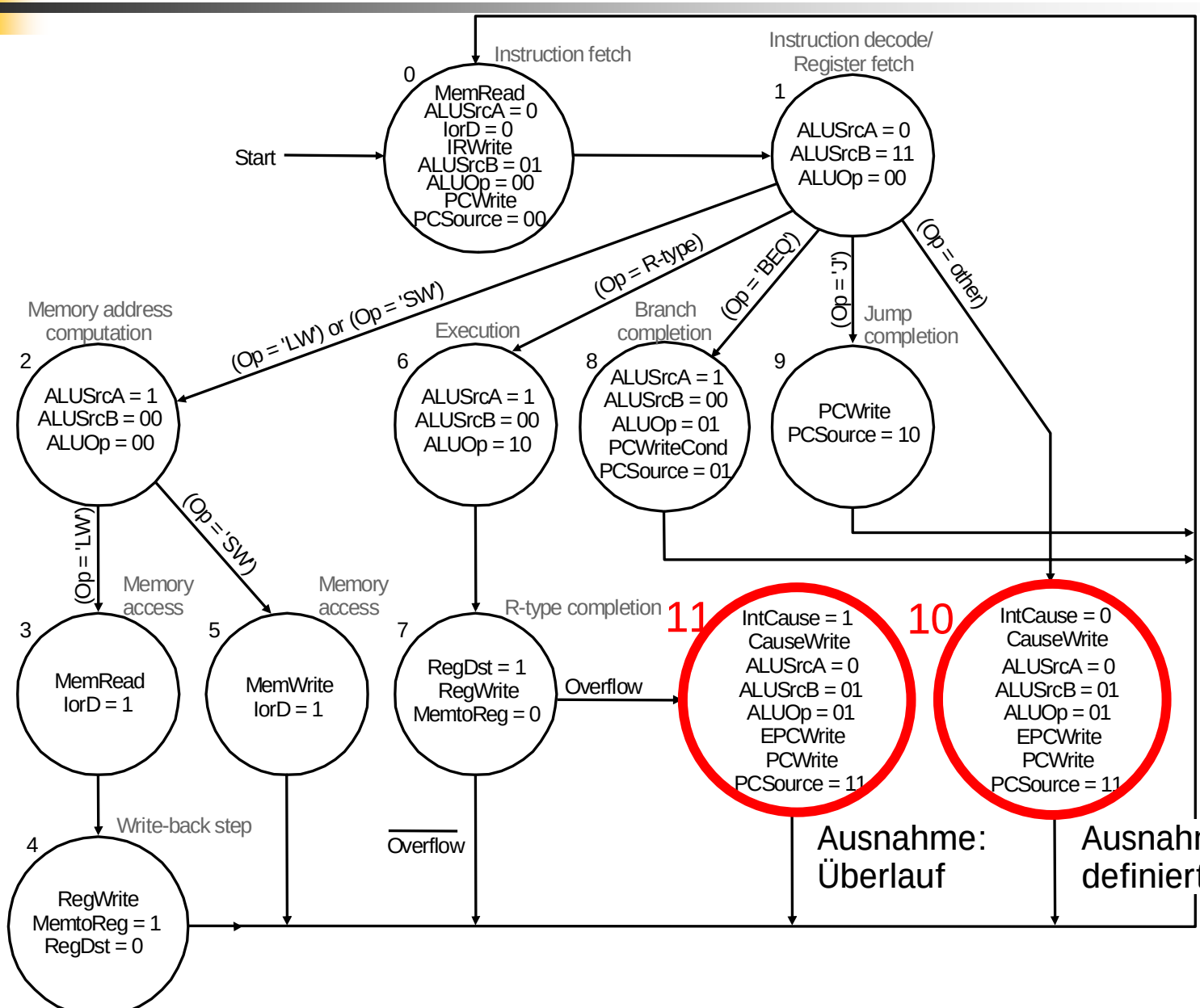
Mehrtakt-MIPS-Prozessor mit Ausnahmen



Zustandsdiagramm mit Ausnahmebehandlung



EA für Steuerung mit Ausnahmebehandlung



Zusammenfassung

- Eintakt-Implementierung MIPS war lehrreich
  - aber der kritische Pfad ist sehr lang
  - Programm- und Datenspeicher waren notwendig
- Mehrtakt-Implementierung ist uns vom DINATOS her bekannt.
  - Die Befehlsinterpretation wurde übersichtlich als (synchrones) Zustandsdiagramm mit Mikrooperationen dargestellt.
- Mehrtakt-Implementierung MIPS
  - Benötigt nur einen gemeinsamen Programm- und Datenspeicher
  - Zusätzliche Hilfsregister, Multiplexer waren erforderlich.
  - Je nach Befehlstyp werden 3-5 Takte benötigt.
  - Taktrate ist höher, weil der kritische Pfad kürzer ist.
  - Steuerwerk sendet die Steuersignale zum Operationswerk entsprechend der Reihenfolge, die durch das Zustandsdiagramm vorgegeben ist.
  - Steuerwerk kann als Hardware-Steuerwerk oder als Mikroprogramm-Steuerwerk implementiert werden.
  - Mikroprogramm-STW kann leichter umprogrammiert werden, um alte oder erweiterte Befehlssätze zu interpretieren.