Kapitel 7 (2. Teil MIPS): Implementierung des Operationswerks

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 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.

Wiederholung: Erste MIPS-Befehle

(1) Arithmetisch-logische Operationen

add $s1,$s2, $s3	$s1 = $s2 + $s3
sub $s1,$s2, $s3	$s1 = $s2 – $s3
slt $t0, $s1, $s2	wenn $s1 < $s2, dann $t0=1, sonst $t0=0

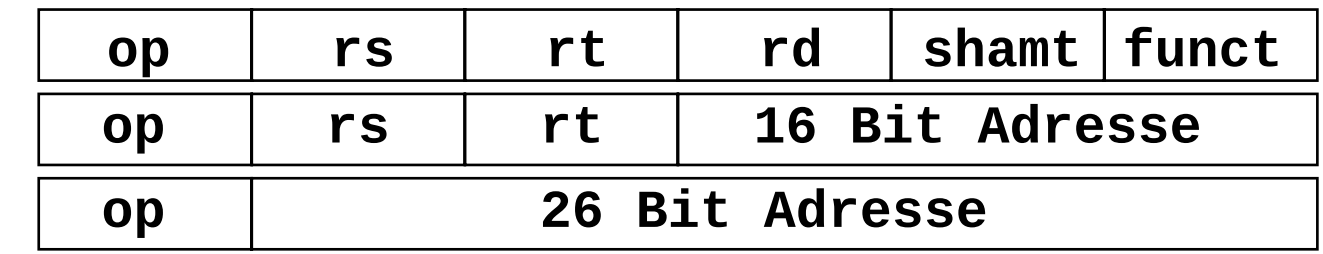
(2) Speicherzugriffsbefehle

lw $s1, 100($s2)	$s1 = Memory[$s2+100]
sw $s1, 100($s2)	Memory[$s2+100] = $s1

(3) Verzweigungsbefehle

bne $s4, $s5, L	nächste Instr. ist bei L, wenn $s4 != $s5
beq $s4, $s5, L	nächste Instr. ist bei L, wenn $s4 = $s5
j Label	nächste Instr. ist bei Label

Formate:



Befehlsgruppen

- 3 Gruppen von Befehlen:
  a) Arithmetisch-logische Operationen vom R-Typ (arithmetic-logical instructions), z. B. **`add, sub, and, or, slt`**
  b) Speicherzugriffsbefehle (memory-reference instructions), z. B. **`lw, sw`**
  c) Verzweigungsbefehle (branch instructions), z. B. **`beq, j`**
- Durch Einfachheit und Regularität des Befehlssatzes ergeben sich für alle drei Gruppen ähnliche Ausführungsschritte → (s. nächste Folie)

Ausführungsschritte

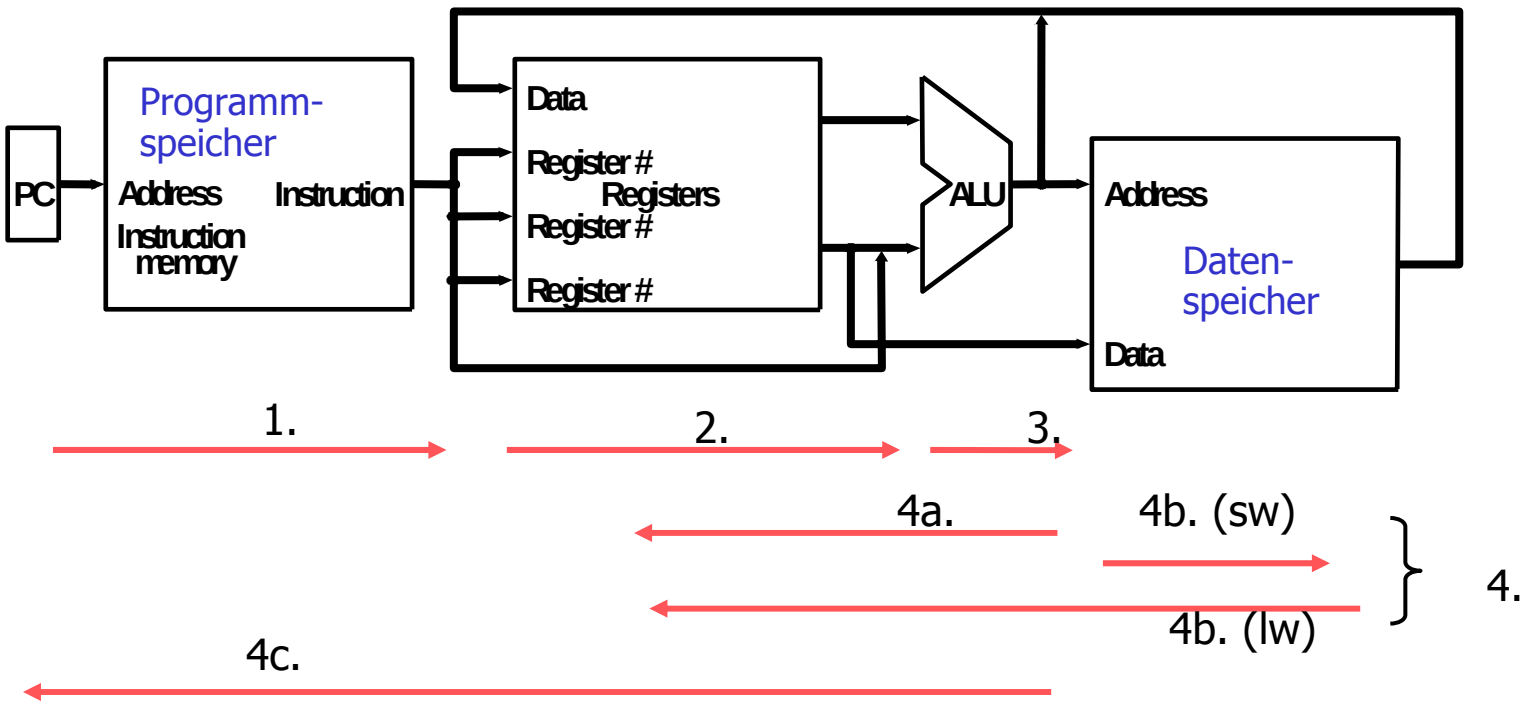
1. **(Befehl holen)** Befehl an der durch PC (program counter) markierten Stelle aus dem Programmspeicher holen
2. **(Register-Operanden holen)** Lesen der adressierten Registerinhalte
3. **(Operation)** Operationsausführung oder Adressberechnung mit der ALU (bei lw, sw)
4. **(Befehl beenden)**
   - siehe nächste Folie

Befehl beenden (4. Schritt)

a) **(add, sub, ...)** Rechenergebnis von der ALU in das Zielregister schreiben

b) **(lw, sw)** Operanden in den Datenspeicher schreiben (sw) oder aus dem Datenspeicher lesen und in das Zielregister schreiben (lw)

c) **(beq, j, ...)** Befehlszähler PC auf die Sprungadresse (Target-Adresse) setzen

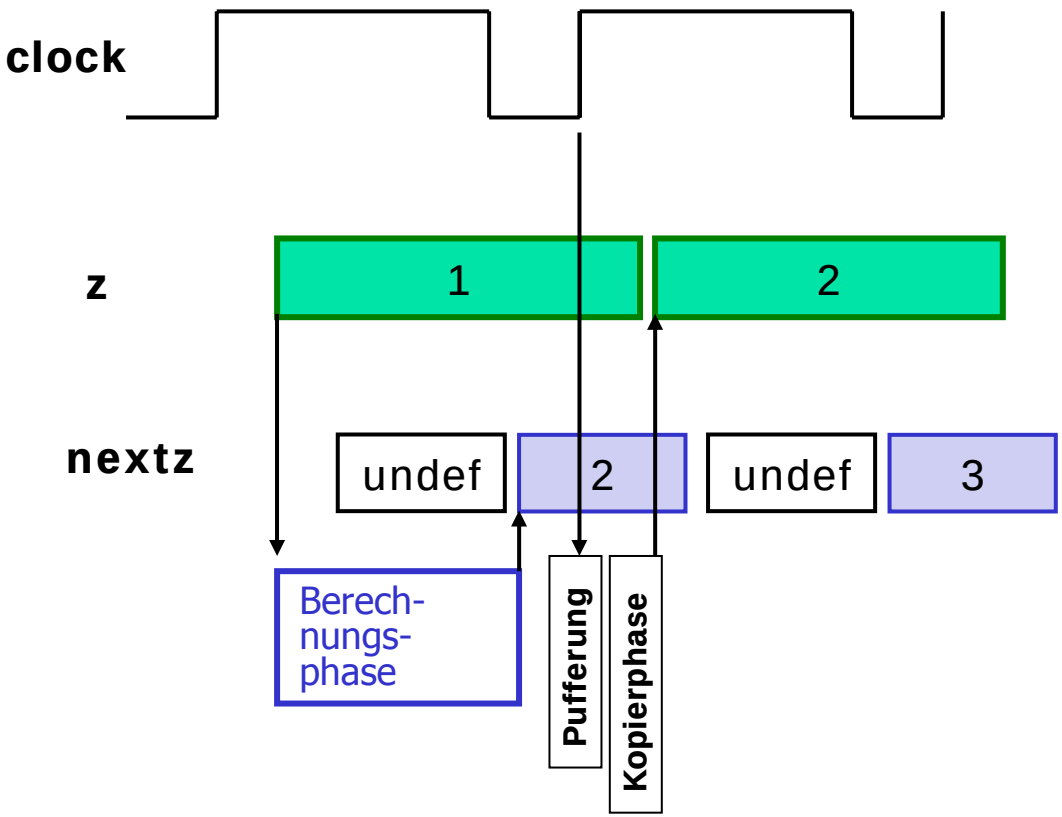


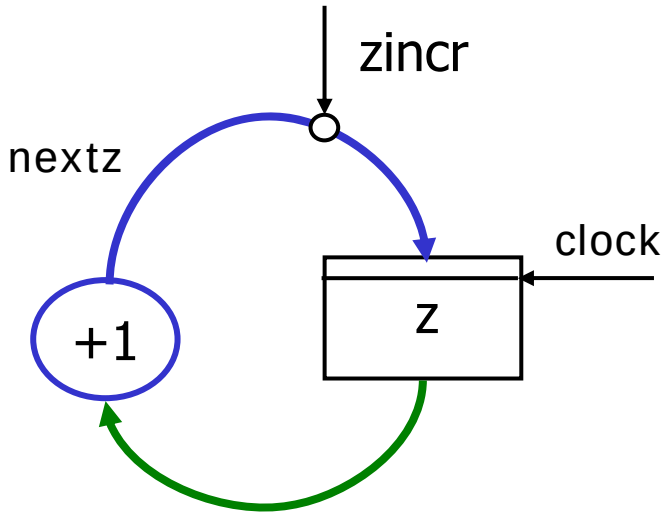
Logik-Komponenten

- MIPS-Implementierung beruht auf zwei unterschiedlichen Logik-Komponenten:
  - (Schaltnetze, combinational circuits). Die Ergebnisse hängen nur von den aktuellen Eingängen ab (z. B. ALU)
  - (Speicherelemente, state elements) Komponenten, die Zustände speichern können.
    - Speicherungsart
      - **flüchtig**: Daten gehen bei Stromausfall verloren
      - **nicht flüchtig**: Daten bleiben bei Stromausfall erhalten

Zeitlicher Ablauf einer Mikrooperation

- Beispiel z ← z+1



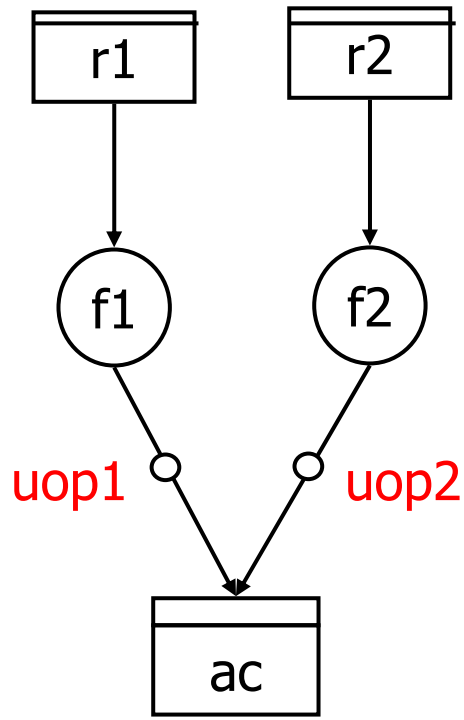


zincr: (z<=z+1)

Durch das Steuersignal **zincr** wird die synchrone Mikrooperation z<=z+1 aktiviert.

Aktivierung und Dauer einer Mikrooperation

- Aktivierung durch ein Steuersignal (z. B. `uop1` = CounterInkrement)
- Wenn das Operationswerk das Steuersignal erkennt:
  - Lies die (alten, momentanen) Werte aus den Registern.
  - Berechne die neuen Werte durch ein Schaltnetz.
  - Speichere die neuen Werte mit der positiven Taktflanke.
- Meist wird versucht, die Mikrooperation innerhalb eines Taktzyklus auszuführen. Falls die Berechnung länger dauert
  - Taktzeit verlängern oder n Takte warten und erst danach das Ergebnis verwenden.



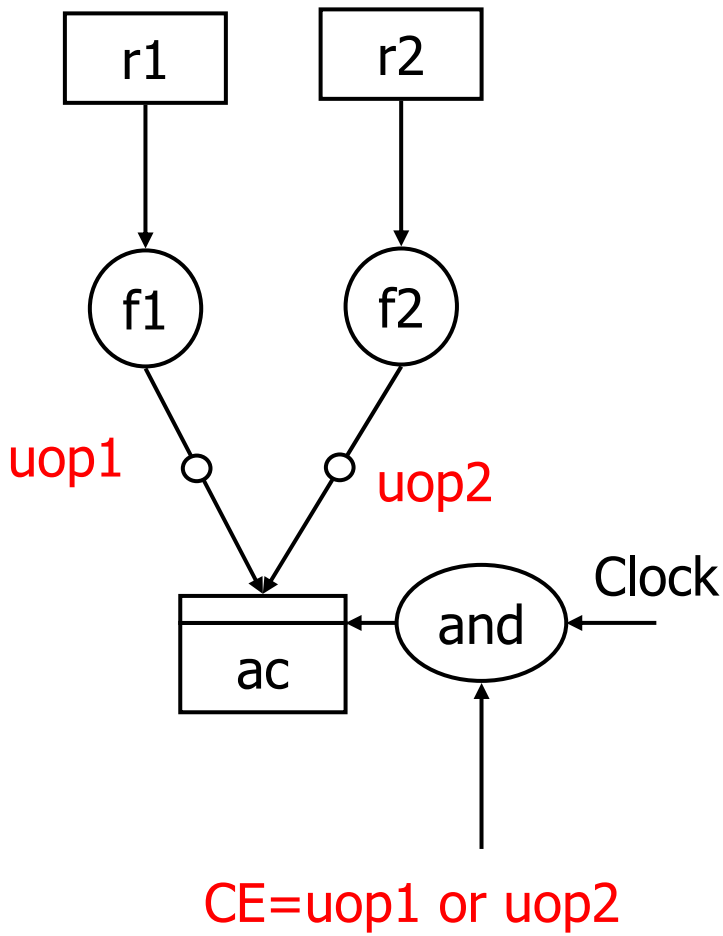
uop1: (ac<=f1(r1))

uop2: (ac<=f2(r2))

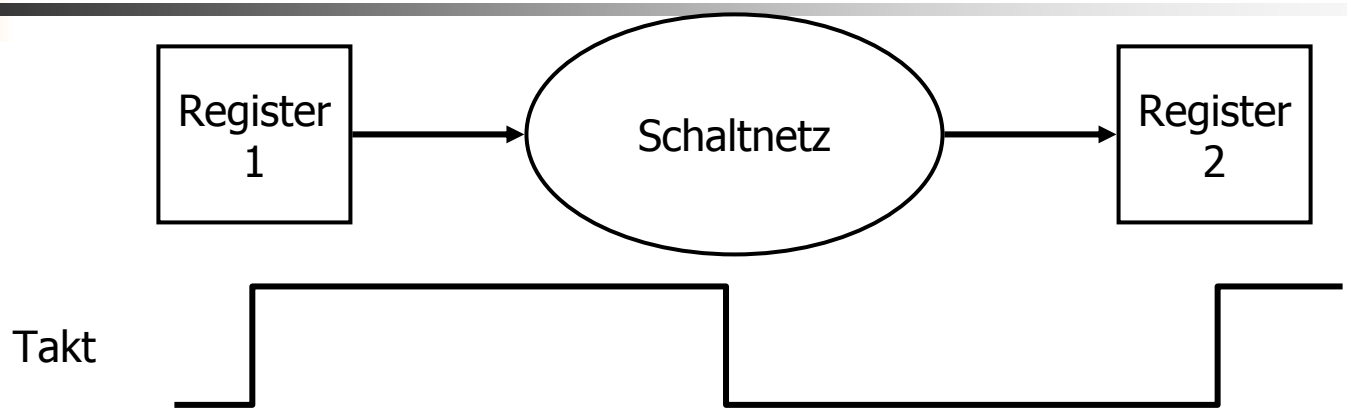
Write-Enable / Clock-Enable

- Falls ein Register nicht während jedes aktiven Taktimpulses geändert werden soll, dann wird ein explizites Schreibsignal (Write-Enable, Clock-Enable) zum Schreiben benötigt. D. h. Schreiben erfolgt dann nur bei aktivem Clock-Enable (CE) und Auftreten der Taktflanke.
- Wenn das Register sich mit jedem Takt verändern soll, dann ist ein Clock-Enable nicht erforderlich, weil es immer 1 ist.

Implementierung der Aktivierung durch ClockEnable



Synchrone Schaltung



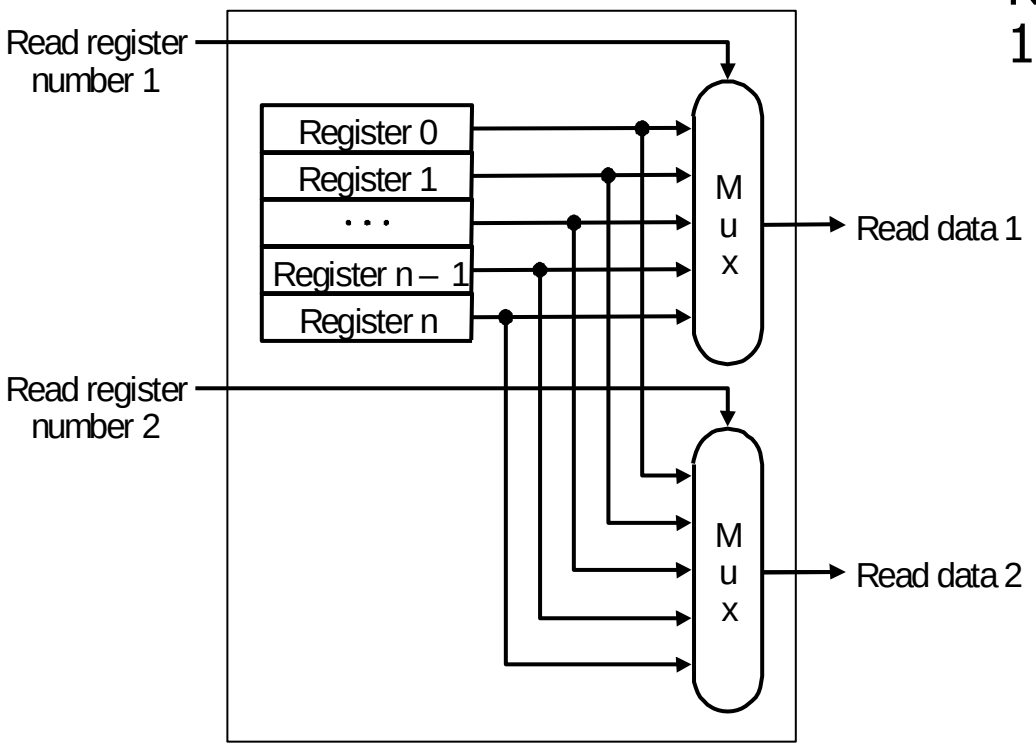
- Eine **Synchrone Schaltung** (die bei der Implementierung von MIPS verwendet wird) besteht aus synchronen **Registern** und **Schaltnetzen**.
- **Register** übernehmen die Signale an ihren Eingängen zu festen Zeitpunkten, die durch die Flanken eines Taktsignals definiert sind.
- Ein **Schaltnetze** ist kontinuierlich aktiv. Stabile Ausgangswerte stellen sich erst nach einer bestimmten Verzögerungszeit (Berechnungszeit) nach dem Anlegen stabiler Eingangswerte ein.
- Eine korrekte Implementierung einer Schaltung stellt sicher, daß sich die Schaltnetze zwischen zwei signifikanten Taktflanken stabilisieren können.

Registerspeicher

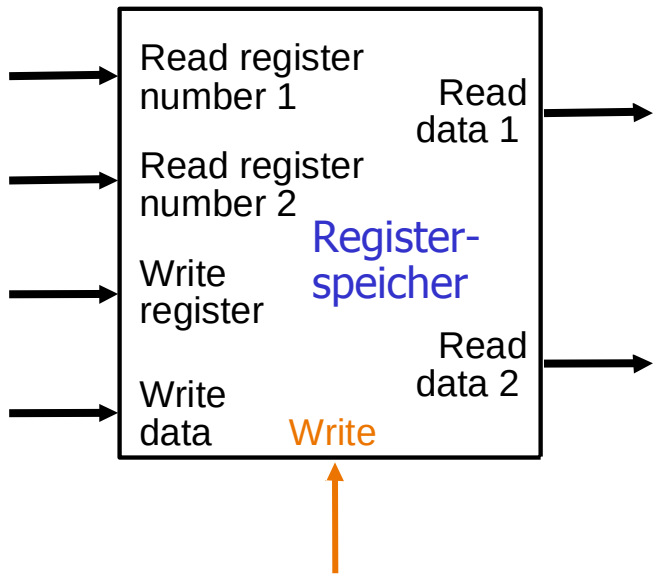
- Registerspeicher ist von zentraler Bedeutung für das Rechenwerk.
- Registerspeicher besteht aus
  - einer Anzahl Register,
  - die durch Angabe einer Registernummer (auch Registeradresse) ausgelesen oder geschrieben werden können.
- Auslesen eines Registers ändert keinen internen Zustand
  - 1. Eingabe: Registernummer
  - 2. Ausgabe: Datenwert des Registers
- Schreiben eines Registers benötigt (mind.) 3 Eingaben:
  - Registernummer
  - zu schreibender Datenwert
  - Steuersignal (Schreibsignal, Schreibimpuls, write), das das Schreiben des Registers steuert

Registerspeicher

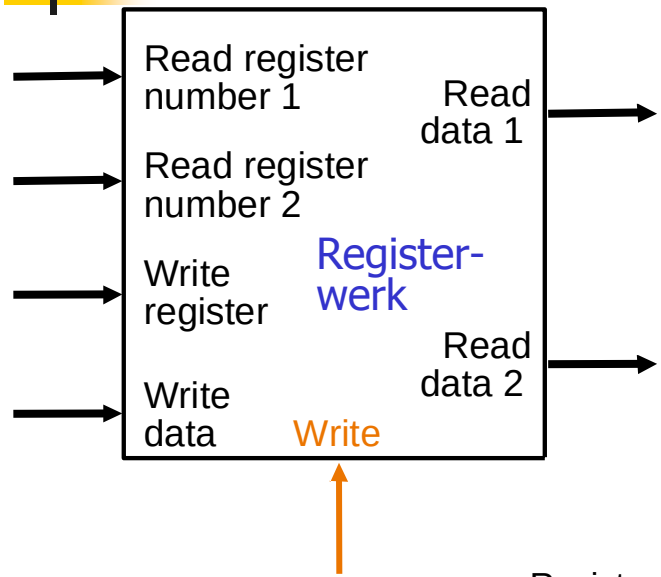
- Realisierung eines Registers durch Feld von D-Flip-Flops
- Realisierung der zwei Read-Ports durch zwei Multiplexer (n: Anzahl der Register, hier n=32) :



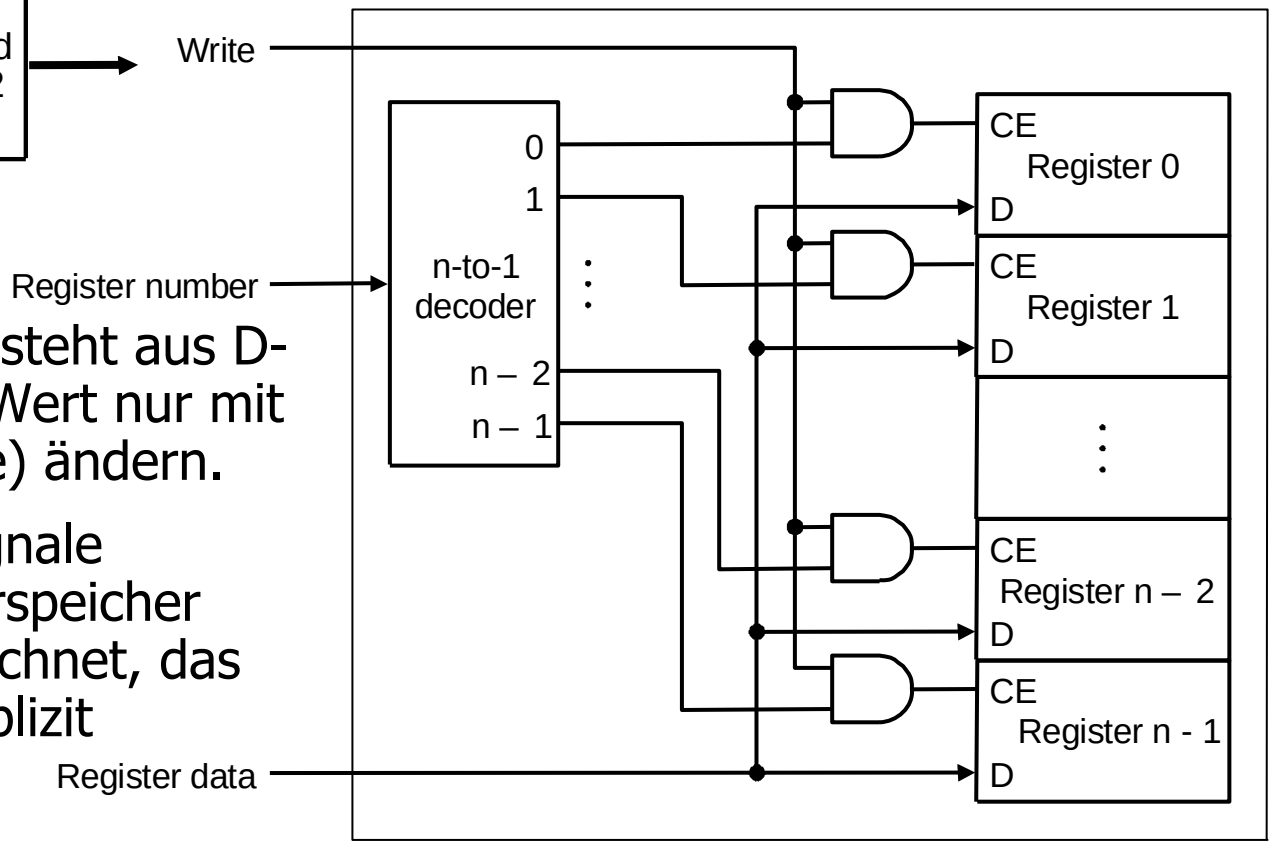
- Für Befehle im R-Format wird ein Registerspeicher mit 2 Read- und 1 Write-Port benötigt:



Registerspeicher



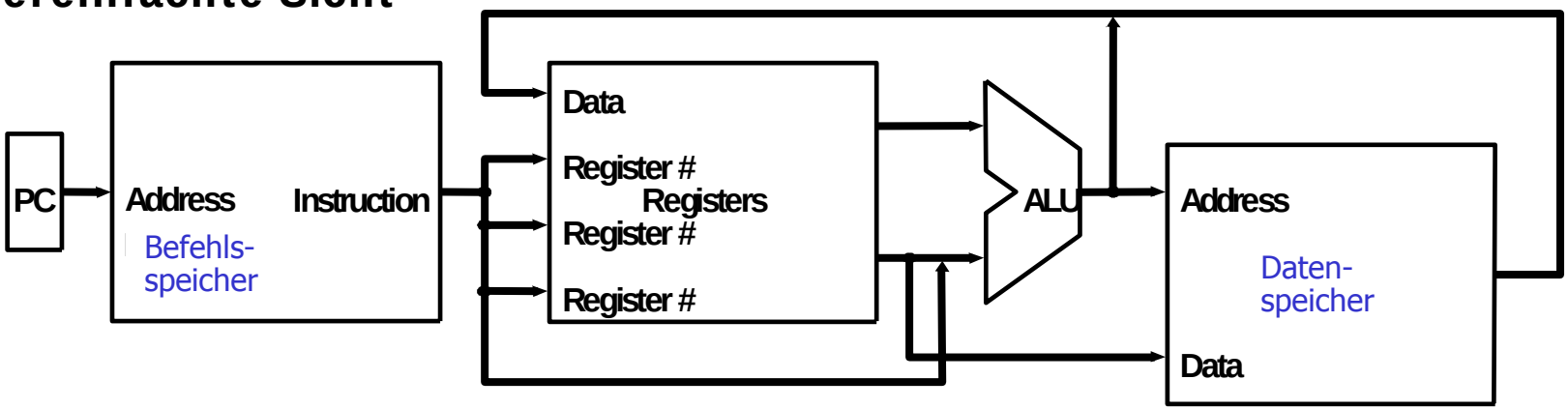
- Write wird auf das zu schreibende Register geleitet durch einen Decoder sowie UND-Gattern (ergibt einen Demultiplexer für Write)



- Achtung: Register besteht aus D-Flip-Flops, die ihren Wert nur mit der Taktflanke (Write) ändern.
- Notation: Schreib-Signale („Write“) für Registerspeicher werden explizit gezeichnet, das Taktsignal C wird implizit angenommen.

Entwurf des MIPS-Prozessors

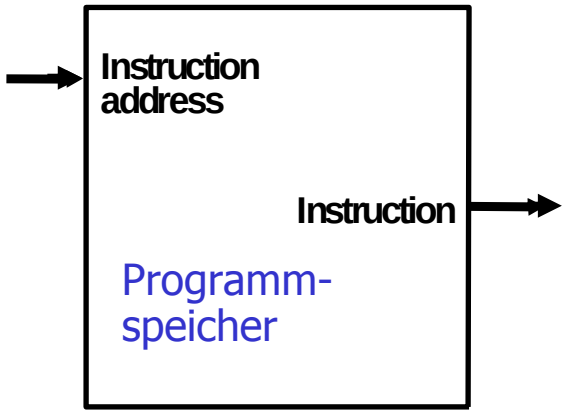
Vereinfachte Sicht



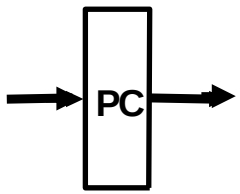
- Der Entwurf des MIPS-Prozessors wird im folgenden beschrieben, wobei ein konstruktives bottom-up Verfahren verwendet wird. Dabei wird zuerst das Operationswerk schrittweise mächtiger gemacht, bis es alle Operationen durchführen kann.
  - Schritt 1: Konstruktion von Teilen des Operationswerks. Ermittlung der Datenwege, die zur Implementierung der Mikrooperationen für die einzelnen Befehlsklassen erforderlich sind.
  - Schritt 2: Zusammensetzen der Datenwege zum Operationswerk

1. Datenweg für das Holen eines Befehls

- Grundkomponenten für den ersten Schritt:

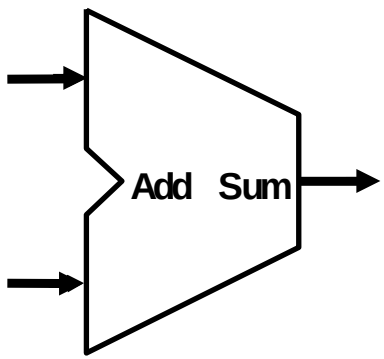


Ein Programmspeicher für die auszuführenden Befehle



Ein Register zur Speicherung der aktuellen Befehlsadresse

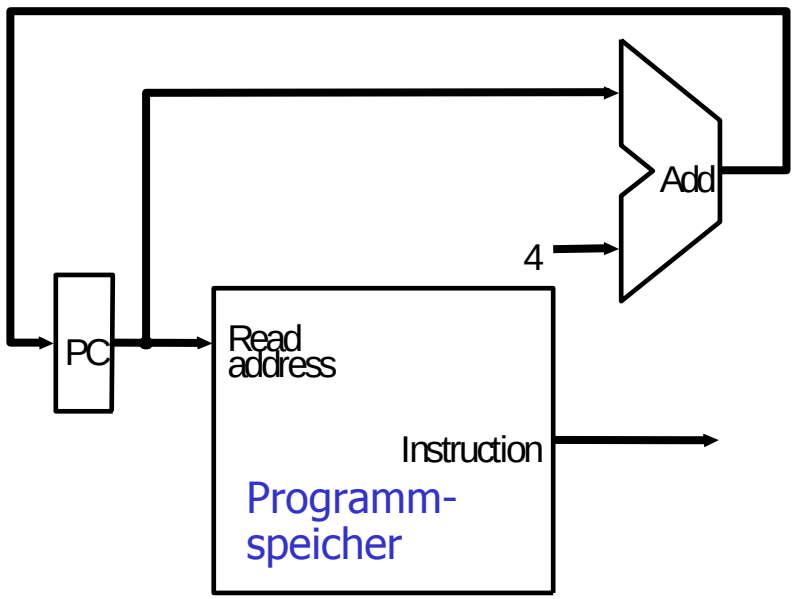
(Program Counter)



Ein Addierer zur Berechnung der nächsten Befehlsadresse

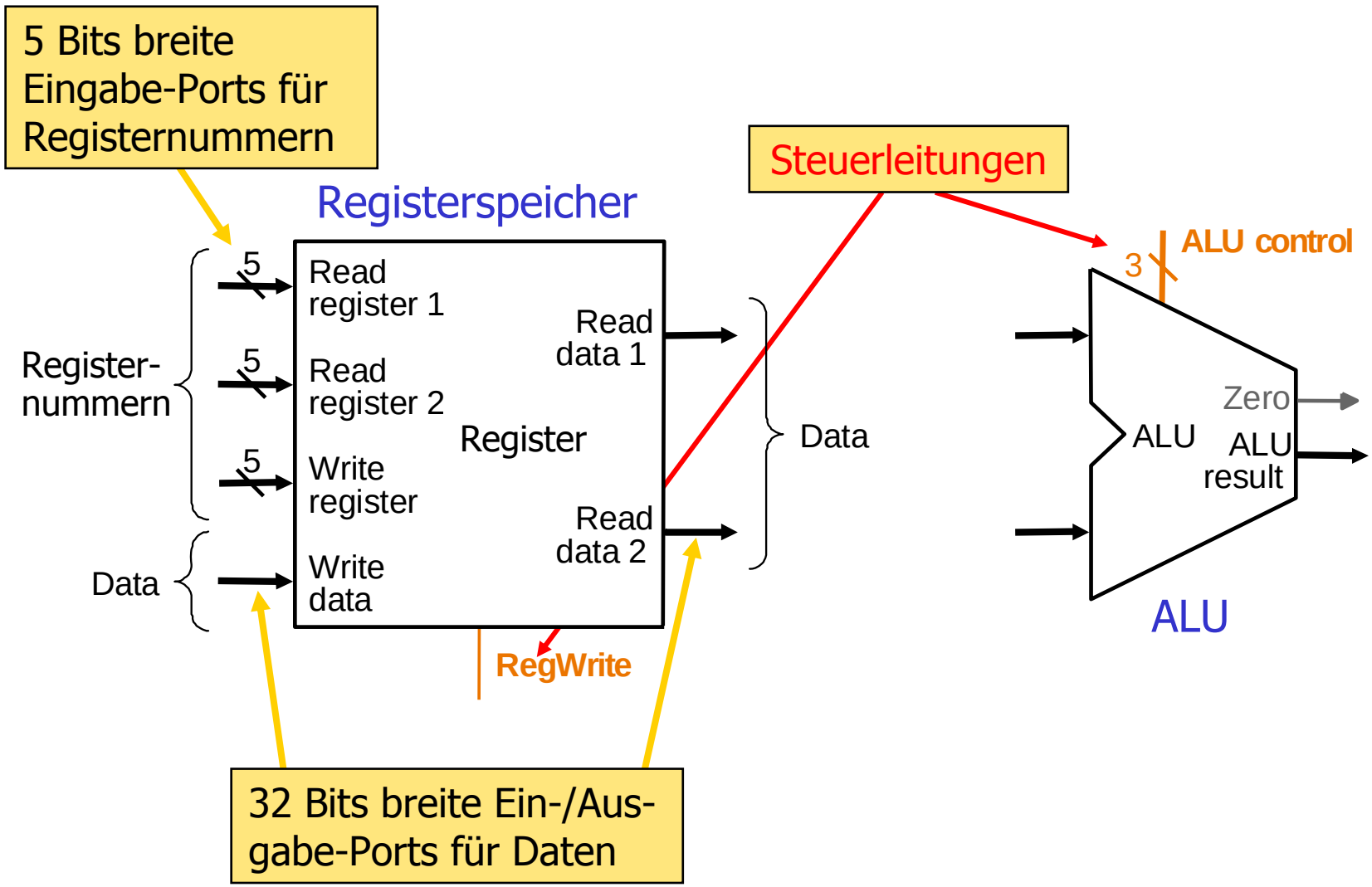
1. Datenweg für das Holen eines Befehls

- Programmspeicher, auf viele Arten realisierbar
  - SRAM, DRAM, Flash, MRAM, ...
- Speicherung der Adresse des Befehls im Befehlszähler (PC)
- Aufeinanderfolgende Befehle werden durch sukzessives Addieren von 4 (Bytes) auf den Befehlszähler adressiert.

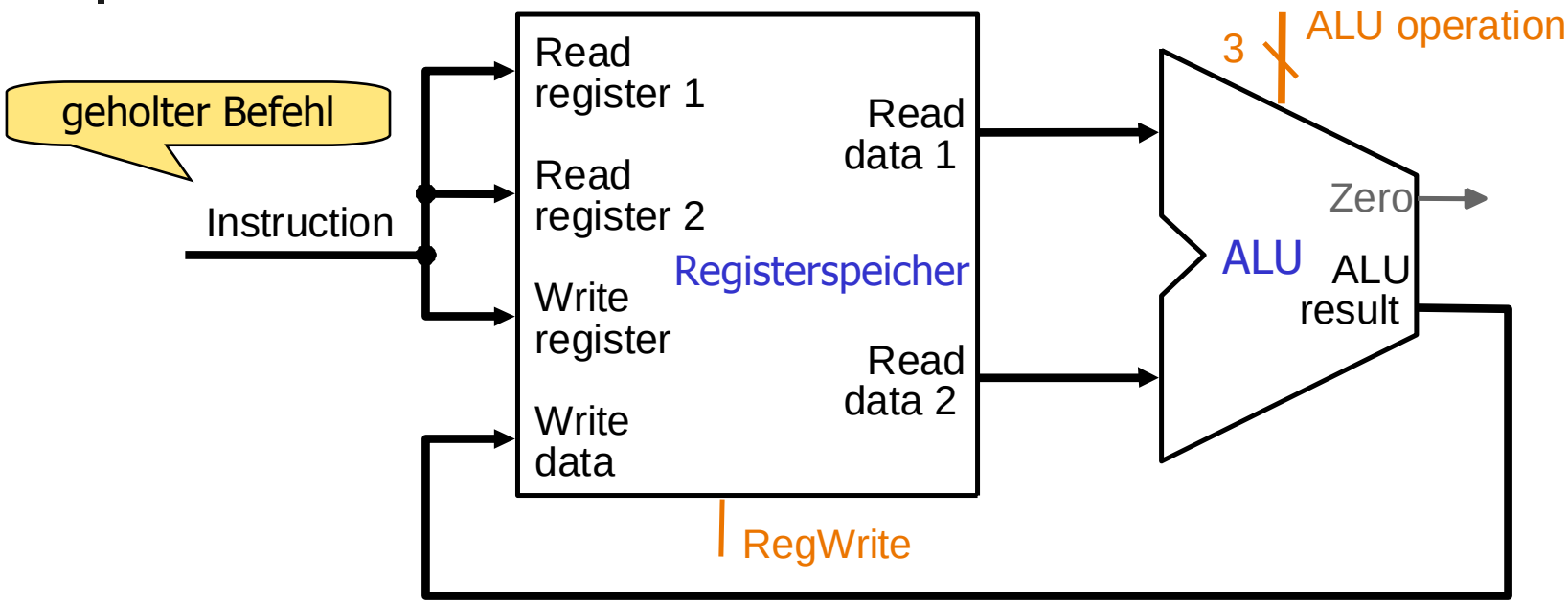


2. Datenwege für Befehle im R-Format

2 Grundelemente werden benötigt:



2. Datenwege für Befehle im R-Format



- Die im R-Format angegebenen Registeradressen werden an die Adresseingänge des Registerspeichers gelegt.
- Die Ausgänge des ausgewählten Register (Quellregister) sind Eingänge der ALU.
- Der ALU Ausgang wird ins ausgewählte Zielregister geschrieben.

3. Datenwege für Datenspeicherzugriffe

lw $s1, 100($s2)	$s1 = Memory[$s2+100]
sw $s1, 100($s2)	Memory[$s2+100] = $s1

I-Format

- Zuerst ist eine Adressberechnung durchzuführen. Der Inhalt des Registers **$s2** und die Konstante **Offset** sind zu addieren. Dazu wird die ALU benutzt.

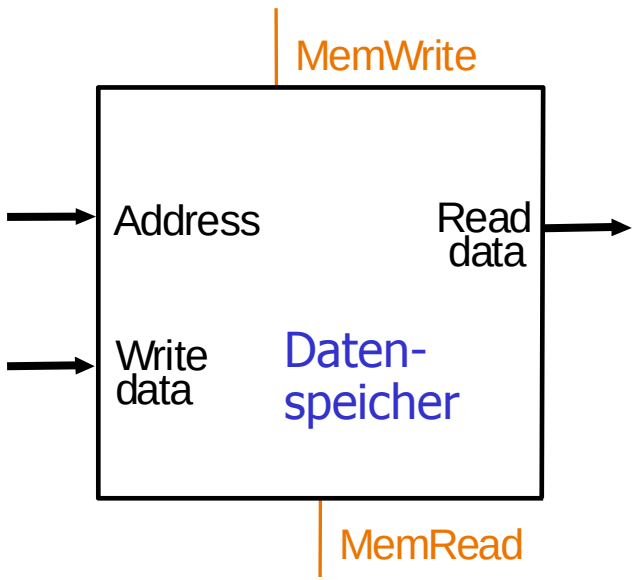
3. Datenwege für Datenspeicherzugriffe

op	rs	rt	16 Bit Adresse

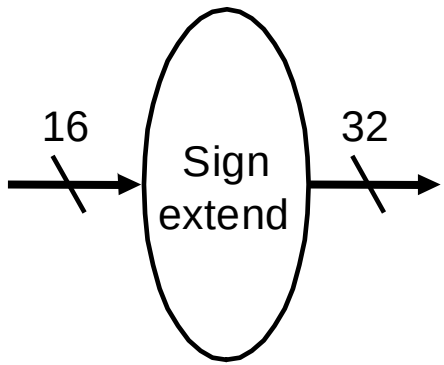
rt ← DataMemory[rs+sext(Offset)]
rt → DataMemory[rs+sext(Offset)]

- Zusätzlich benötigt man:

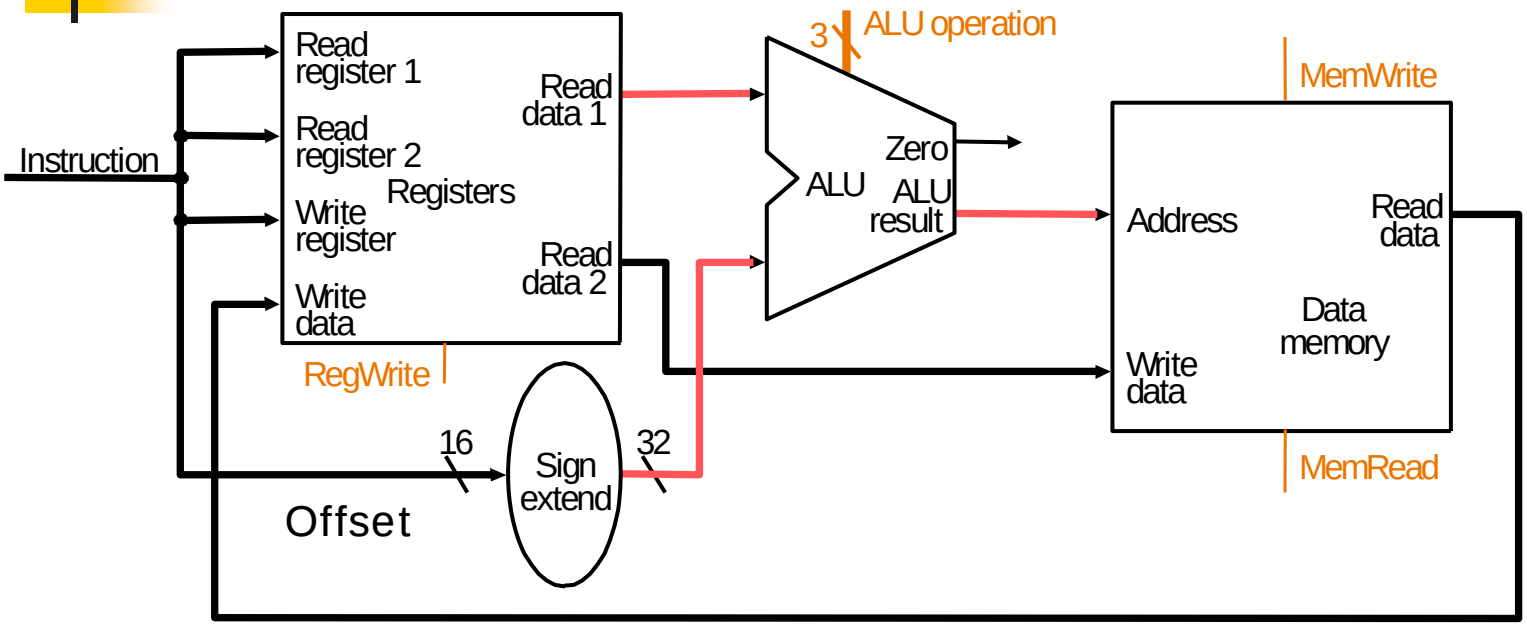
Datenspeicher
mit Adress- und Daten-Ein/Ausgängen



Sign-Extend-Einheit
für die Offset-Addition



3. Datenweg für Datenspeicherzugriffe



- Datenweg für **lw** oder **sw** benötigt:
  - Register lesen (ReadData1=Basisadresse),
  - Datenadressberechnung: Addition des Offsets aus dem Befehl liefert Address
    - (bei **lw**) anschließendes Lesen vom Datenspeicher (ReadData) und Schreiben in den Registerspeicher (Write Register, Write Data)
    - (bei **sw**) ReadData 2 in den Datenspeicher schreiben.

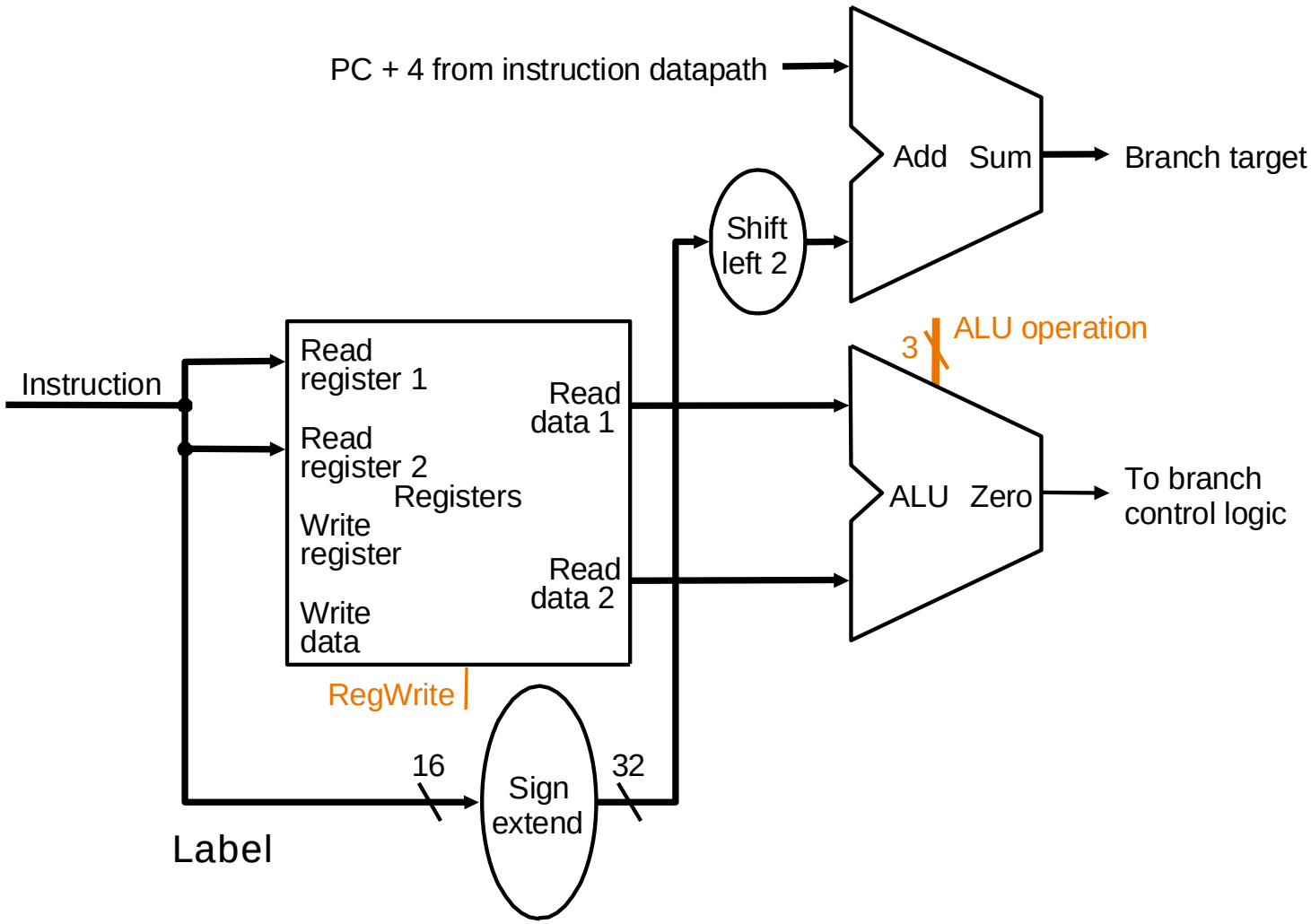
4. Datenweg für bedingte Verzweigung

- Befehl: **`beq $t1, $t2, Offset`**
- Besonderheiten:
  - Es sind zwei Registerinhalte zu vergleichen. Dies geschieht durch Subtraktion und Nutzung des Zero-Ausgangs der ALU.
  - Abhängig vom Ergebnis des Vergleichs ist möglicherweise ein neuer Wert in den Befehlszähler zu laden. Die Entscheidung wird durch Aktionen der Steuerung realisiert (siehe später).

- Die Berechnung des neuen Wertes für den Befehlszähler geht von einer Basis **PC+4** aus. (Beim Holen eines Befehls wird der Befehlszähler immer schon vorweg um 4 erhöht)
- Der im Befehl angegebene Offset wird als Wortindex (Anzahl Speicherworte) interpretiert. Der PC adressiert aber Bytes.
  Daher ist die Konstante aus dem Befehl um 2 Bits nach links zu verschieben (Multiplikation mit 4 durch Schift).
- Für die Berechnung des Sprungziels (branch target), der in den PC geladen werden muß, ist eine Addition erforderlich (genauer: sinnvoll). Dafür wird ein separater Addierer benötigt, da die ALU bereits mit dem Vergleich (siehe oben) beschäftigt ist.

4. Datenweg für bedingte Verzweigung

- Datenweg für **`beq $t1, $t2, Offset`**

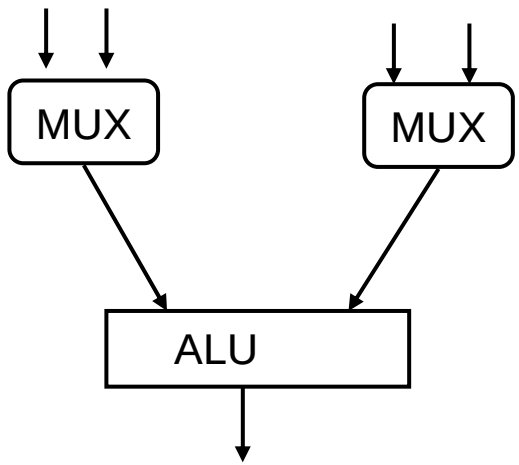


5. Vollständiges Operationswerk

- Bisher wurden die Datenwege einzeln für die Befehlsklassen konstruiert.
- Aus diesen entsteht ein vollständiges Operationswerk durch Zusammenfügen der einzelnen Datenwege.
- Ein Steuerwerk muß noch hinzugefügt werden, das die Datenwege in Abhängigkeit von dem Befehl in der richtigen Auswahl und Reihenfolge durchschaltet.
- Im folgenden wird zunächst eine einfache Realisierung des OPW vorgestellt, die davon ausgeht, daß eine vollständige Befehlsausführung in einem einzigen Taktzyklus stattfinden kann: „Single-Cycle-Datapath“.

Zusammenfügen der Datenwege

- Ähnliche Datenwege können durch dieselbe Hardware-Komponente realisiert werden, wenn sie nicht gleichzeitig aktiv sein müssen!
- Unterschiedliche Datenquellen für dieselbe Hardware-Einheit können durch Multiplexer implementiert werden.



Operationswerk-Implementierung: Schritt (i)

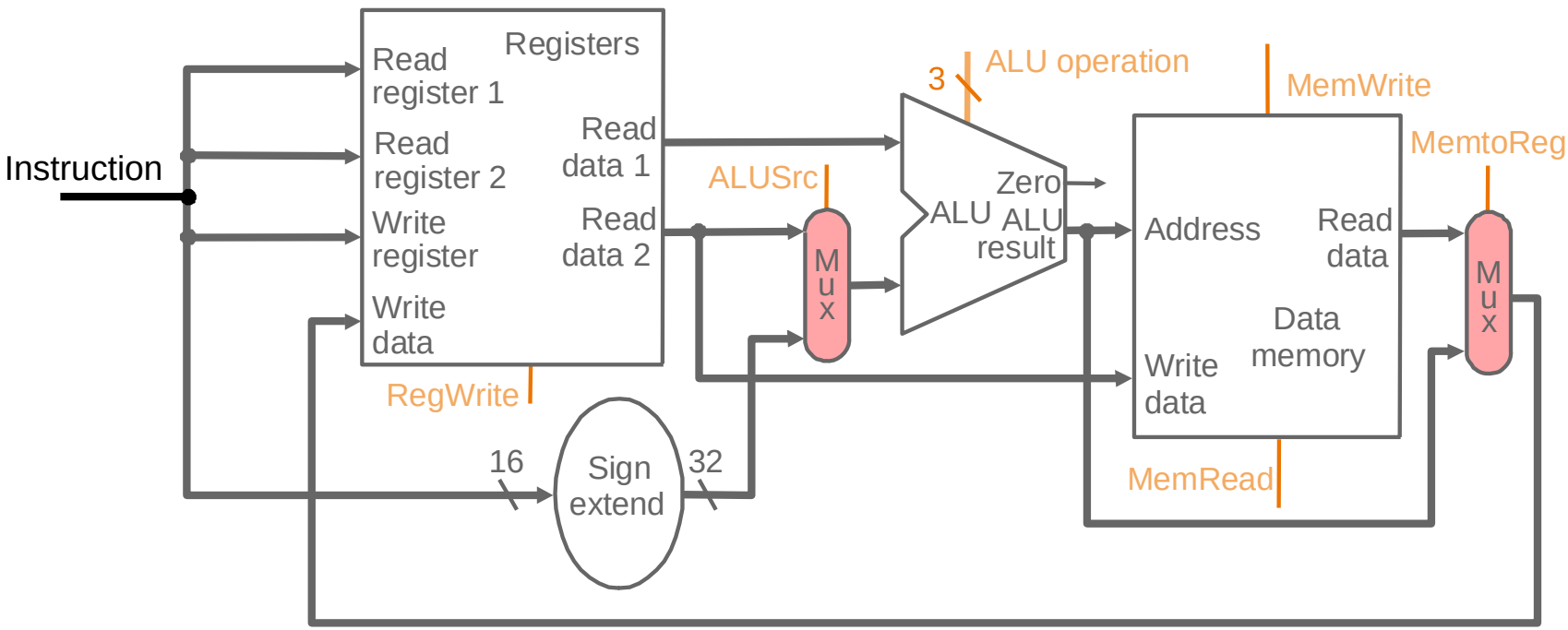
- Kombination von R-Typ- und Speicherzugriffs-Befehlen

Der 2. ALU-Eingang ist entweder ein Register oder eine Befehlswort-Konstante (nach Vorzeichenerweiterung)
→Multiplexer vor dem zweiten ALU-Eingang

Der Wert, der in ein Zielregister geschrieben wird, stammt entweder aus der ALU oder aus dem Datenspeicher.
→ Multiplexer vor dem Dateneingang des Registerspeichers

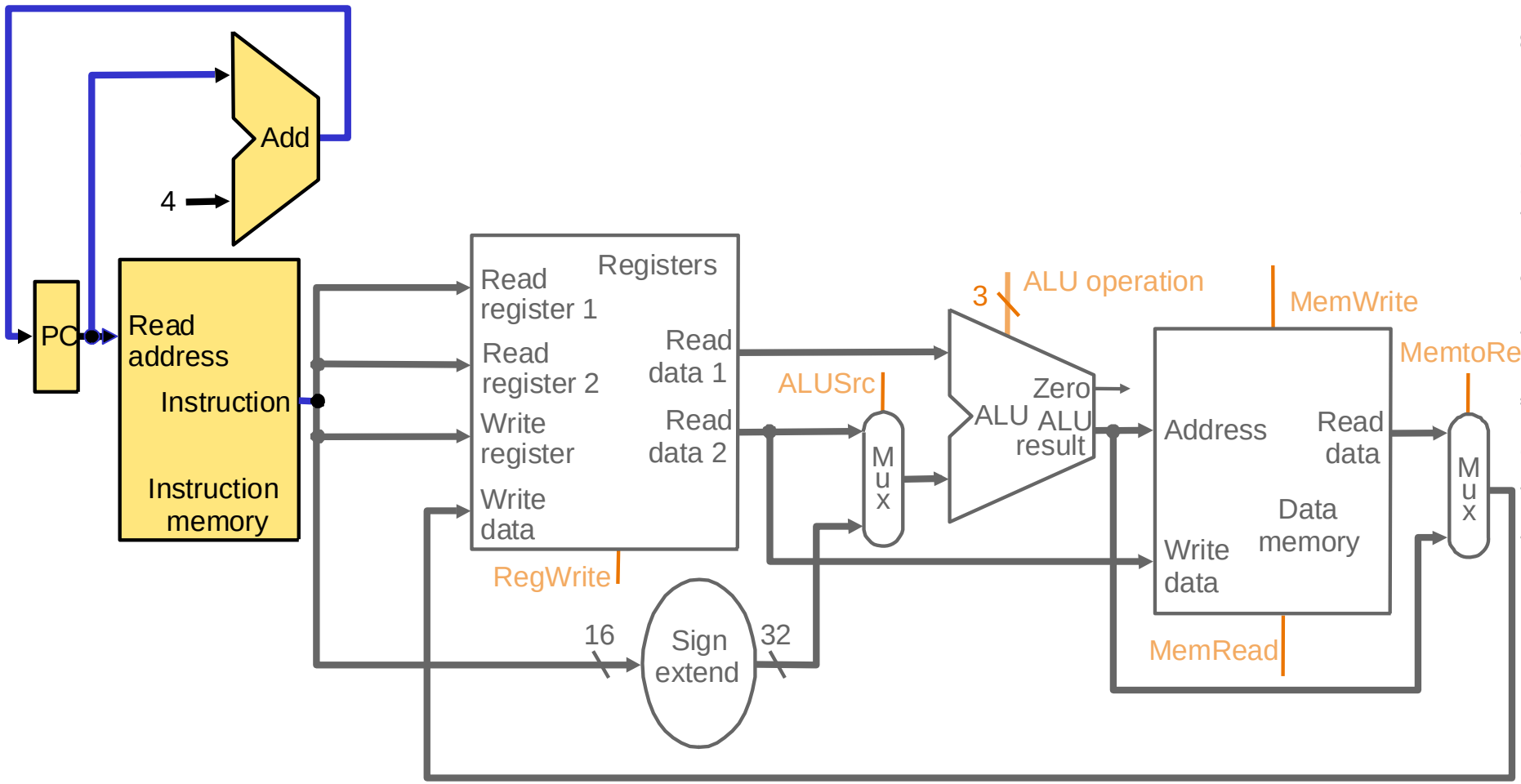
r3 ← ALUop(r1,r2)

r3 ← Dmem[ALUplus(r1,sext(Offset))]
r2 → Dmem[ALUplus(r1,sext(Offset))]



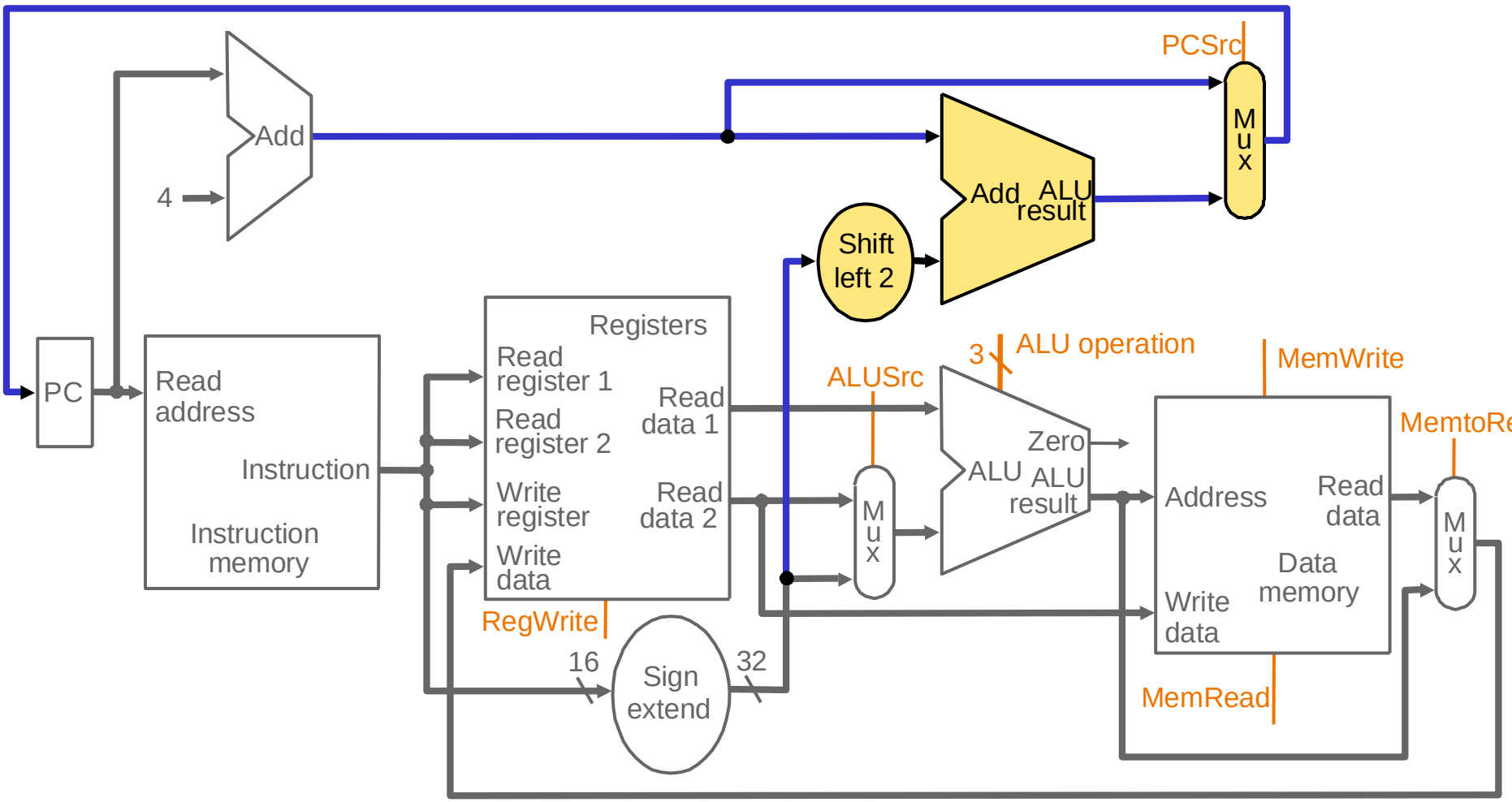
Operationswerk-Implementierung: Schritt (ii)

- Hinzufügen des Datenwegs für das Holen des Befehls mit PC, PC-Addierer und Programmspeicher



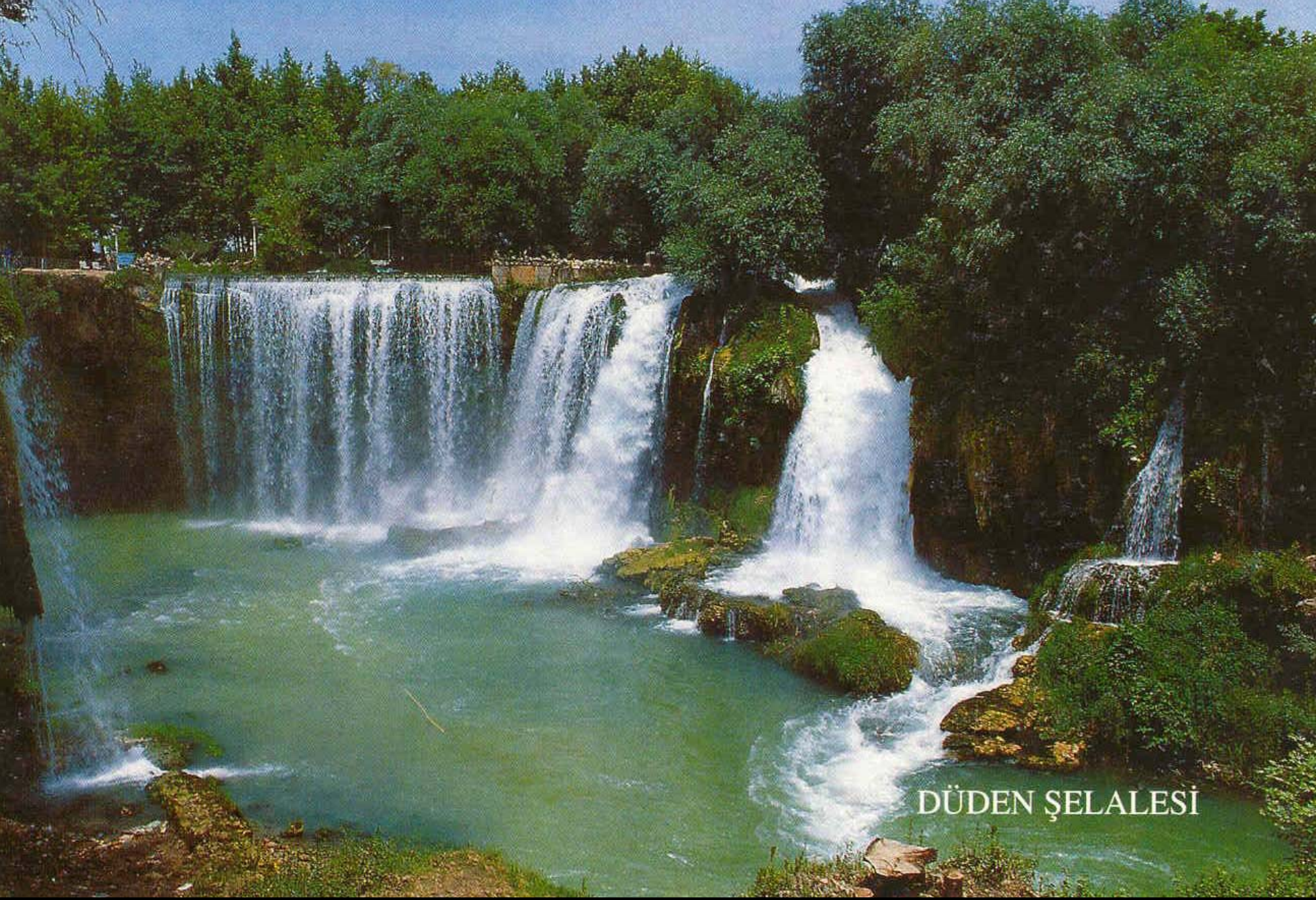
Operationswerk-Implementierung: Schritt (iii)

- Hinzufügen der **beq**-Einheit
- Modifikation der **PC**-Einheit durch Kombination mit der Berechnung des Verzweigungsziels



Zusammenfassung

- Wir haben den Datenpfad (das Operationswerk) des MIPS-Rechners schrittweise aus den Mikrooperationen zusammengesetzt, die die Befehle erfordern.
- Die ALU wird für sowohl für die eigentlichen Rechenoperationen wie auch für die Adressberechnung bei den Load/Store-Befehlen benutzt.
- Für die Berechnung der relativen Sprungadresse wird ein separater Addierer verwendet.


DÜDEN ŞELALESİ

Anhang: über Register

- Speicherelemente
- Synchrone Register
- Latches und Flip-Flops
- D-Flip-Flop
- Flankengetriggerte Register

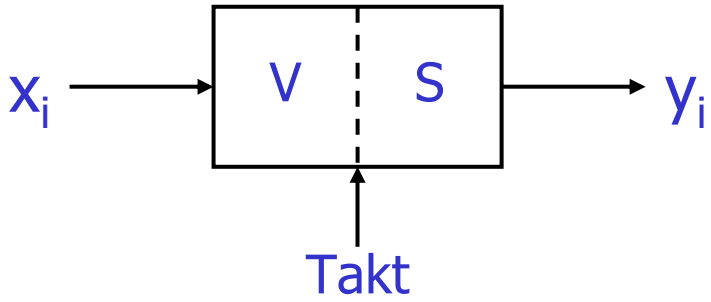
Speicherelement

- Ein Speicherelement hat mindestens 2 Eingänge und 1 Ausgang:
  - Eingang für Datenwert, der ins Element geschrieben werden soll
  - Steuer-Eingang, bestimmt wann der Datenwert wie gespeichert wird und wann der gespeicherte Wert am Ausgang erscheint (Taktsignal, Schreib/Lesesignal, Abtastsignal)
  - Ausgang für den gespeicherten Datenwert
- Beispiele für Speicherelemente:
  - RS-Flipflop (asynchron)
  - Latch (RS-Flipflop mit Aktivierungssignal)
  - Register (synchron)
  - Speicher (asynchron oder synchron)
- Logische Komponenten mit Speicherelementen heißen sequentielle Logik (sequential circuits, Schaltwerke), da ihre Ausgabe von ihren Eingängen und ihrem internen Zustand abhängt.

Sequentielle Logik

- Wenn für die Berechnung eines Ausgangs nicht nur der aktuelle Eingang sondern z. B. eine frühere Eingabe maßgebend ist, müssen Schaltungen mit Rückkopplungen betrachtet werden.
- Direkte asynchrone Rückkopplungen werden in größeren Schaltungen meist verboten, da sie nur sehr schwer sicher betrieben werden können oder einen hohen Realisierungsaufwand erfordern.
- Die Rückkopplungen werden in der Regel durch Taktimpulse synchronisiert, die von einen zentralen Taktgeber (clock generator) ausgehen.
- Zur Datenspeicherung und Entkopplung der Eingänge von den Ausgängen werden meist synchrone Register verwendet.
- Schaltungen mit synchronen Registern nennt man (synchrone) Schaltwerke oder einfach synchrone Schaltungen

Synchrone Register

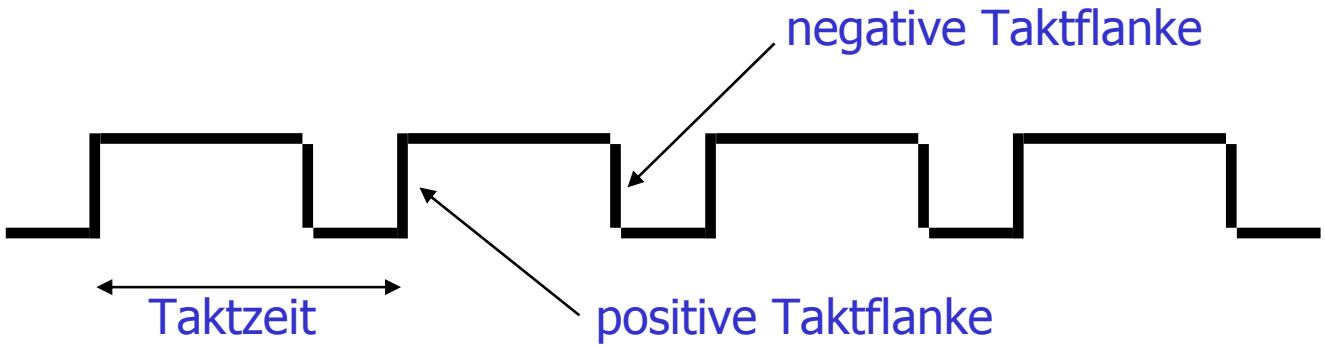


- Ein synchrones Register besteht aus einem Vorspeicher V (Master) und Speicher S (Slave).
  V und S sind durch eine Sperre getrennt.
- Arbeitsweise in 2 Phasen, die durch den Takt definiert werden:

  Arbeitsphase (Berechnungsphase) Φ1:
  - Inhalt y_i von S wird „nach rechts" (z. B. in Schaltnetz) abgegeben und steht als Signal für längere Zeit zur Verfügung.
  - Der neu berechnete Signalwert x_i wird in V „abgelegt" (gepuffert). (Die Pufferung erfolgt entweder asynchron oder wird gezielt am Ende von Φ1 oder ganz am Anfang von Φ2 (vor der Setzphase) durchgeführt)

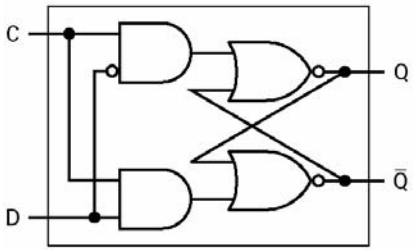
  Setzphase (Übernahmephase, Kopierphase, Speicherphase) Φ2:
  - Die Sperre wird kurzzeitig aufgehoben. Dadurch wird der Inhalt von V an S weiter gegeben („Setzen").
- Die Setzphase ist i. allg. wesentlich kürzer als Arbeitsphase.

Taktsignal

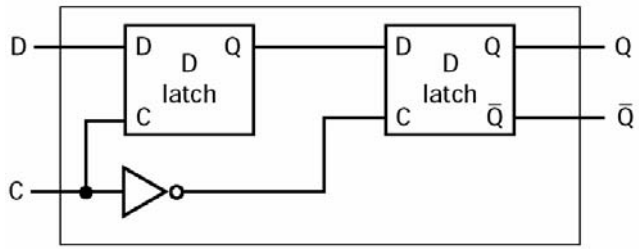


- Taktsignal oszilliert zwischen hohem und niedrigem Wert.
- Taktfrequenz = 1/Taktzeit
- Im Grundrhythmus der Taktzeit (clock cycle) vollziehen sich synchron alle rechnerinternen Abläufe.
- Taktzeiten liegen meist in Größenordnungen von 10^{-8} bis 10^{-9} Sekunden.
- Aufgrund langer Signalwege bzw. Verzögerungen beim Durchlauf von Schaltnetzen kann u. U. nicht in jedem Takt ein Fortschritt erzielt werden (z. B. Addition zweier Dualzahlen kann mehrere Takte erfordern.)

Latches und Flip-Flops



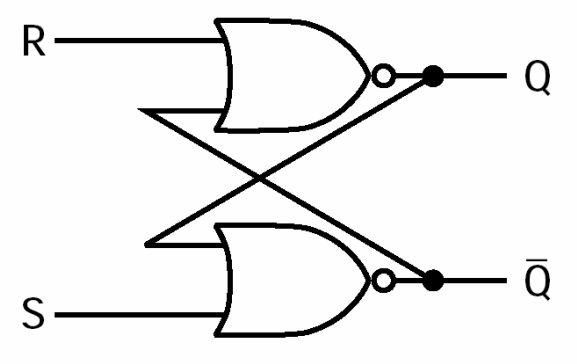
D-Latch



D-Flip-Flop

- Einfachste Speicherelemente sind Latches und Flip-Flops.
- Bei beiden ist Ausgabe gleich dem im Element gespeicherten Zustand.
- Zustandswertänderung erfolgt taktgesteuert.

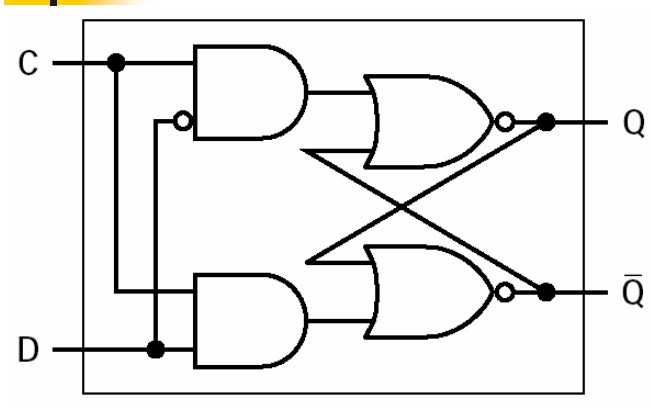
Einfaches, ungetaktetes Speicherelement



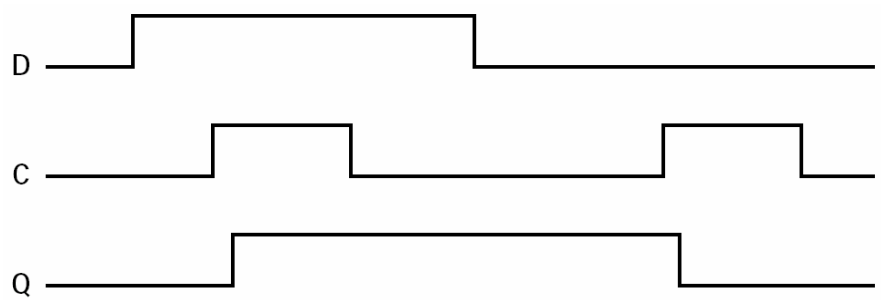
- Das S-R Latch (set-reset):
  - Ausgang abhängig von momentaner Eingabe und gespeicherter, früherer Eingabe
  - bestehend aus zwei kreuz-gekoppelten NOR-Gatter (oder NAND-Gattern)

- Funktionsweise:
  - S, R ungesetzt: Q, ¬Q behalten ihre Werte
  - S gesetzt: Q gesetzt, ¬Q ungesetzt
  - R gesetzt: ¬Q gesetzt, Q ungesetzt
  - S und R sollten nicht gleichzeitig gesetzt werden, da dann Q und ¬Q nicht mehr invers zueinander sind und bei der gleichzeitigen Wegnahme von R und S der Ergebniszustand nicht determiniert ist.
- Basis für komplexere Speicherelemente

Latch

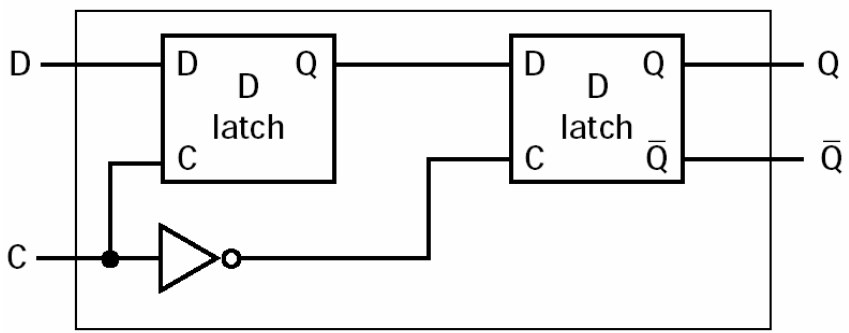


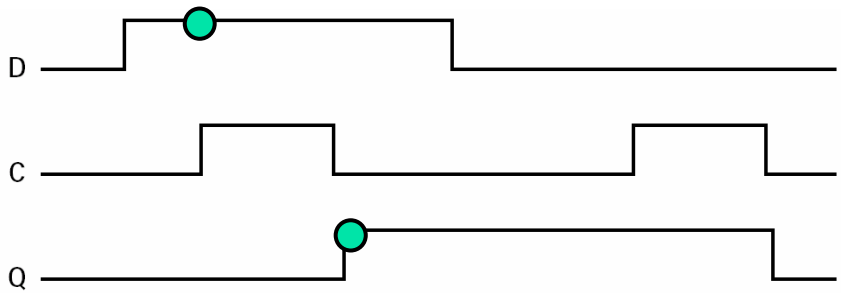
- Zwei Eingänge:
  - der zu speichernde Datenwert D
  - der Taktimpuls C, bei dem D gelesen und gespeichert werden kann
- Zwei Ausgänge:
  - der Wert des internen Zustands Q und sein (logisches) Komplement



- Wenn Taktimpuls C gesetzt: Latch ist offen und Ausgang Q wird auf Eingang D gesetzt.
- Solange Taktimpuls C nicht gesetzt: Latch ist geschlossen und Ausgang Q behält zuletzt gesetzten Wert.

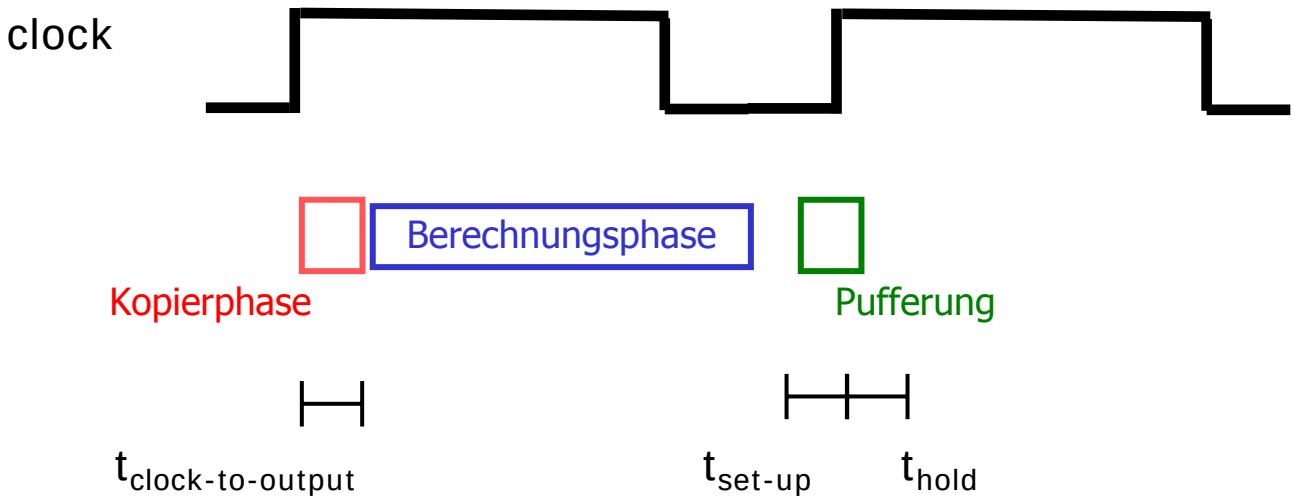
D-Flip-Flop





- **(Pufferung von D)** Erstes Latch („Master“) ist offen und setzt seinen internen Ausgang Q auf D, falls Taktimpuls C=1.
- **(Kopieren)** Sobald Takt auf Null wechselt (C→0) fällt, wird erstes Latch geschlossen und zweites Latch („Slave“) geöffnet. Der Slave übernimmt den im Master gespeicherten Wert und stellt ihn am Ausgang über ein Taktintervall zur Verfügung.

Flankengetriggerte Register, allgemein



- **Positiv flankengetriggert**
  - Nach der positiven Taktflanke wird zuerst die Pufferung des neuen Wertes abgeschlossen und die Kopierphase angestoßen. Spätestens nach der Zeit $t_{clock\text{-}to\text{-}output}$ ist der Datenausgang stabil.
  - Danach beginnt die Berechnungsphase. Sie muß rechtzeitig vor der nächsten positiven Taktflanke (Set-up time) ein stabilen Wert liefern, damit der neue Wert sicher gepuffert werden kann.
  - Der stabile Eingangswert muß manchmal noch eine gewisse Zeit nach der positiven Taktflanke stabil bleiben (hold-time), damit die Pufferung sicher abgeschlossen werden kann.

Vergleich Latches und Flip-Flops

- **Getaktetes Latch:** besitzt nur einen internen Speicher. Ausgang folgt dem Eingang wenn C=1. Mit C→0 wird der letzte Wert gespeichert
- **Synchrone Flip-Flops:** besitzen zwei interne Speicher (z. B. mit Latches oder Ladungsspeichern implementiert)
  - **Einflankengetriggert:** Nach der Taktflanke wird zuerst die Pufferung des neuen Wertes abgeschlossen und kurze Zeit später in den Slave kopiert.
  - **Zweiflankengetriggert:** Eine Taktflanke definiert den Beginn der Übernahme in den Master, die andere den Beginn des Kopiervorganges vom Master in den Slave. Vorteil: Ein Taktversatz (das Taktsignal kommt leicht zeitlich versetzt bei verschiedenen Flipflops an (clock skew) kann durch das Tastverhältnisses des Taktes (High Dauer zu Low Dauer) sicher beherrscht werden)