Rechnerarchitektur

Einführung in die Informatik & Rechnerarchitektur (EIR1/EIF1)

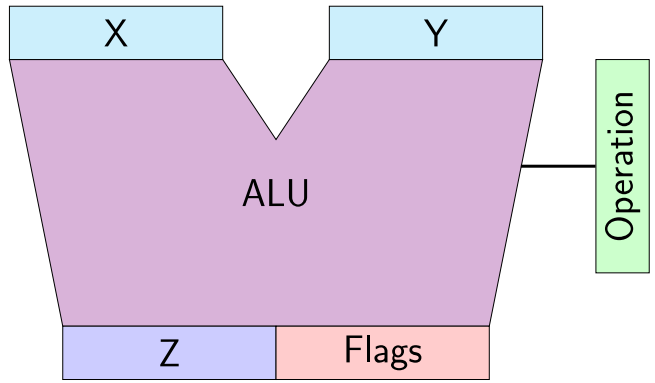
Erik Pitzer

SE & MBI – FH Hagenberg – WS 2025/26

Mikroarchitektur

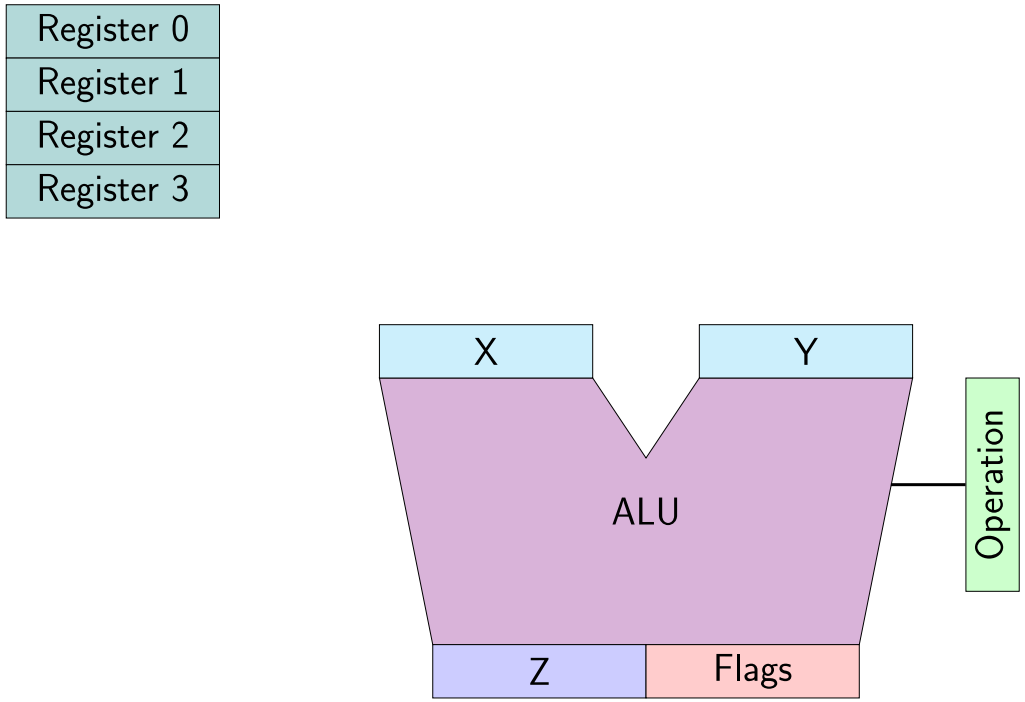
Aufbau einer CPU: ALU (1/10)

- Zentrales Element ALU mit
  - Eingangsregister: X, Y und Operation
  - Ausgangsregister: Z und Flags



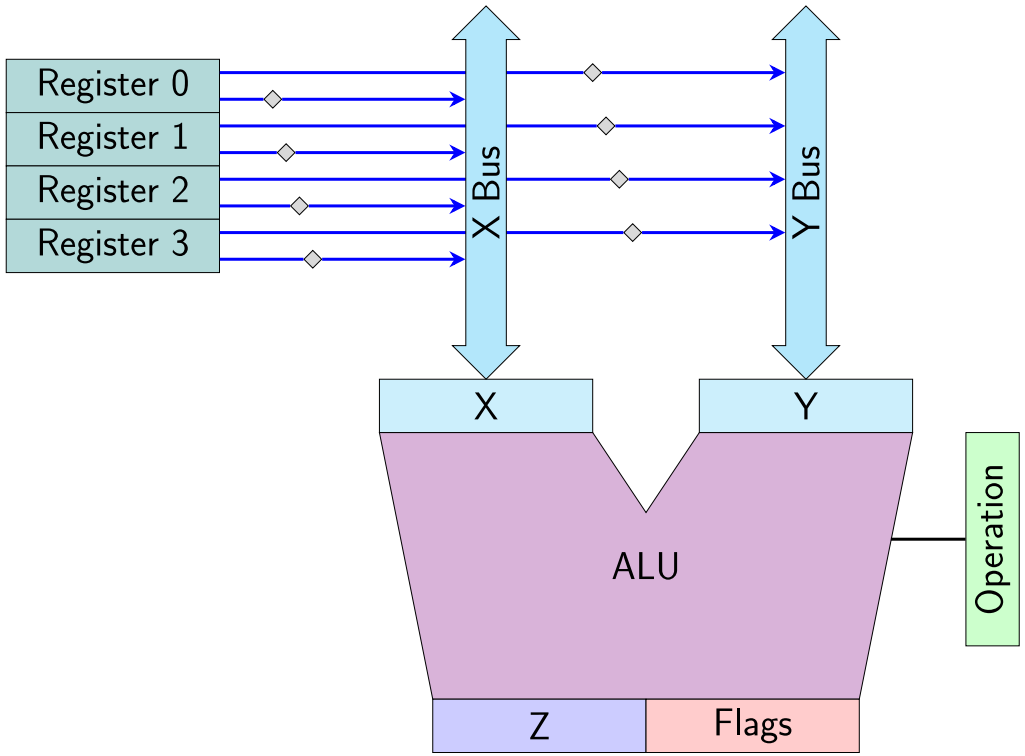
Aufbau einer CPU: Register (2/10)

- Register (z.B. R0–R3) enthalten Werte zur unmittelbaren Verarbeitung



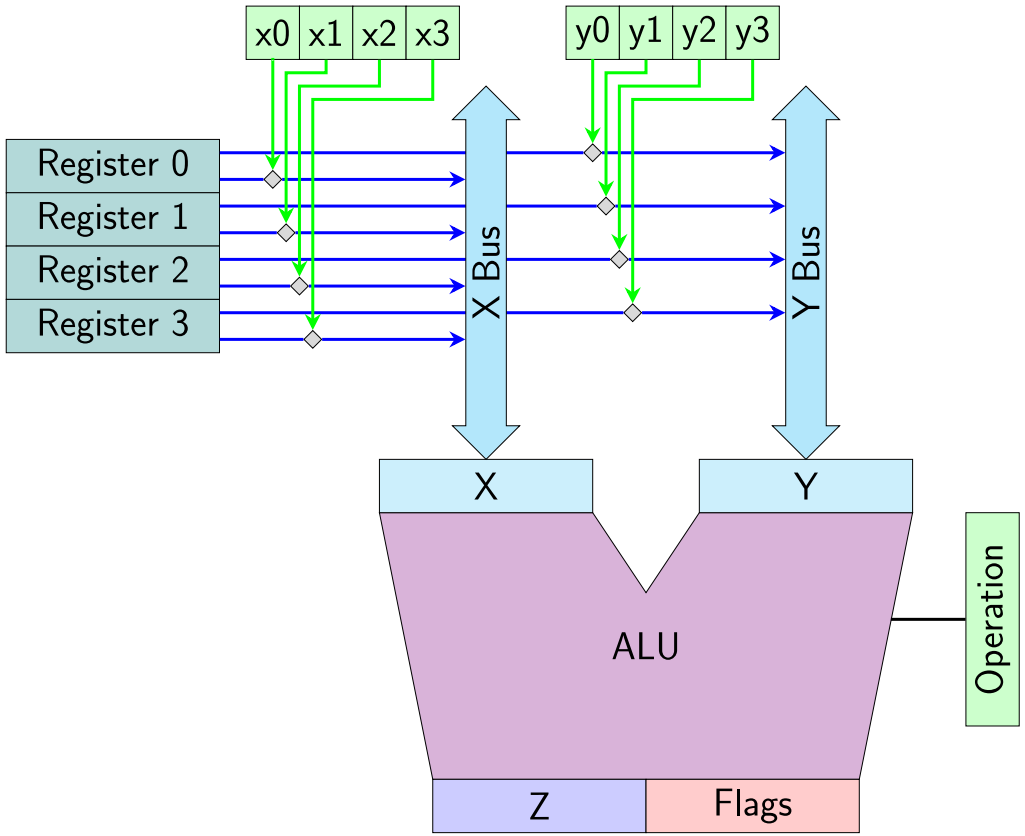
Aufbau einer CPU: X- & Y-Bus (3/10)

- Verbindung der allgemeinen Register (R0–R3) mit X- & Y-Register der ALU über X- und Y-Bus
  - Datenleitungen (⟶) haben Puffer (◇) zum Ein- & Ausschalten



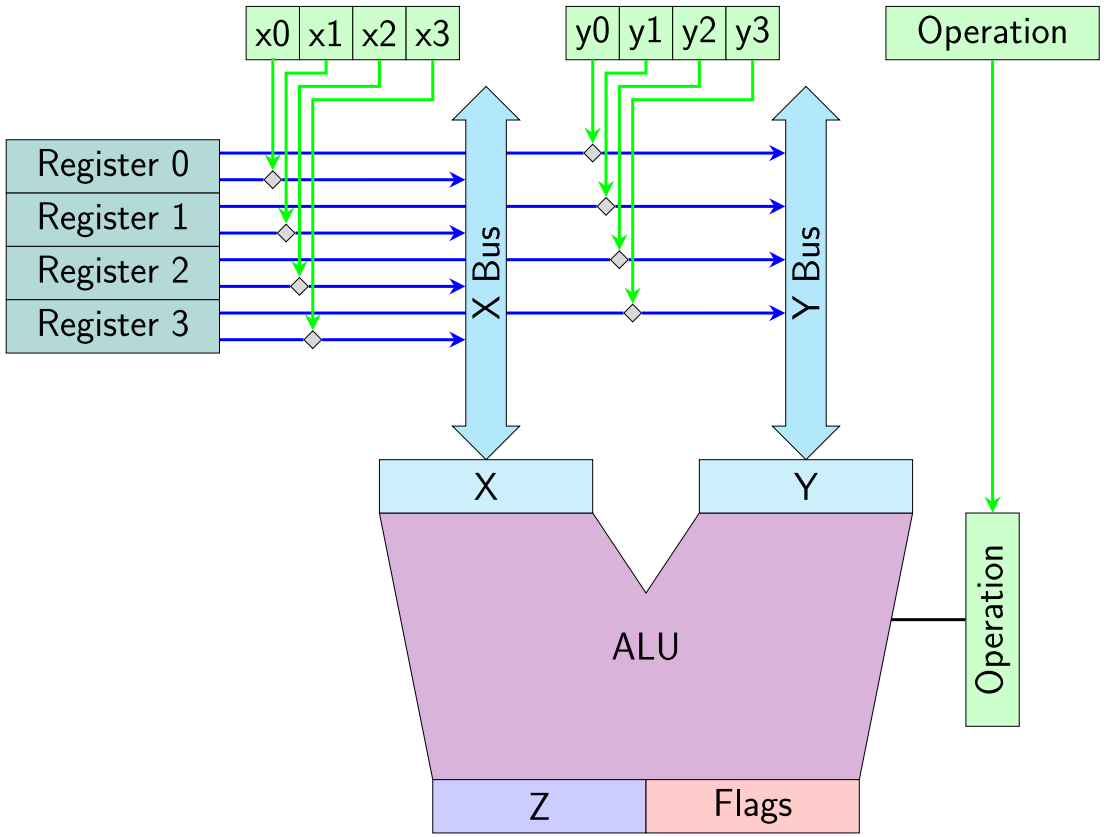
Aufbau einer CPU: X- & Y-Bus Steuerung (4/10)

- Auswahl der Register deren Werte zur ALU (zu X, Y) geleitet werden sollen über einzelne Bits/Schalter



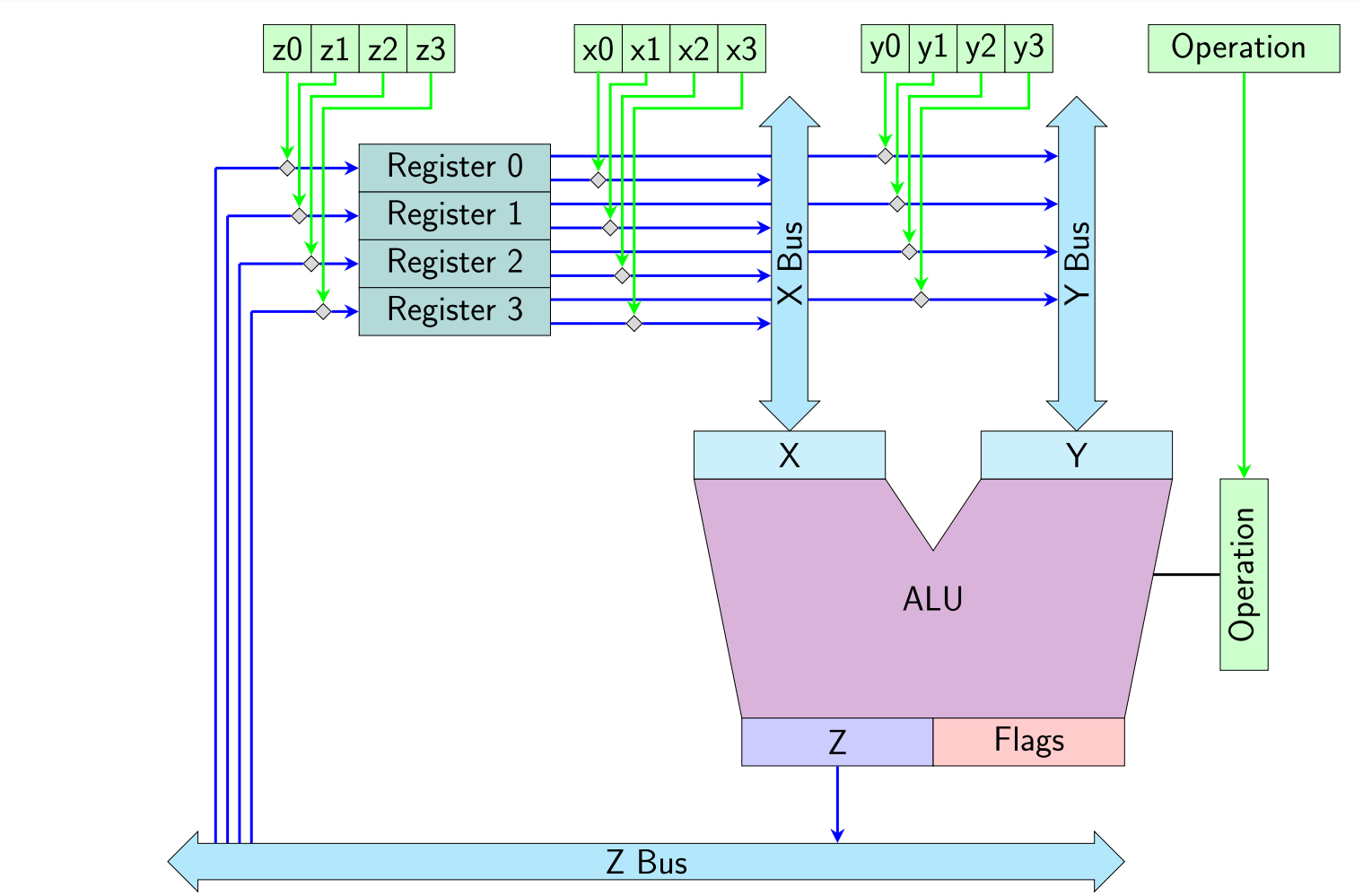
Aufbau einer CPU: ALU Operation (5/10)

- Auswahl der Operation durch weitere Schalter (Nr. der Operation)



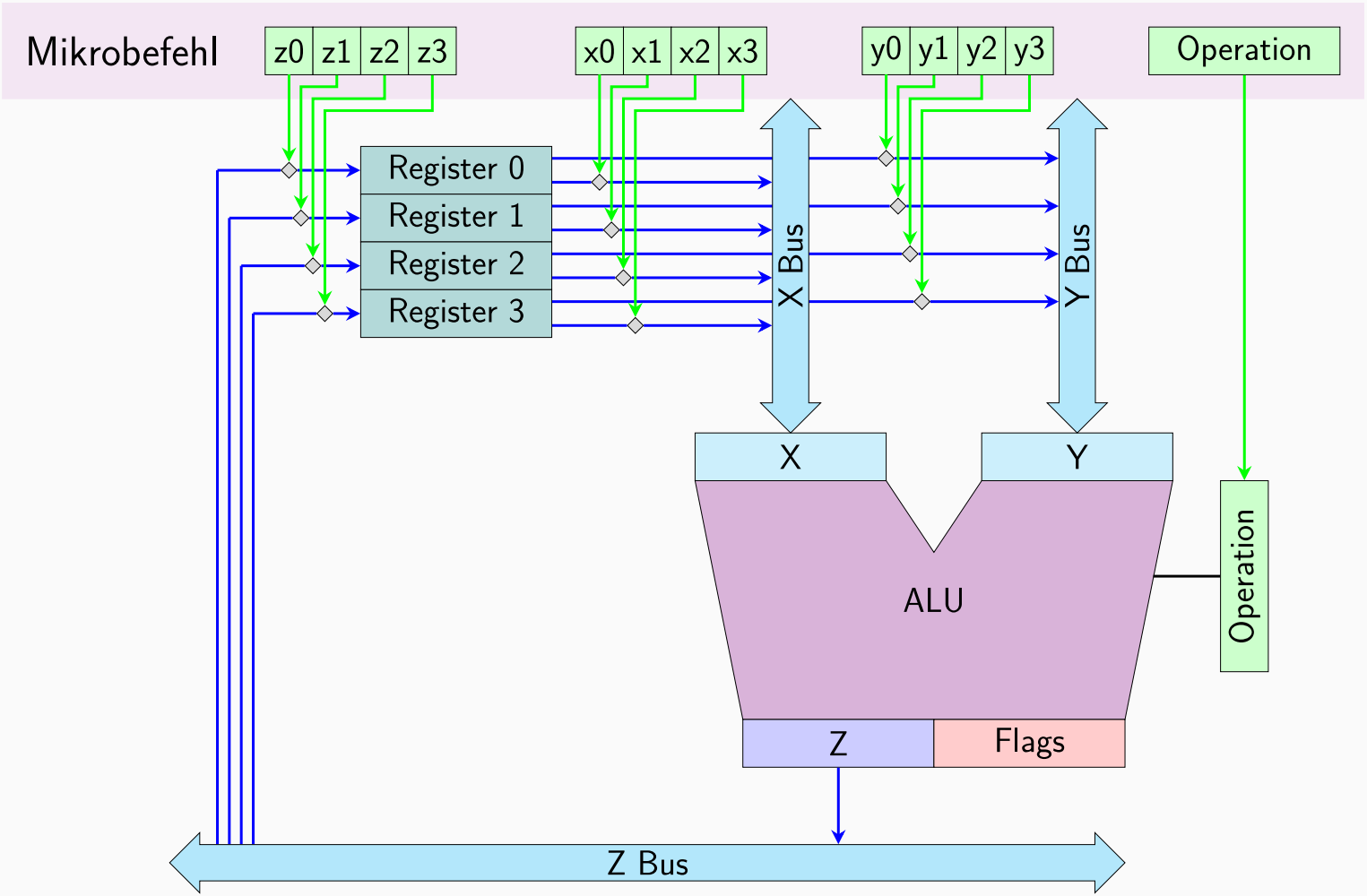
Aufbau einer CPU: Z-Bus & Steuerung(6/10)

- Auswahl auf welche Register (mehrere möglich) das Ergebnis von Z geleitet werden soll



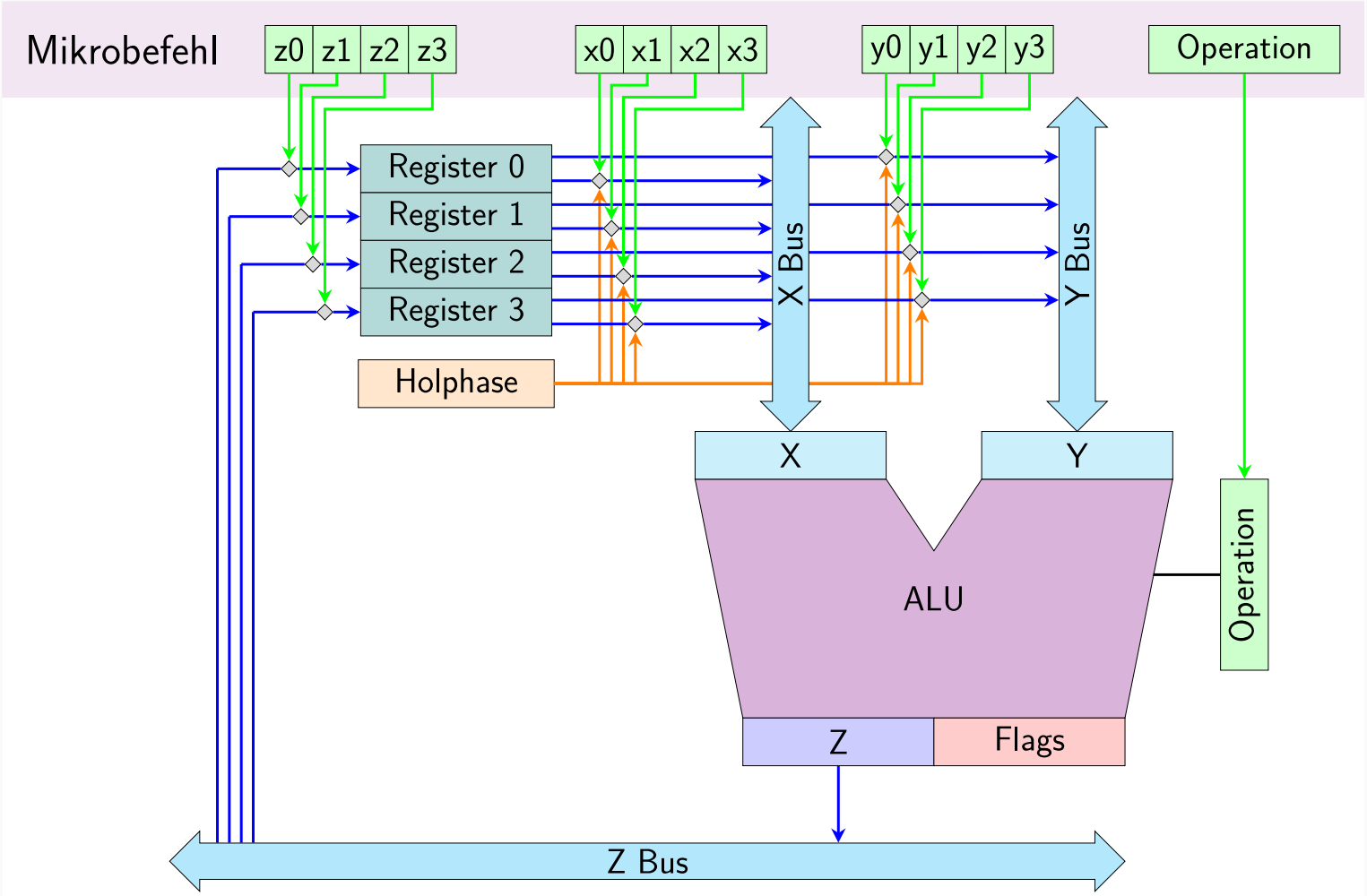
Aufbau einer CPU: Mikrobefehl (7/10)

- Zusammenfassen aller Schalter zum Mikrobefehl



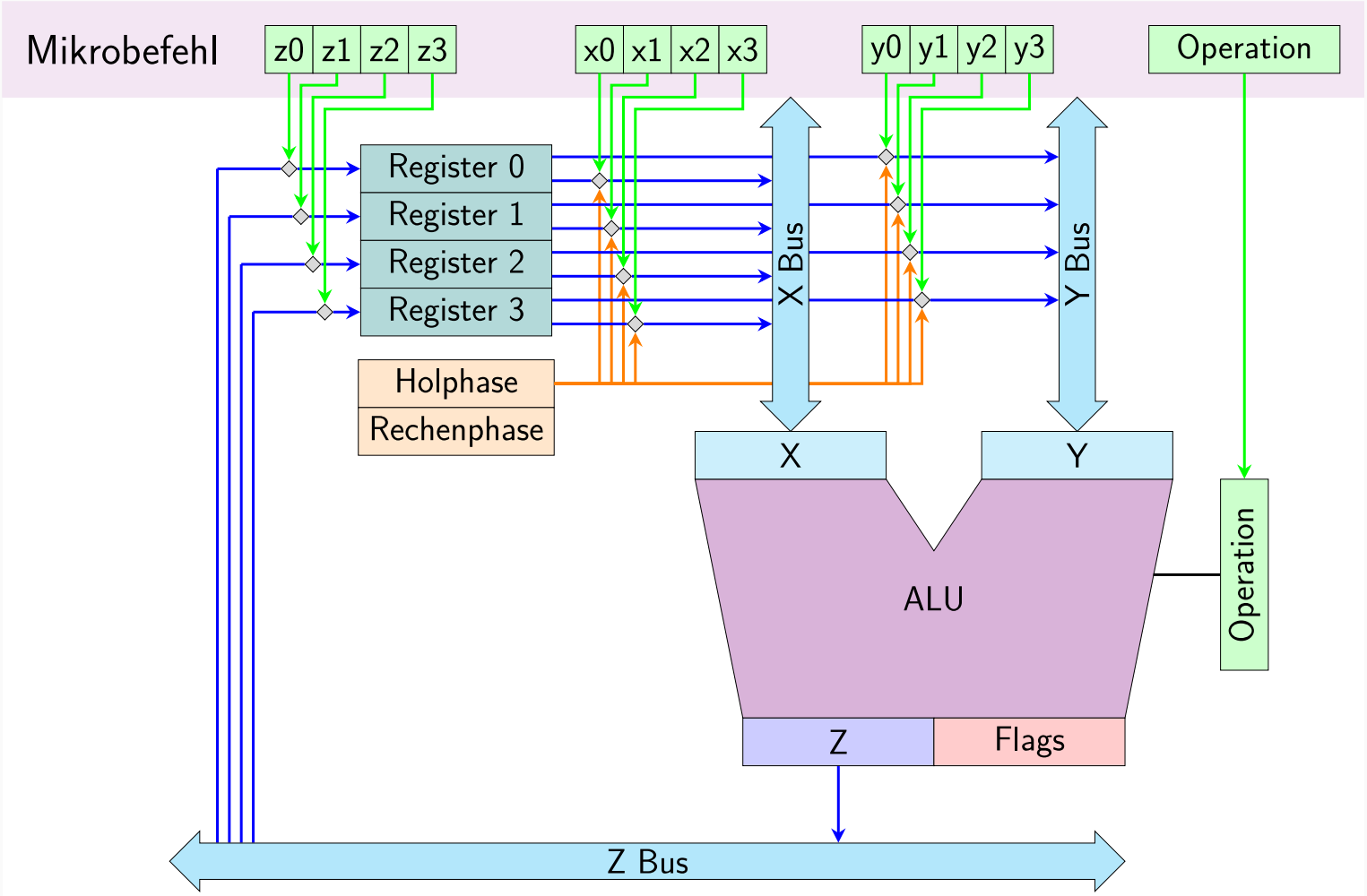
Aufbau einer CPU: Holphase (8/10)

- Koordination des Ablaufs in einzelne Phasen
  - Holphase: Werte von allg. Register zur ALU



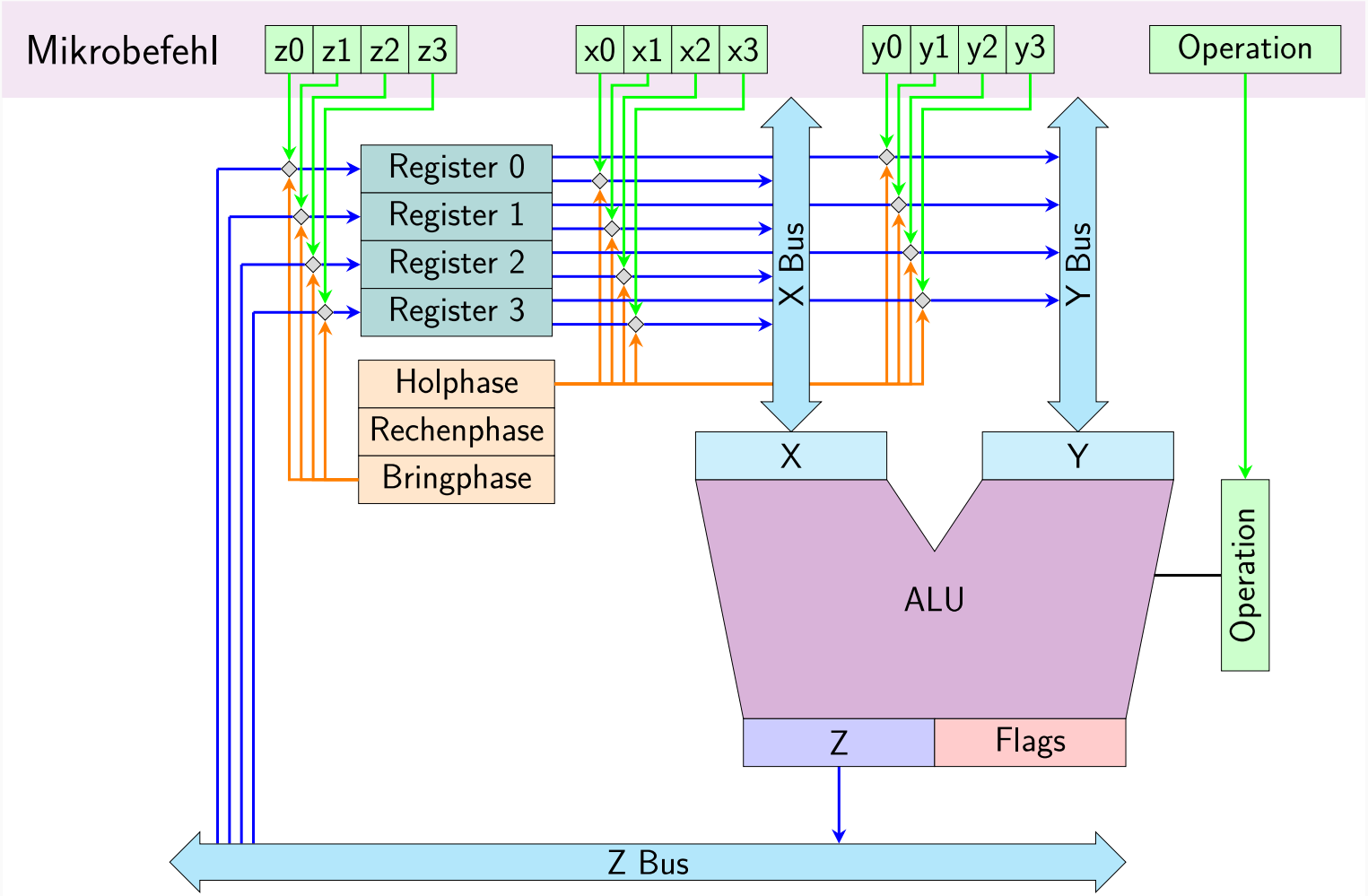
Aufbau einer CPU: Rechenphase (9/10)

- Koordination des Ablaufs in einzelne Phasen
  - Rechenphase: ALU nimmt Input (X,Y) berechnet Ergebnis (Z)



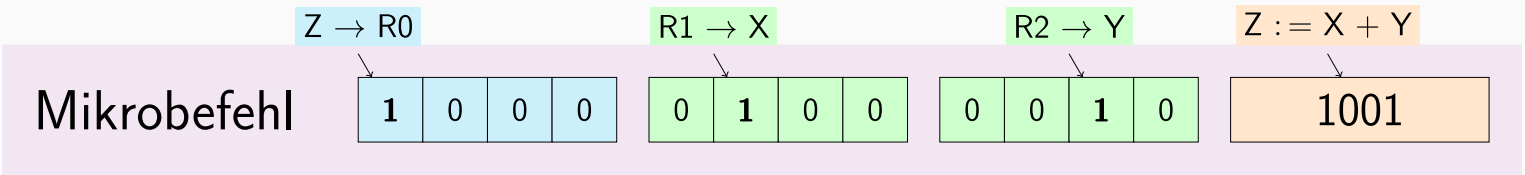
Aufbau einer CPU: Bringphase (10/10)

- Koordination des Ablaufs in einzelne Phasen
  - Bringphase: Ergebnis (Z) wird zu Registern (R0–R3) gebracht



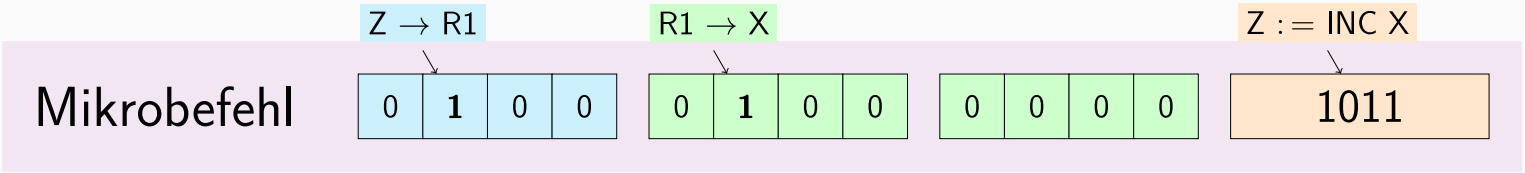
Notation Mikrobefehl

- Pseudosprache für Notation von Schaltern / Bits
- Einzelne Bits des Mikrobefehls werden benannt, z.B.
  - R1 $\rightarrow$ X, R2 $\rightarrow$ Y
  - Z $\rightarrow$ R0
- ALU Operationen sind durchnummeriert und werden binär codiert, z.B.
  - Operation 0: Konstante generieren: `Z := 0`
  - Operation 6: Bitweises Und: `Z := X AND Y`
  - Operation 9: Addition: `Z := X + Y`

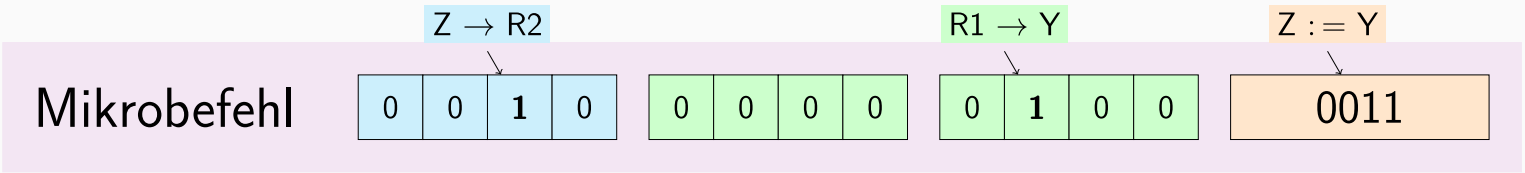


Mikrobefehl Beispiele

- H: R1 $\rightarrow$ X
- R: Z := INC X
- B: Z $\rightarrow$ R1

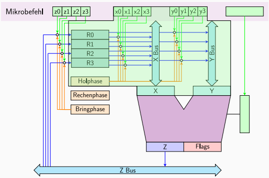


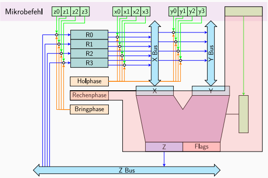
- H: R1 $\rightarrow$ Y
- R: Z := Y
- B: Z $\rightarrow$ R2

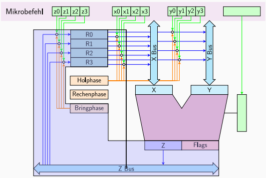


Aufteilung des Takts in Phasen

- Jeder Takt wird in drei Phasen unterteilt
  - Holphase
  - Rechenphase
  - Bringphase
- Mikrobefehl liegt währende des gesamten Takts an
- wird phasenweise “abgearbeitet”

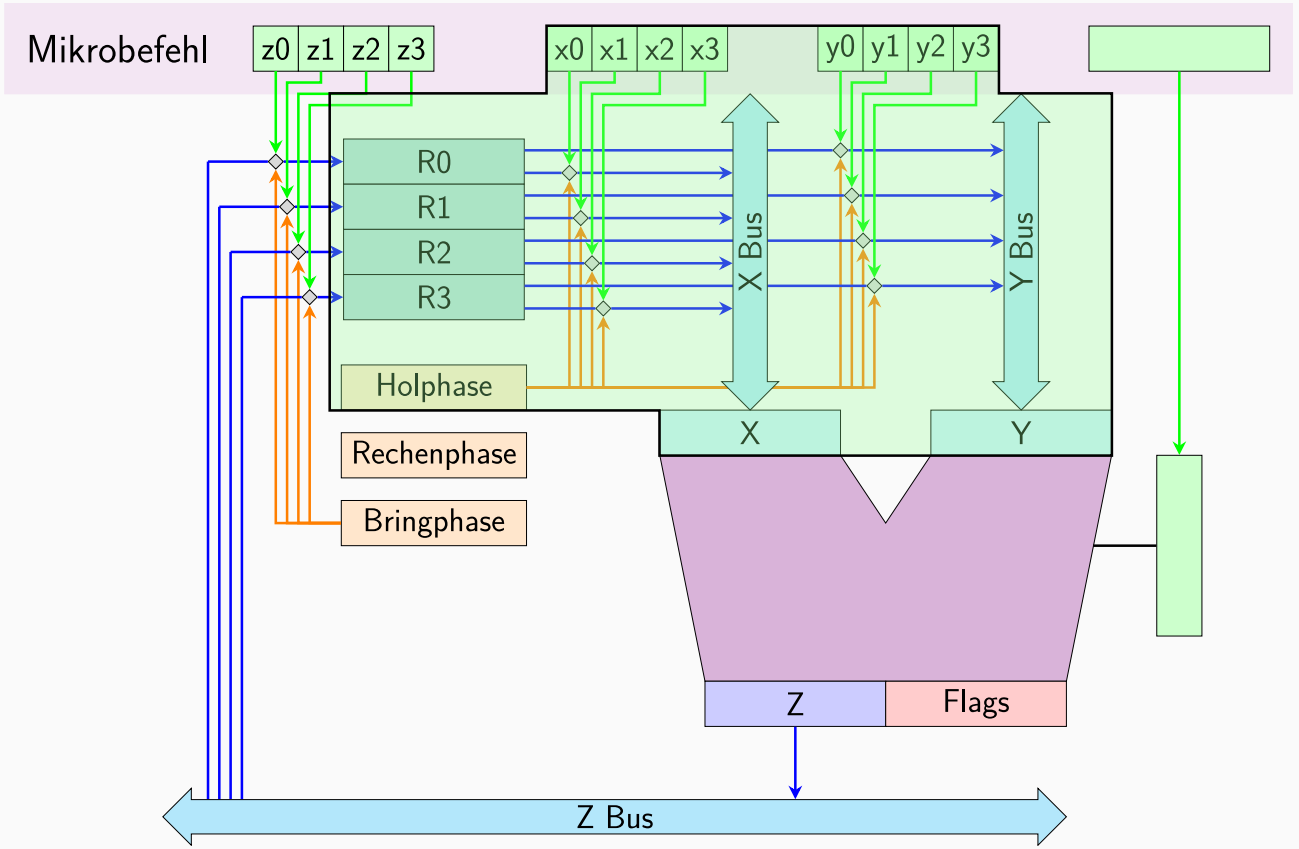






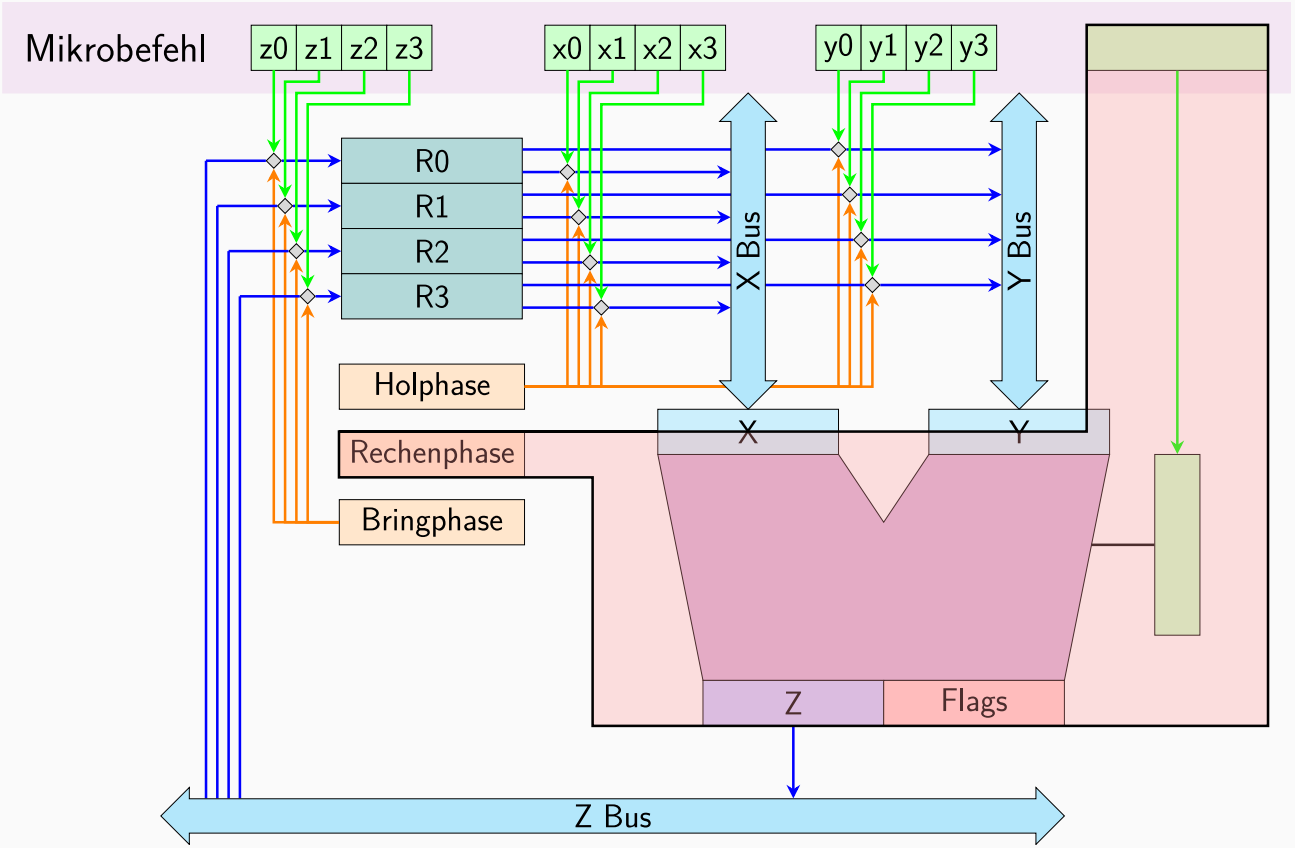
Holphase

- Werte werden aus den allgemeinen Registern (R0–R3) geholt liegen an X-Bus und Y-Bus an
  - Auswahl erfolgt durch Bit/Schalter im Mikrobefehl, im Pseudocode z.B. R0 $\rightarrow$ X, R1 $\rightarrow$ Y
- Werte gelangen in die X- und Y-Register der ALU



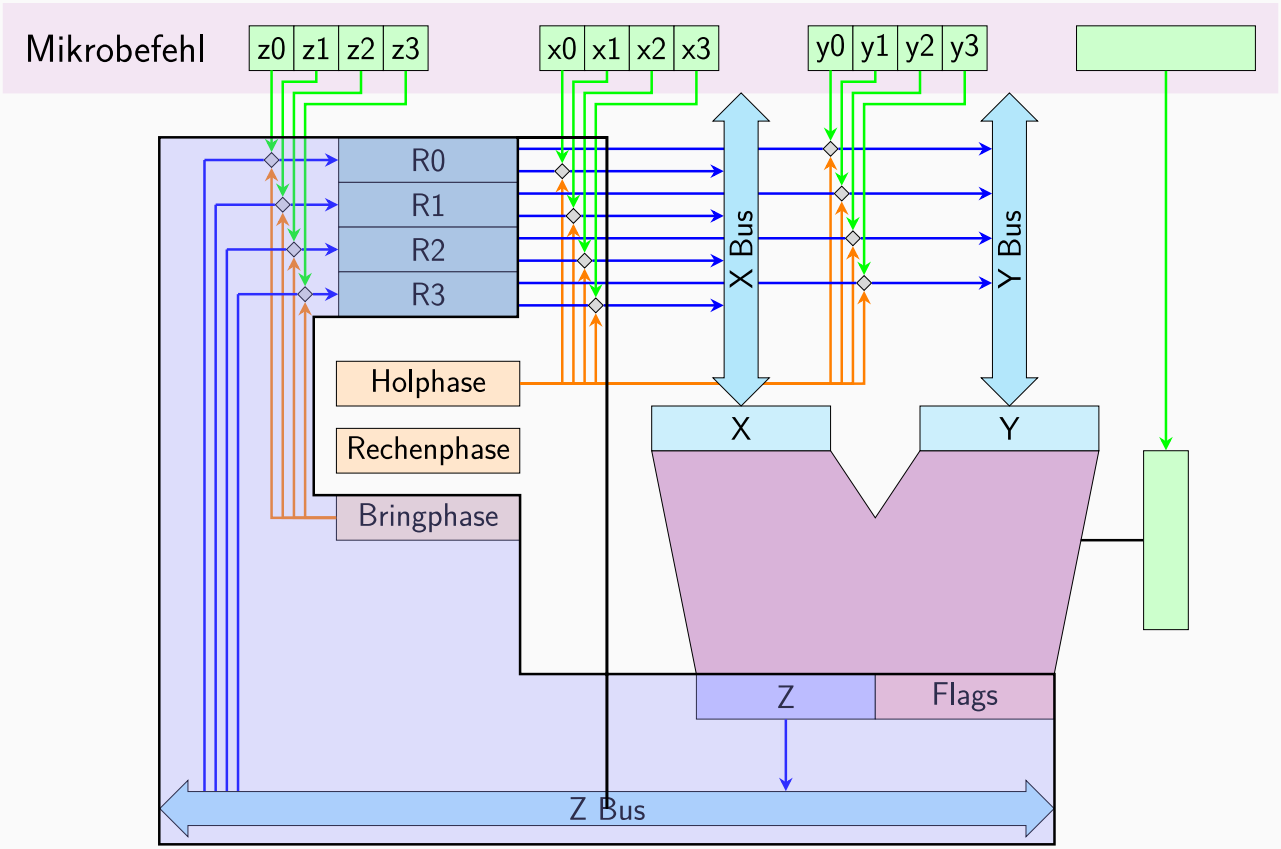
Rechenphase

- Werte aus X- und Y-Registern werden in der ALU verarbeitet (Berechnung, Durchschleusen, oder Konstante generieren)
  - Auswahl erfolgt durch Mikrobefehl, z.B. `Z := X + Y`
- Ergebnis stabilisiert sich im Z Register



Bringphase

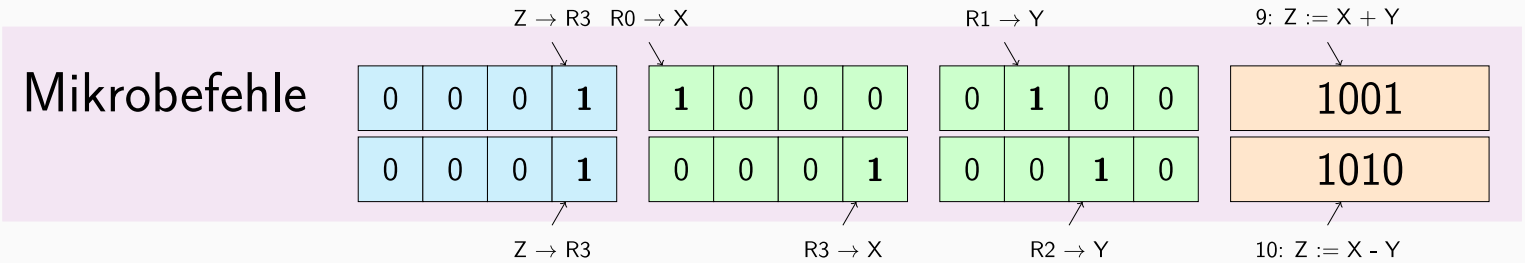
- Ergebnis im Z-Register der ALU wird über Z-Bus zurück zu Registern (R0–R3) gebracht
  - Auswahl (mehrere gleichzeitig möglich) erfolgt über Bits/Schalter im Mikrobefehl, z.B. Z → R2,R3



Beispiel: Zerlegen in Schritte und Phasen

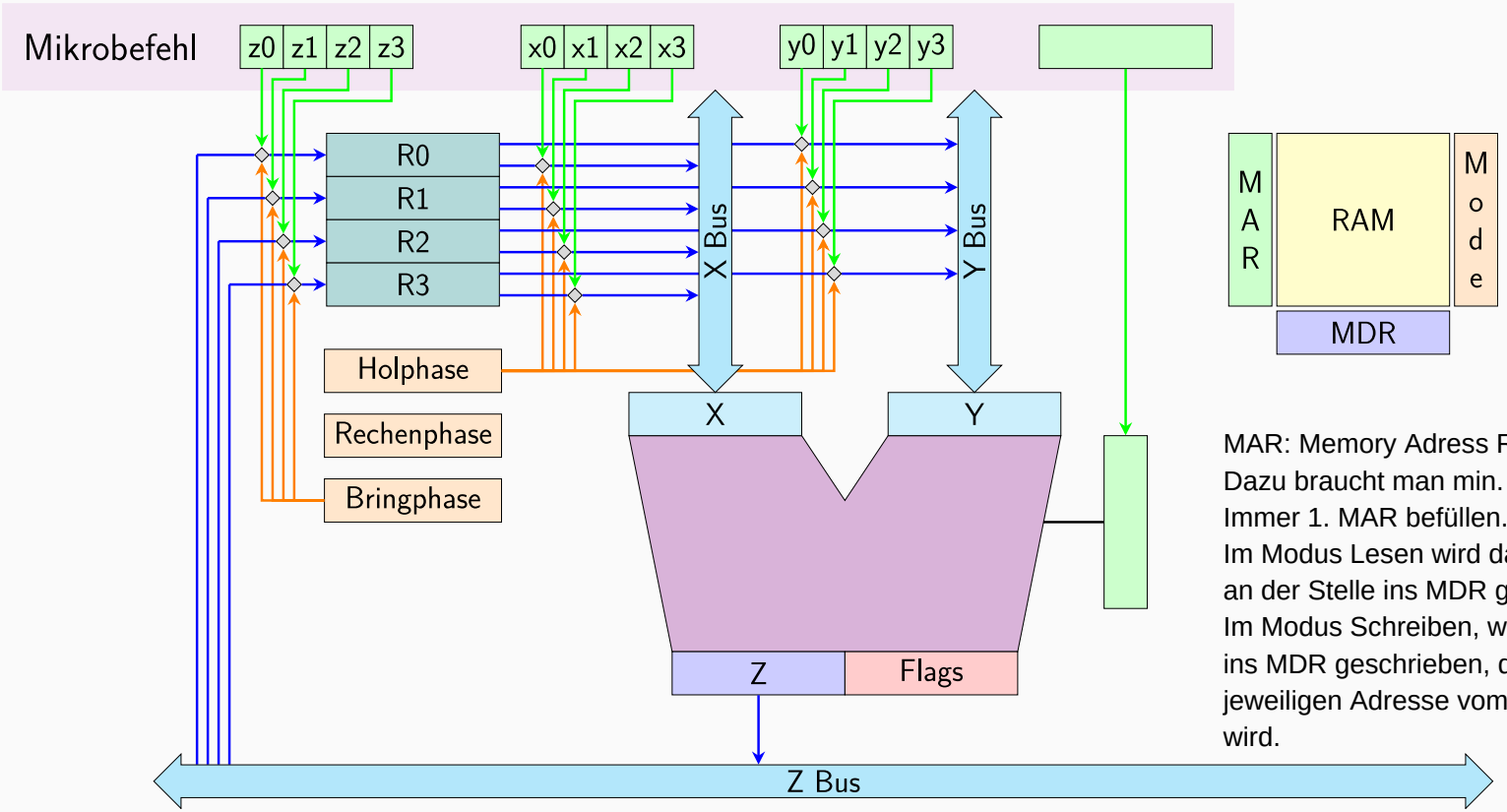
- Aufgabe: R0 + R1 - R2 → R3, ergibt folgende Schritte/Takte
  1. R0 + R1 → R3
  2. R3 - R2 → R3
- Schritte zerlegen in Phasen:
  1. R0 + R1 → R3
     - Holphase: R0 → X, R1 → Y
     - Rechenphase: Z := X + Y
     - Bringphase: Z → R3
  2. R3 - R2 → R3
     - Holphase: R3 → X, R2 → Y
     - Rechenphase: Z := X - Y
     - Bringphase: Z → R3

`demos/CPU.circ` ALU + Reg



CPU und Speicher (1/5)

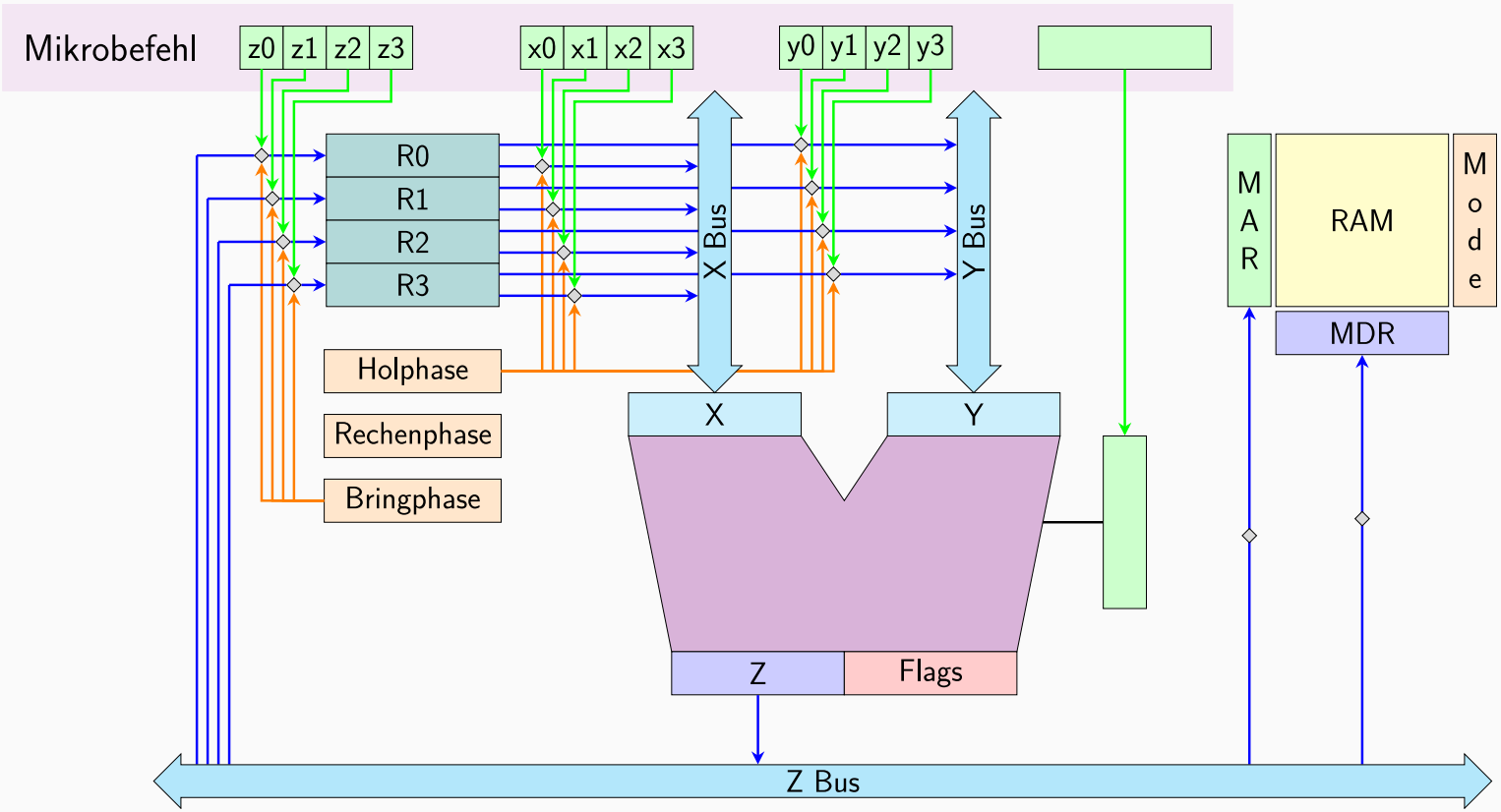
- RAM-Anbindung



MAR: Memory Adress Register
Dazu braucht man min. 2 Takte
Immer 1. MAR befüllen.
Im Modus Lesen wird dann der Wert an der Stelle ins MDR geschrieben.
Im Modus Schreiben, wird ein Wert ins MDR geschrieben, der dann an der jeweiligen Adresse vom MAR gespeichert wird.

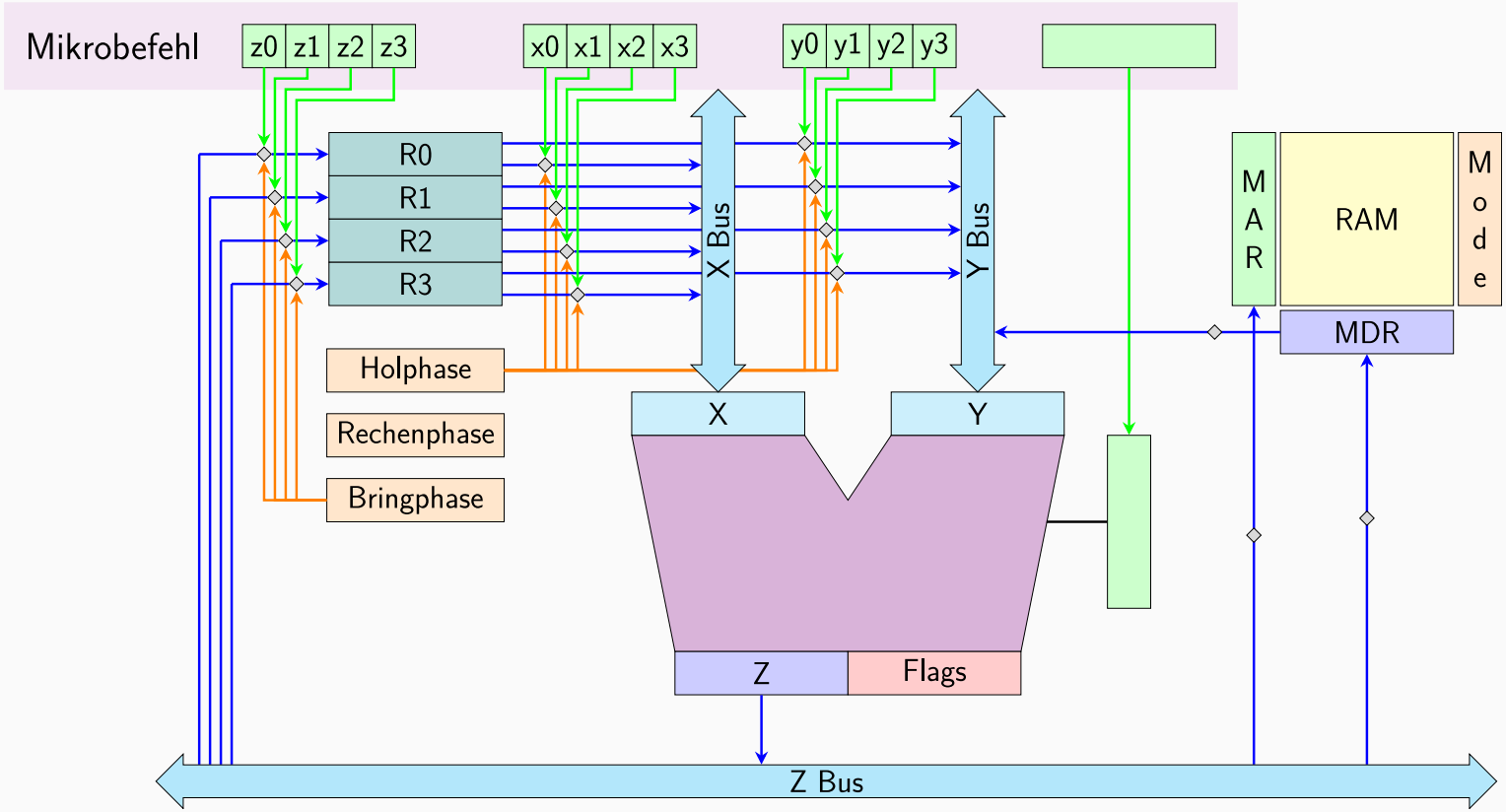
CPU und Speicher: Z→MAR, Z→MDR (2/5)

- neue Datenleitung zum Schreiben (Bringphase)
  - Z → MAR
  - Z → MDR (Bringphase)



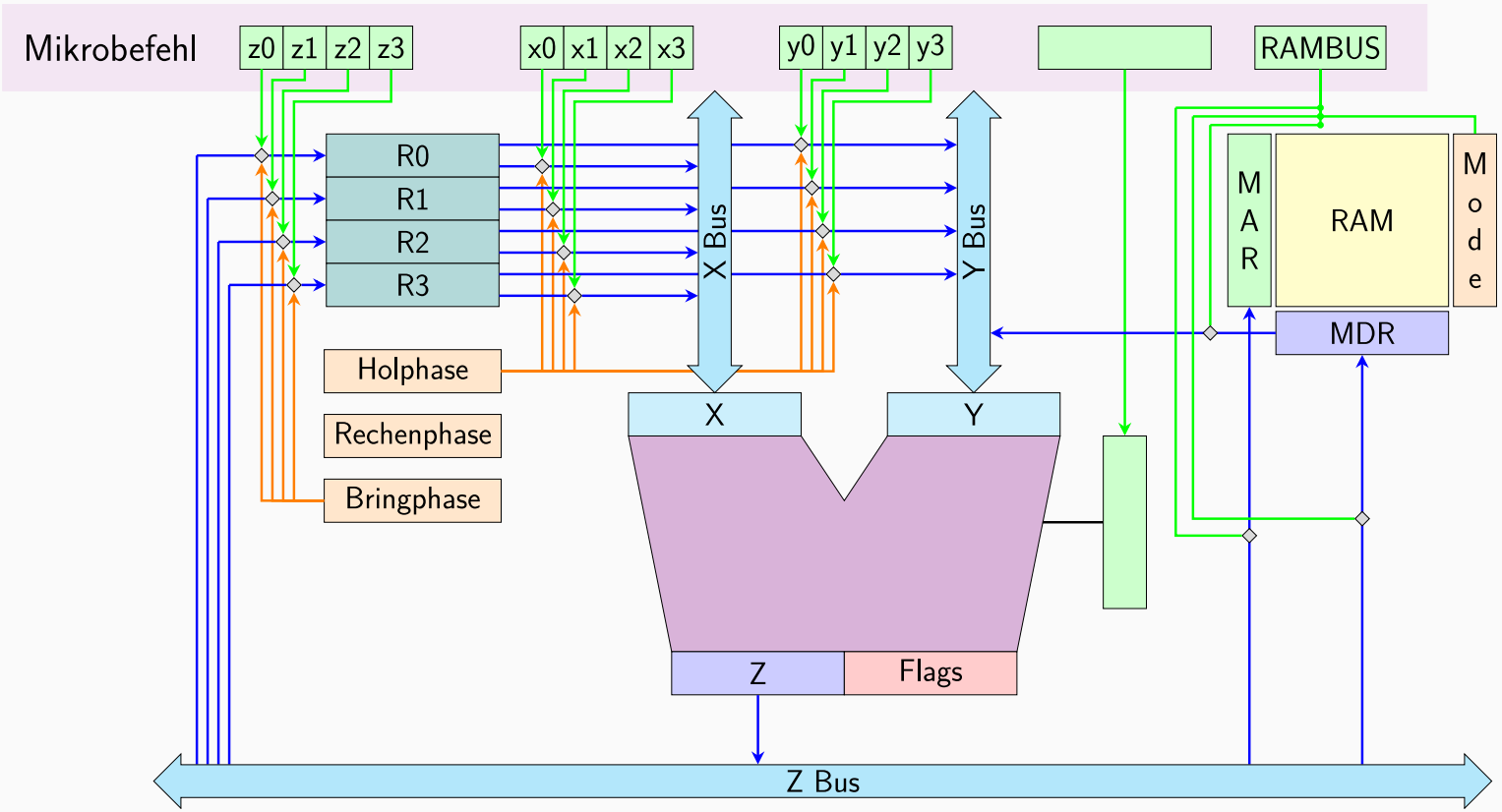
CPU und Speicher: MDR→Y (3/5)

- neue Datenleitung zum lesen
  - MDR → Y (Holphase)



CPU und Speicher: Steuerung (4/5)

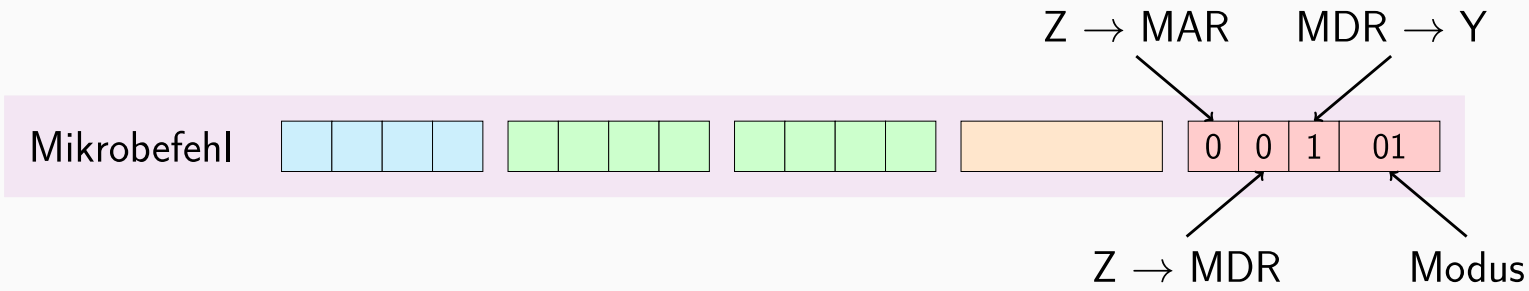
- Mikrobefehl erweitern zur Steuerung von Schalter und Modus:
  - Lesen (vor der Holphase)
  - Schreiben (nach der Bringphase)
  - Warten



CPU und Speicher: Zusammenfassung Schalter (5/5)

- Folgende RAMBUS Schalter stehen im Mikrobefehl zur Verfügung
  - Z $\rightarrow$ MAR: ALU Ergebnis ins Adressregister (Bringphase)
  - Z $\rightarrow$ MDR: ALU Ergebnis ins Datenregister (Bringphase)
  - MDR $\rightarrow$ Y: Datenregister zur ALU (Holphase)
  - Modus (2 bit):

Bits	Bedeutung
00	Warten
01	Lesen
10	Schreiben
11	Warten



Ablauf Speicherzugriff (Lesen)

- Zugriff auf Wert im RAM an Adresse *x* (Notation `[x]`)
- Beispiel [2] → R0 (Wert im RAM an der Adresse 2 in das Register 0 lesen)
  1. Takt
     - Holphase: - (nichts zu tun)
     - Rechenphase: `Z := 2` (Adresse in ALU generieren)
     - Bringphase: Z → MAR (Adresse über Z-Bus in MAR übertragen)
  2. Takt
     - vor Holphase: Mode: lesen (Wert wird von RAM in MDR übertragen, [MAR]→MDR)
     - Holphase: MDR → Y (vom Datenregister zur ALU)
     - Rechenphase: `Z := Y` (ALU schleust Wert durch)
     - Bringphase: Z → R0 (Wert in Register 0 bringen)

Ablauf Speicherzugriff (Schreiben)

- Zugriff auf Wert im RAM an Adresse *x* (Notation `[x]`)
- Beispiel R1 $\rightarrow$ [6] (den Wert im Register 1 in den RAM an die Adresse 6 schreiben)
  1. Takt
     - Holphase: - (nichts zu tun)
     - Rechenphase: `Z := 6` (Adresse in ALU generieren)
     - Bringphase: Z $\rightarrow$ MAR (Adresse über Z-Bus in MAR übertragen)
  2. Takt
     - Holphase: R1 $\rightarrow$ X (vom Register zur ALU)
     - Rechenphase: `Z := X` (ALU schleust Wert durch)
     - Bringphase: Z $\rightarrow$ MDR (Wert in Datenregister bringen)
     - nach Bringphase: Mode: schreiben (Wert wird von MDR in RAM übertragen, MDR $\rightarrow$ [MAR])

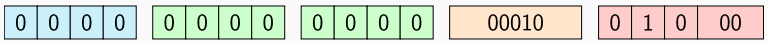
Beispiel Speicherzugriff (1/3)

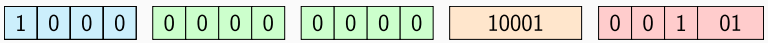
- Schritte um die Werte im RAM an den Adressen 2 und 4 zu addieren und das Resultat an die Adresse 6 im RAM zu schreiben ([2] + [4] $\rightarrow$ [6])
  - [2] $\rightarrow$ R0
    1. Takt: 2 $\rightarrow$ MAR
    2. Takt: [MAR] $\rightarrow$ R0
  - [4] + R0 $\rightarrow$ R0
    1. Takt: 4 $\rightarrow$ MAR
    2. Takt: [MAR] + R0 $\rightarrow$ R0
  - R0 $\rightarrow$ [6]
    1. Takt: 6 $\rightarrow$ MAR
    2. Takt: R0 $\rightarrow$ [MAR]

Beispiel Speicherzugriff (2/3)

[2] → R0

1. Takt: 2 → MAR
   - R: Z := 2
   - B: Z → MAR
2. Takt: [MAR] → R0
   - lesen ([MAR] → MDR)
   - H: MDR → Y
   - R: Z := Y
   - B: Z → R0





[4] + R0 → R0

1. Takt: 4 → MAR
   - R: Z := 4
   - B: Z → MAR
2. Takt: [MAR] + R0 → R0
   - lesen ([MAR] → MDR)
   - H: MDR → Y, R0 → X
   - R: Z := X + Y
   - B: Z → R0

Beispiel Speicherzugriff (3/3)

R0 $\rightarrow$ [6]

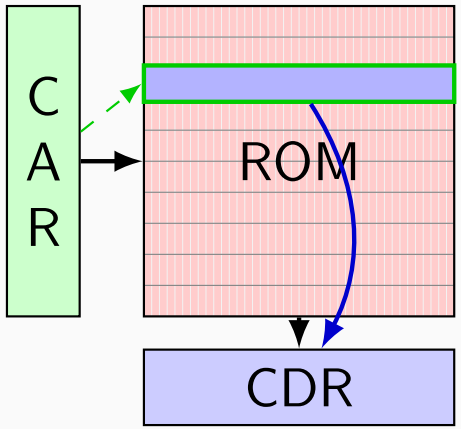
1. Takt: 6 $\rightarrow$ MAR
   - H: -
   - R: Z := 6
   - B: Z $\rightarrow$ MAR
2. Takt: R0 $\rightarrow$ [MAR]
   - H: R0 $\rightarrow$ X
   - R: Z := X
   - B: Z $\rightarrow$ MDR
   - schreiben (MDR $\rightarrow$ [MAR])

`demos/CPU.circ` `ALU + Reg + RAM`

Speicher für Mikrobefehle

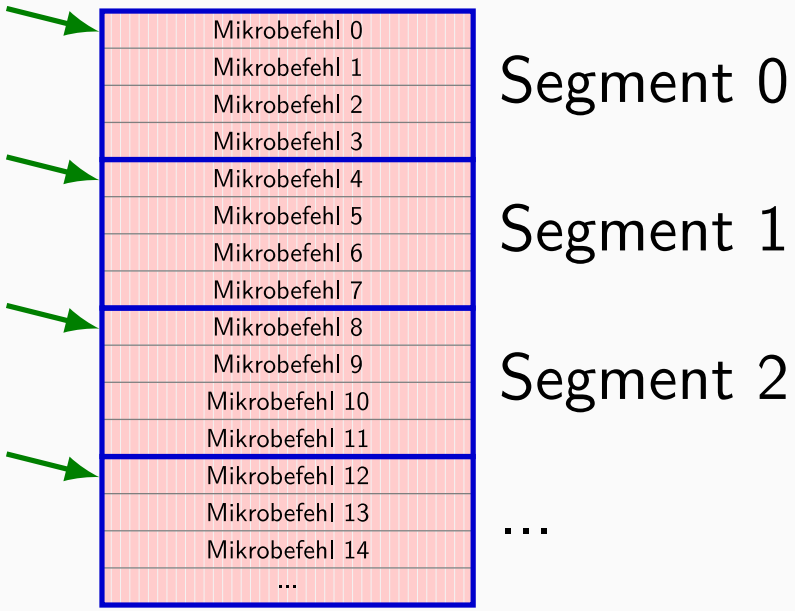
- Mikrobefehle bestehen aus einzelnen Schaltern → Bitmuster
  - können ebenfalls gespeichert werden
- eigener Speicher für “Liste” von Mikrobefehlen
  - vom Benutzer nicht veränderbar: Read-only Memory (ROM)

- Schnittstelle ähnlich zu RAM, aber Modus ist immer lesen
  - Code Address Register (CAR)
  - Code Data Register (CDR) enthält den aktuellen Mikrobefehl



Segmentierung des ROMs

- vier Mikrobefehle im ROM werden zu einem Segment zusammengefasst
- Anfang der Segmente dienen als Einsprungpunkte



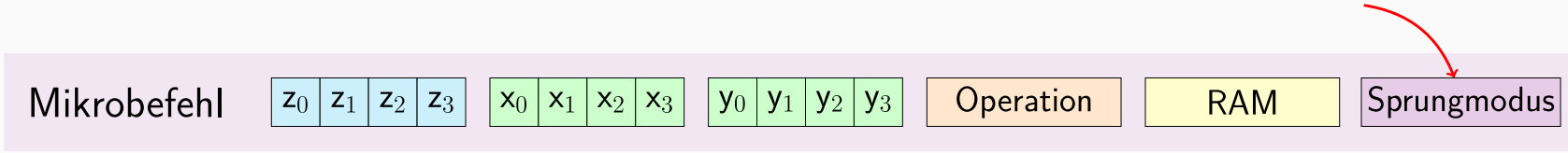
Auswahl des nächsten Mikrobefehls

- folgende Sprungmodi sind möglich
  - nächster Befehl liegt direkt hinter aktuellem Befehl
    - `CAR := CAR + 1` oder `CAR++`
  - nächster Befehl liegt an fixer Stelle im ROM
    - Segmentnummer steht im Mikrobefehl als Code Next (`CN`)
    - `CAR := 4 CN, CN = ...`
  - nächster Befehl liegt an variabler Stelle im ROM
    - Segmentnummer wird aus Code OPeration Register (`COP`) geholt
    - `CAR := 4 COP`
  - nächster Befehl hängt von Berechnungsergebnis ab (Verzweigung)
    - Bedingung steht in einer Bit-*Maske* im Mikrobefehl
    - `CAR := 4 COP` falls Bedingung erfüllt (sonst `CAR++`)

Auswahl des nächsten Mikrobefehls: Adressrechner

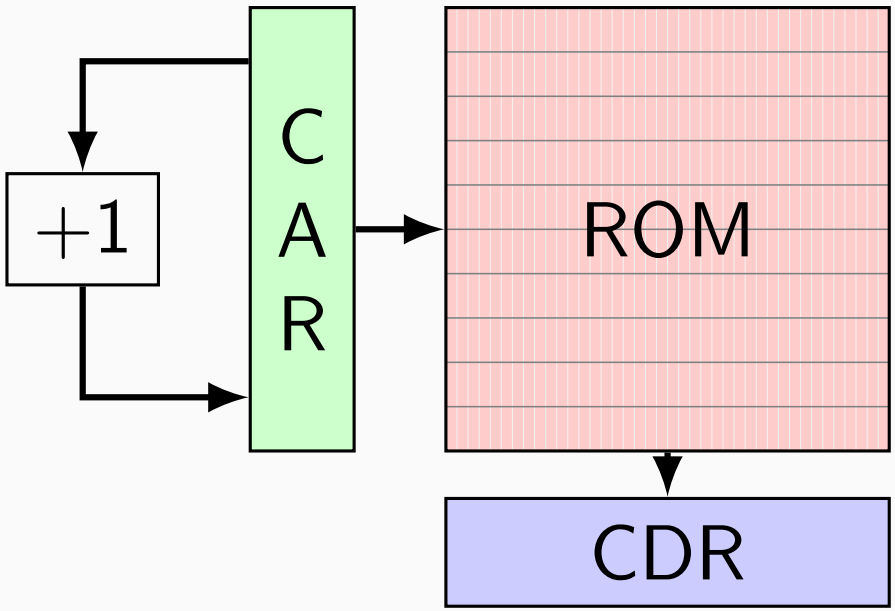
- Adresse des nächsten Mikrobefehls (next CAR, **NCAR**) wird durch Adressrechner berechnet
- Die Art des Sprungs wird durch zusätzliche Schalter im Mikrobefehl ausgewählt
- folgende Sprungmodi sind möglich

Sprungmodus	Berechnung von next CAR
00	`CAR := CAR + 1` oder `CAR++`
01	`CAR := 4CN`
10	`CAR := 4COP`
11	`CAR := 4COP if cond<>0 else CAR++`



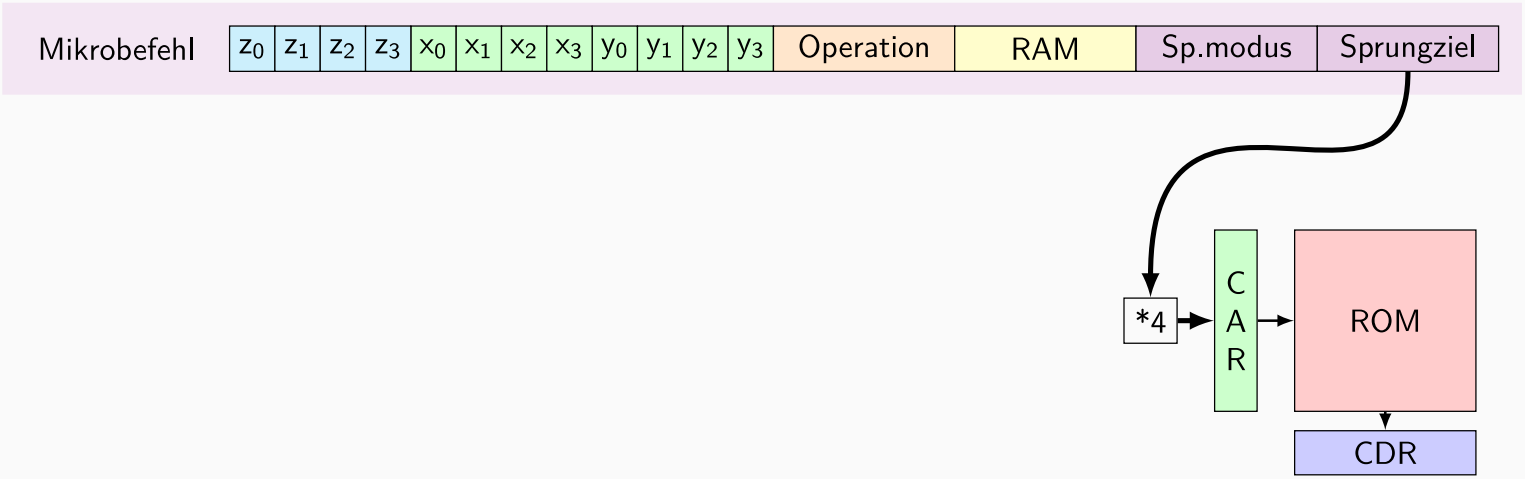
Sprungmodus: darauf folgender Befehl (kein Sprung)

- Die Adresse des nächsten Befehls (next CAR) kann einfach durch Inkrementieren des aktuellen Wertes von CAR berechnet werden
- Notation: `NCAR := CAR+1` oder `CAR++`



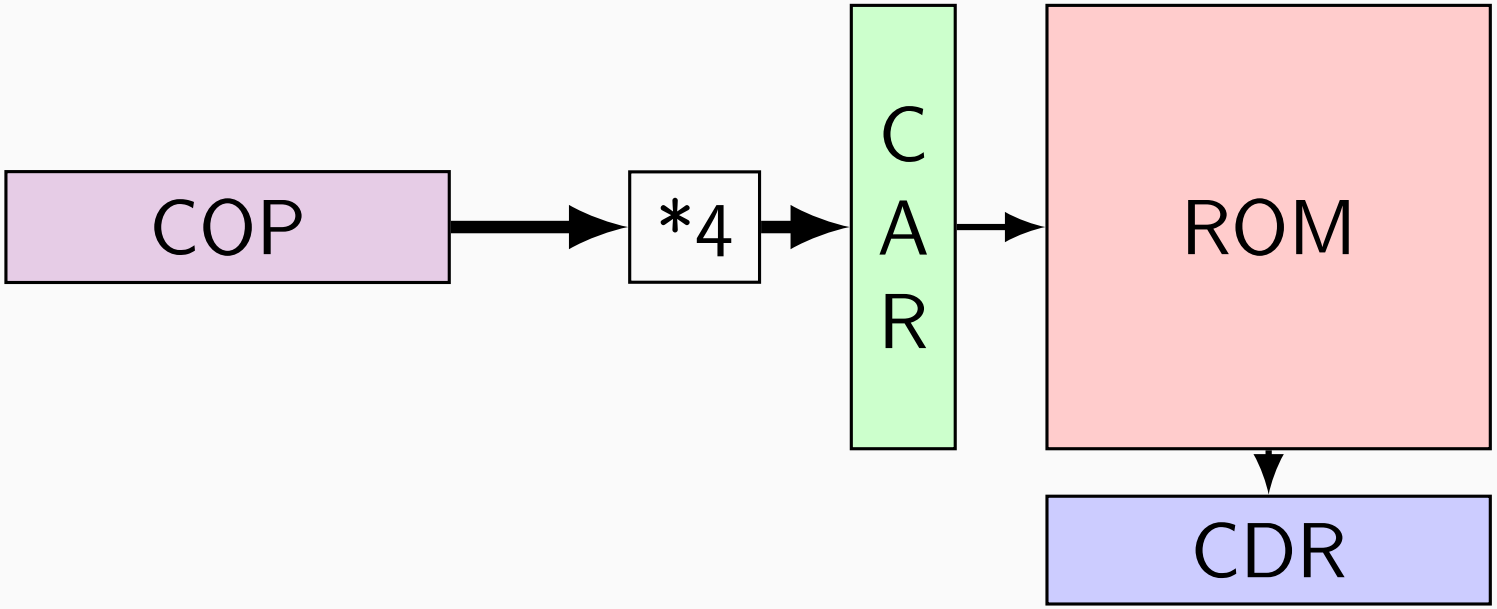
Sprungmodus: fixer Sprung

- Die Adresse des nächsten Befehls wird durch den Mikrobefehl vorgegeben, dazu wird zusätzlich zum Sprungmodus die Segmentnummer (Code Next, **CN**) in den Mikrobefehl eingebaut
- Notation: `NCAR := 4 · CN`, `CN=...`



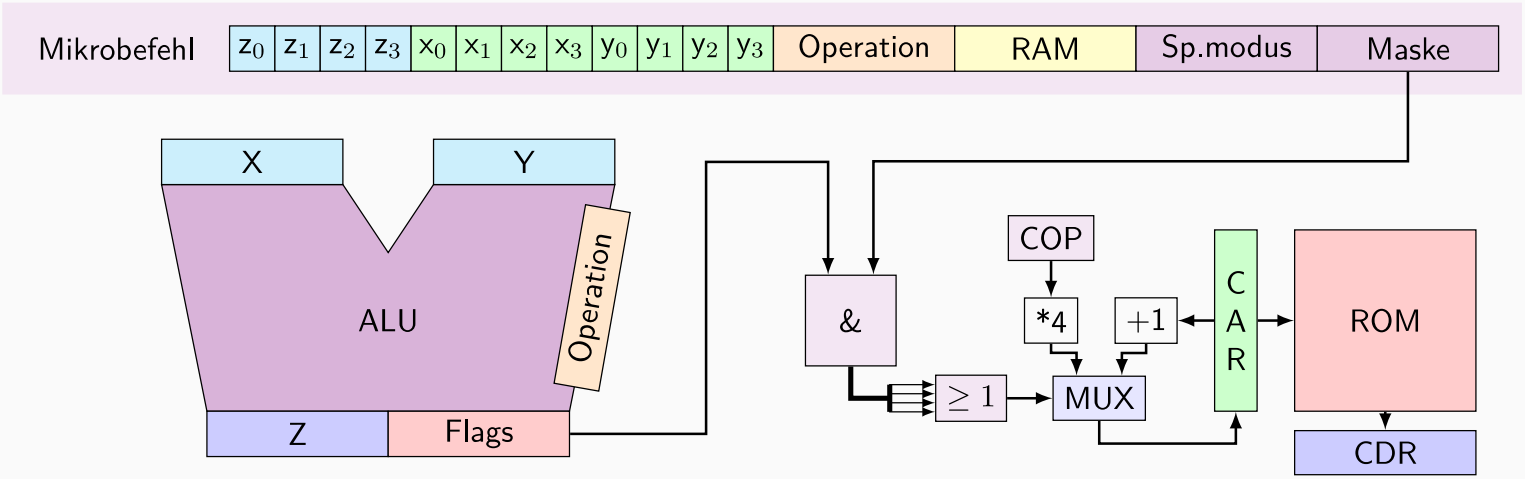
Sprungmodus: berechneter Sprung

- Ziel des Sprungs wird aus (neuem) Register (Code Operation Register, COP) gelesen
- dort wird Segmentnummer angegeben (COP*4 ergibt Mikrobefehlsadresse)
- kann bei erneuter Ausführung (desselben Befehls) woanders hin springen
- Notation: $\mathrm{NCAR} := 4 \cdot \mathrm{COP}$



Sprungmodus: bedingter Sprung (1/2)

- Sprungziel steht im **COP** Register, Bedingung wird im Mikrobefehl als Maske für ALU-Flags angegeben
- Sprung nur falls mind. eines der ALU-Flags in der Maske vorkommt
  - andernfalls wird nächster Befehl ausgeführt (**CAR++**)
- Notation: `NCAR := 4 · COP IF cond<>0, MASKE=...`

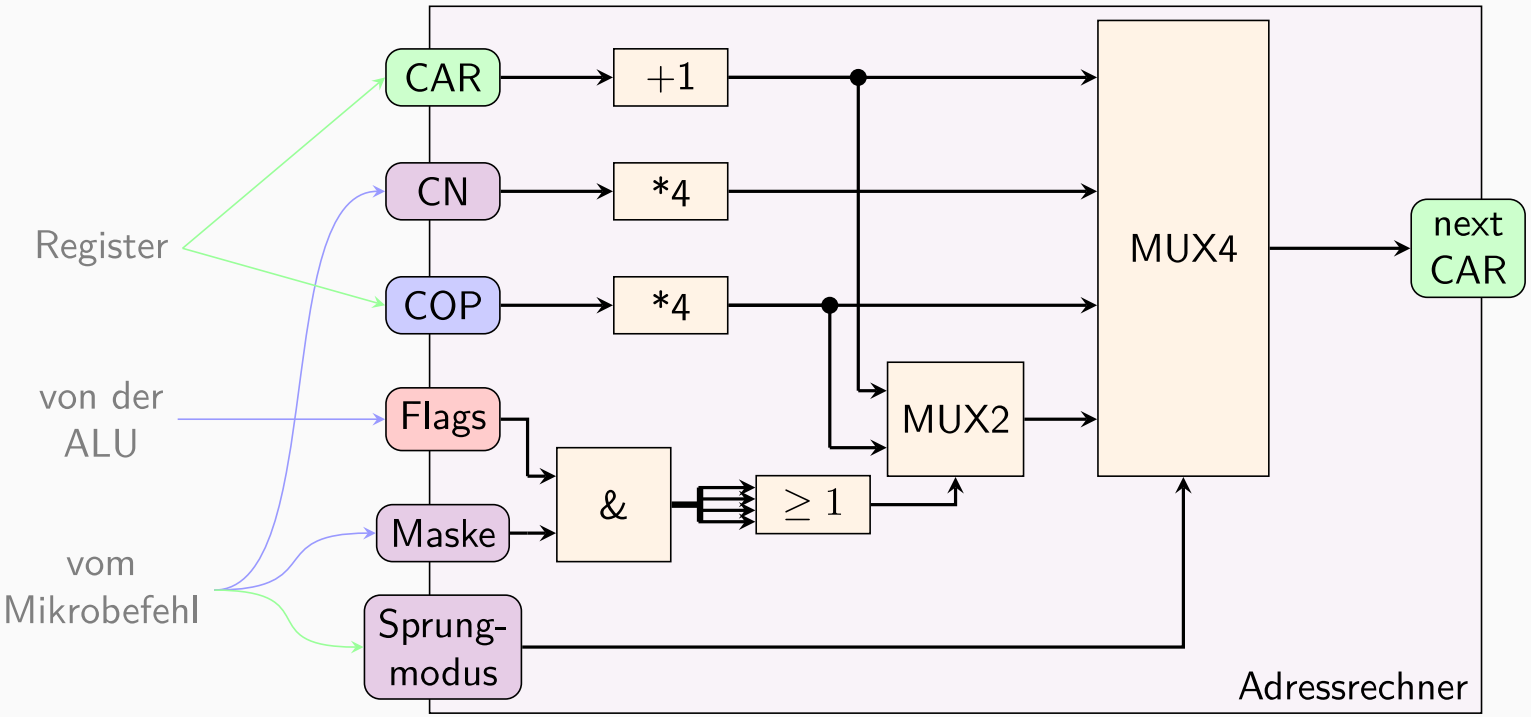


Sprungmodus: bedingter Sprung (2/2)

- Annahme Flags sind: Zero Flag, Positive Flag, Negative Flag, Overflow Flag
- Beispiele
  1. Beispiel für bedingten Sprung
     - Maske=1000 (Zero, Positive, Negative, Overflow)
     - Annahme X=0, Z:=X ergibt Flags=1000 (Z P N Of)
     - Sprung wird zu 4·COP ausgeführt
  2. Beispiel für bedingten Sprung
     - Maske=1000 (Zero, Positive, Negative, Overflow)
     - Annahme X=1, Z:=X ergibt Flags=0100 (Z P N Of)
     - Sprung wird nicht ausgeführt, sondern CAR++

Implementierung Adressrechner

- Der Adressrechner wählt je nach Sprungmodus eine Berechnungsmethode für den nächsten Wert von CAR



`demos/CPU.circ` Adressrechner

Maske und CN braucht man nie gleichzeitig,
daher kann man sie an der selben Stelle im ROM speichern
Je nach Modus wird die Zahl dann anders verwendet

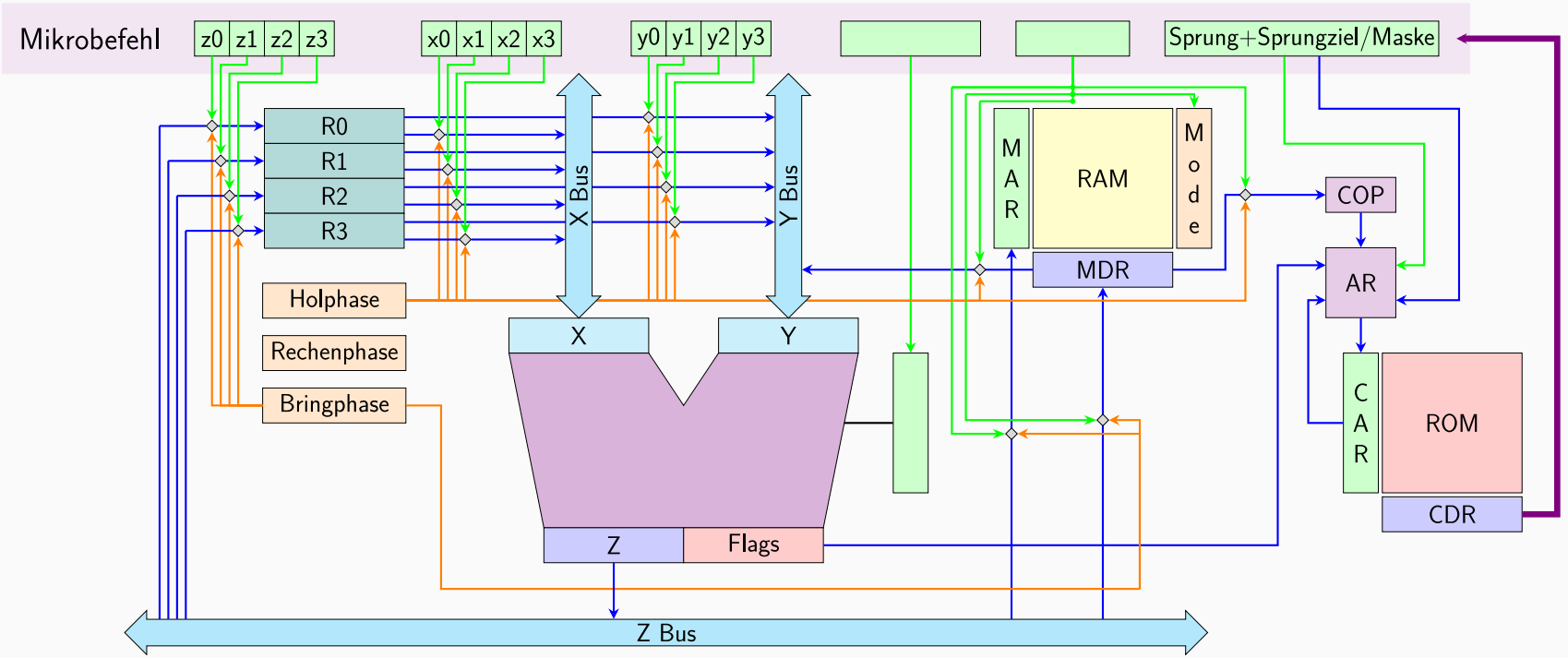
Überlagerung von Code Next und Maske

- Um Platz zu sparen, werden Code Next `CN` und `Maske` mit denselben Bits im Mikrobefehl codiert

Z-Bus	X-Bus	Y-Bus	ALU-Code	Sprungmodus
				00 - - - -
				01 Code Next
				10 - - - -
				11 Maske

Gesamte CPU Architektur

- Adressrechner (**AR**) berechnet nächste Mikrocode Adresse
- Vor jedem Takt wird der Mikrobefehl aus dem Code Data Register (**CDR**) geladen



Überblick Phasen eines Taktes

1. Phase Mikrocode Laden
   - Mikrobefehl wird aus ROM geladen
2. Phase Lesen
   - falls RAM Modus lesen: Wert aus RAM in MDR lesen
3. Phase Holen
   - Werte aus Registern zur ALU (X, Y) holen
4. Phase Rechnen
   - ALU rechnet und schreibt Ergebnis in Z und Flags
   - Adressrechner bestimmt nächsten Mikrobefehl (next CAR)
5. Phase Bringen
   - Rechenergebnis aus Z zu Registern bringen
6. Phase Schreiben
   - falls RAM Modus schreiben: Wert von MDR in RAM schreiben

Mikrocode Laden

- Vor der Ausführung eines Befehls muss dieser zuerst aus dem ROM geladen werden.
- An der im vorherigen Takt eingestellte Mikrocode-Adresse im Code Address Register (CAR) wird der Mikrocode ins Code Data Register (CDR) geladen.
- Dieser neue Mikrobefehl liegt während des gesamten Taktes (allen folgenden Phasen) an den Steuerleitungen an.

Lesephase

- falls im aktuellen Mikrobefehl der RAM Modus auf `lesen` gestellt ist, wird das Lesen durchgeführt, falls Modus `schreiben` oder `warten` wird in dieser Phase nichts gemacht
- Die passende Adresse muss bereits in einem vorherigen Takt in das Adressregister (MAR) gebracht worden sein
- Der Wert im Hauptspeicher (RAM) an der ausgewählten Adresse wird dann ins Datenregister (MDR) gelesen
- Notation:
  - `lesen` oder `[MAR]` $\rightarrow$ `MDR` am Anfang des Mikrobefehls bezeichnet Modus Lesen
  - `warten` oder keine Angabe bezeichnen Modus Warten

Holphase

- Jedes Register der ALU (X und Y) kann einen oder auch keinen Wert von anderen Registern erhalten
- Als Quelle stehen folgende Register zur Verfügung
  - R0, z.B. `R0` $\rightarrow$ `X`
  - R1, z.B. `R1` $\rightarrow$ `Y`
  - R2
  - R3
  - MDR, z.B. `MDR` $\rightarrow$ `Y`
- Außerdem kann in dieser Phase der Wert im Datenregister (`MDR`) in das Code Operation Register (`COP`) als Sprungziel gebracht werden
  - Notation: `MDR` $\rightarrow$ `COP`
- In anderen Architekturen können auch noch weitere Registertransfers zur Verfügung stehen, z.B. `X` $\leftrightarrow$ `Y` vertauscht X und Y

Rechenphase

- Die Angelegten Werte in X und Y werden in der ALU mit der im Mikrobefehl angegebenen Operation verknüpft und das Ergebnis wird ins Z-Register der ALU gebracht, gleichzeitig werden auch die Flags im Flags-Register aktualisiert
- Zur Auswahl stehen folgende Klassen von Operationen
  - Arithmetik, z.B. `Z:=X+Y`, `Z:=-X` oder `Z:=X SHL Y` (shift left)
  - Logik, z.B. `Z:=X AND Y`, `Z:=X XOR Y` oder `Z:=NOT X`
  - Durchschleusen, z.B. `Z:=X` oder `Z:=Y`
  - Konstanten generieren, z.B. `Z:=0`, `Z:=1`, oder `Z:=8`
- Der Adressrechner wertet den Sprungmodus und die ALU Flags aus und bestimmt die Adresse des nächsten Mikrobefehls (z.B. CAR+1, 4·COP, oder 4·CN)

Bringphase

- Das in der Rechenphase generierte Ergebnis im Z-Register kann zu anderen Register geleitet werden
  - R0 bis R3, z.B. `Z` $\rightarrow$ `R0`
  - `Z` $\rightarrow$ `MAR`
  - `Z` $\rightarrow$ `MDR`
- Es ist möglich auch auf mehrere Register gleichzeitig zu leiten
  - z.B. `Z` $\rightarrow$ `R0`, `R1`, `MAR`

Schreibphase

- falls im aktuellen Mikrobefehl der RAM Modus auf `schreiben` gestellt ist, wird das Schreiben durchgeführt, falls Modus `lesen` oder `warten` wird in dieser Phase nichts gemacht
- Die passende Adresse mus bereits in einem vorherigen Takt in das Adressregister (MAR) gebracht worden sein
- Der Wert im Datenregister (MDR) wird in den Hauptspeicher (RAM) an die ausgewählt Adresse geschrieben
- Notation:
  - `schreiben` oder `MDR` $\rightarrow$ `[MAR]` am Ende des Mikrobefehls bezeichnet Modus Schreiben
  - `warten` oder keine Angabe bezeichnen Modus warten

Mikrocode “Fragebogen”

Holphase (H:)

X-Register

○ **unverändert** (default) ○ R0→X ○ R1→X ○ R2→X ○ R3→X

Y-Register

○ **unverändert** (default) ○ R0→Y ○ R1→Y ○ R2→Y ○ R3→Y ○ MDR → Y

□ MDR → COP

Rechenphase (R:)

Keine Operation: ○ **Z:=Z** (default)

Konstante generieren: ○ Z:=_____ (Konstante)

Wert durchschleusen: ○ Z:=X ○ Z:=Y

Rechenoperation

○ Z:=X+Y	○ Z:=INC X	○ Z:=INC Y	○ Z:=X*Y
○ Z:=X-Y	○ Z:=DEC X	○ Z:=DEC Y	○ Z:=X/Y
○ Z:=X<<Y	○ Z:=X>>Y	○ Z:=-X	○ Z:=X AND Y
○ Z:=X OR Y	○ Z:= X XOR Y	○ Z:=NOT X	○ Z:=NOT Y

Adressrechner (AR:)

○ **CAR++** (default)

○ CAR := 4CN

○ CAR := 4COP

○ CAR := 4COP if cond<>0

CN=___ (Segment Nr)

MASKE=_____ (Zero Pos. Neg. Overflow)

Bringphase (B:)

□ Z→R0 □ Z→R1 □ Z→R2 □ Z→R3

□ Z→MAR □ Z→MDR

RAM (RAM:)

○ **warten** (default)

○ lesen ([MAR] → MDR)

○ schreiben (MDR → [MAR])