Rechnerarchitektur

Einführung in die Informatik & Rechnerarchitektur (EIR1/EIF1)

Erik Pitzer

SE & MBI – FH Hagenberg – WS 2025/26

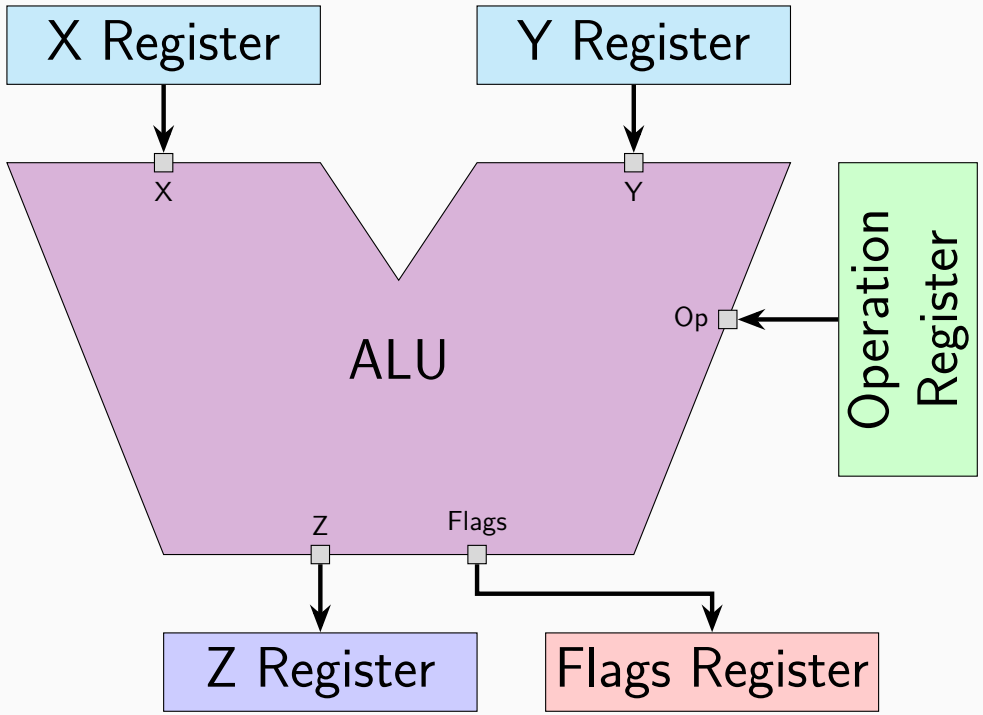
Rechenwerk

Rechenwerk

- Rechenwerk soll beliebige arithmetische und logische Operationen ausführen können, wie z.B.
  - 3+5, 8·7, 7/3
  - 01001_2 AND 11010_2, 1101_2 XOR 0101_2
  - Konstanten generieren und Werte “durchschleusen”
- Komplexität der Schaltung und die Anzahl der Verbindungen klein halten
- Operation wird als Parameter übergeben: f(x, y, operation)
  - `f(3, 5, +)` = 3 + 5
  - `f(8, 7, *)` = 8 · 7
  - `f(01001, 11010, AND)` = 01001_2 AND 11010_2
  - `f(01001, 11010, 1)` = 1

Arithmetisch-Logische Einheit (Arithmetic Logic Unit, ALU)

- X, Y enthalten Operanden, Z erhält das Ergebnis
- Flags beschreiben das Ergebnis, z.B. größer/kleiner/gleich Null, Overflow
- “Wortbreite” ist die Anzahl der Bits die eine ALU pro Operation verarbeiten kann



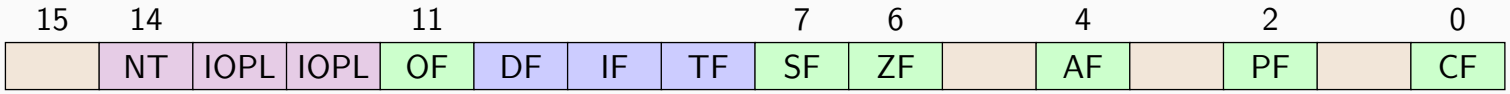
FLAGS Register (1/2)

Enthält Informationen über das letzte Ergebnis (Status Flags) oder Optionen für die Durchführung weitere Befehle (Steuer Flags)

- Status Flags (werden von der ALU gesetzt)
  - Information über letztes Ergebnis (Inhalt des Z-Registers)
    - Zero Flag (ZF): Ergebnis war Null
    - Sign Flag (SF): Ergebnis war negativ
    - Parity Flag (PF): Ergebnis hat gerade Parität
  - Information über letzte Operation
    - Overflow Flag (OF): Operation ergab einen Überlauf
    - Carry Flag (CF): Operation ergab einen Übertrag (Carry) aus dem höchsten Bit
    - Aux Carry (AC): Operation ergab einen Übertrag aus dem vierten Bit

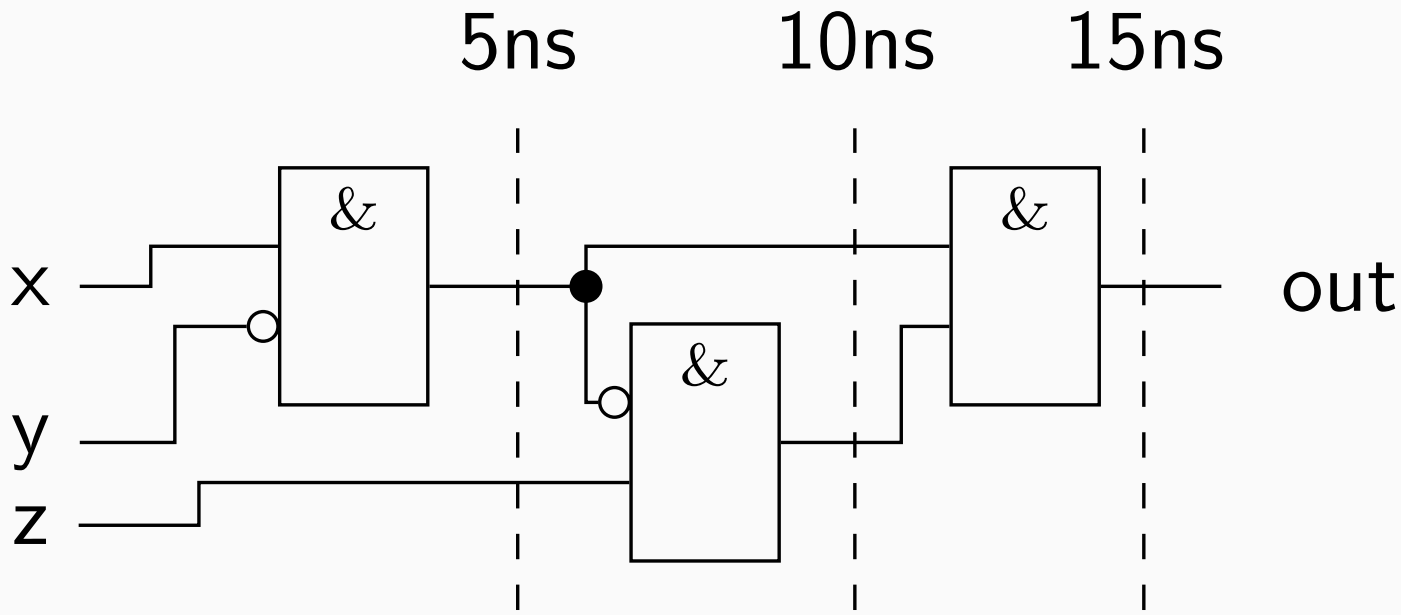
FLAGS Register (2/2)

- Steuer Flags (werden vom Programmierer gesetzt)
  - Trap Flag (TF): Programm nach dem nächsten Befehl anhalten, für Programmentwicklung und Debugging relevant
  - Interrupt Enable Flag (IF): Hardware darf Programmausführung unterbrechen, z.B. Daten von Gerät einlesen
  - Direction Flag (DF): Zeichenkettenverarbeitung rückwärts oder vorwärts
- weitere Flags, z.B. I/O privilege level (IOPL), Nested Task oder reservierte Bits



Gatterverzögerungen in der ALU

- Einzelne Gatter benötigen kurze Umschaltzeit, Kaskade von Gatter benötigt entsprechend länger
- Stabilisierung des Ergebnis muss abgewartet werden

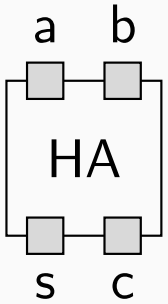


demos/gate_delay.dig

Addition

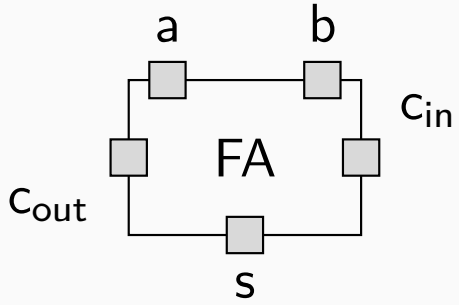
- Halbaddierer (Half Adder)

a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



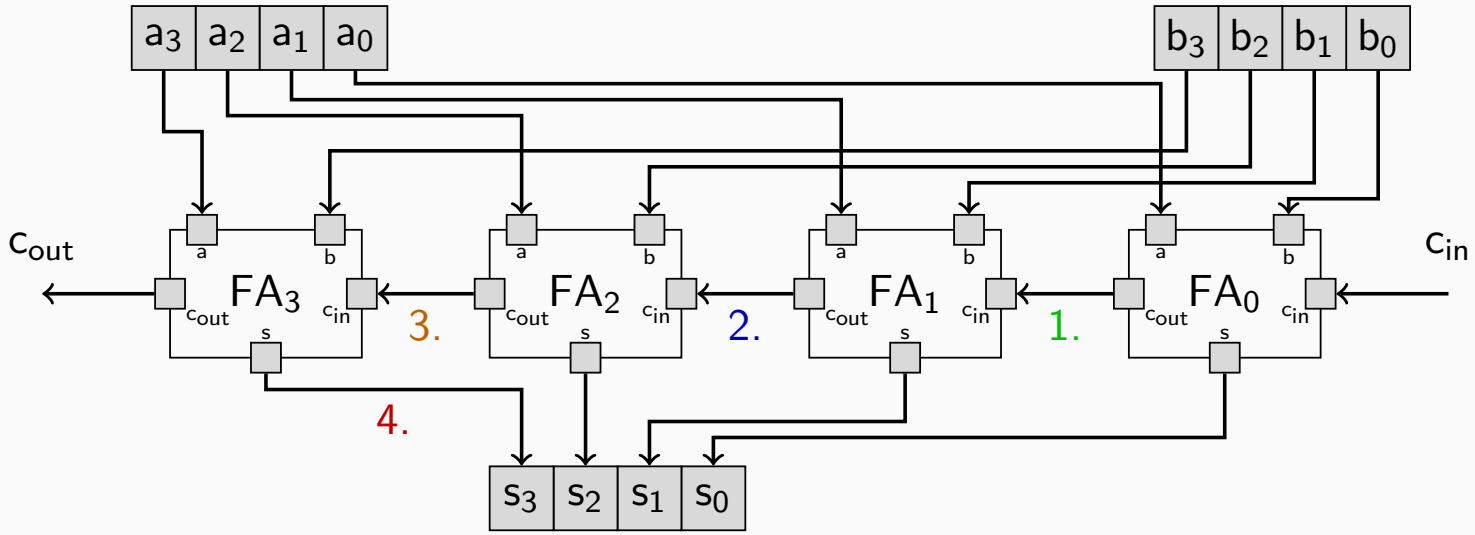
- Volladdierer (Full Adder)

a	b	carry_{in}	sum	$\text{carry}_{\text{out}}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Addition mit Ripple Carry Adder

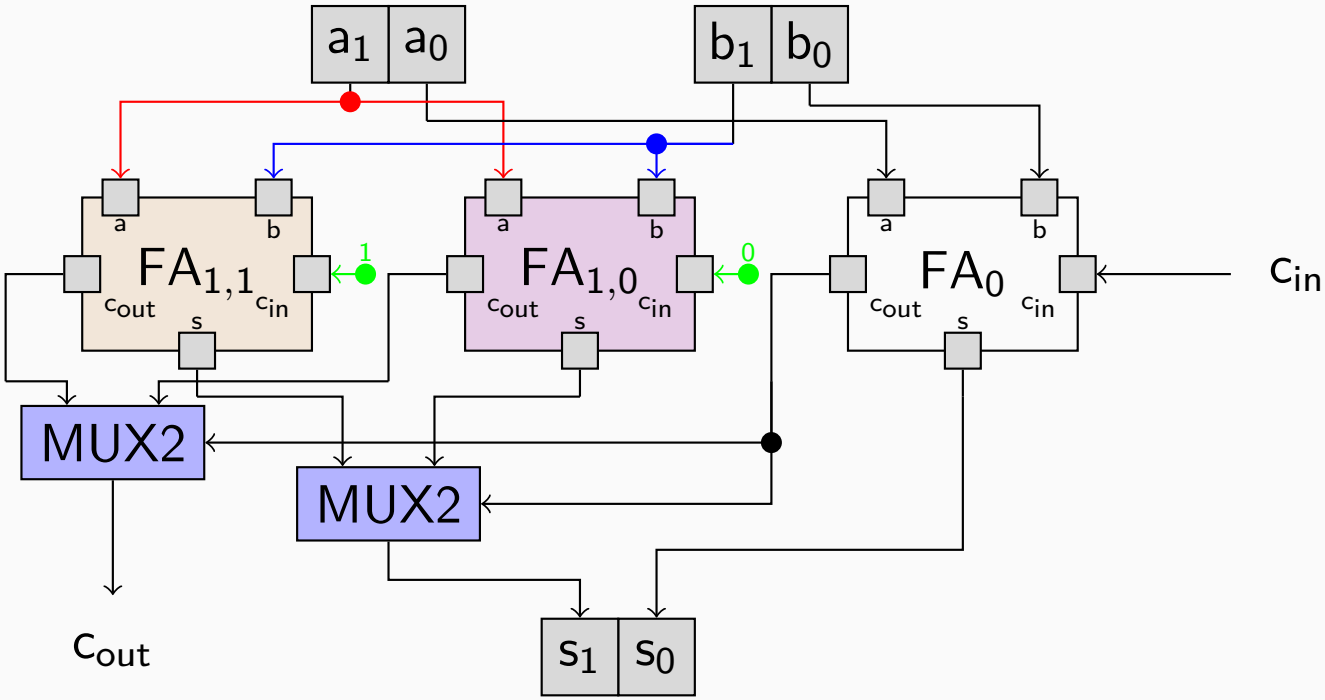
- Einfache Konstruktion, aber langes warten auf Weiterleitung aller Überträge, z.B. 32-Bit-Addierer 32·FA Verzögerung



demos/add4.dig

Addition mit Carry Select Adder (1/2)

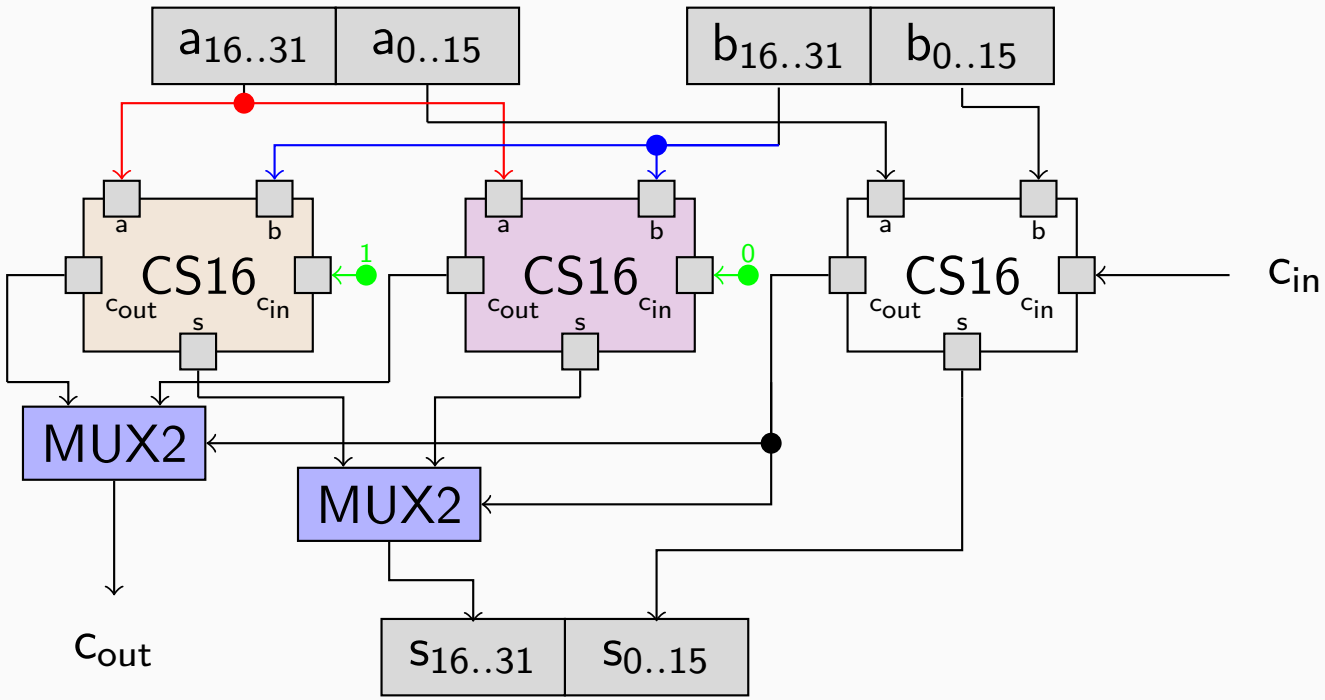
- Alle Additionen parallel berechnen, einmal mit, einmal ohne Carry
- vorheriges Carry bestimmt welches Ergebnis durch Multiplexer gewählt wird



demos/alu.circ CS4

Addition mit Carry Select Adder (2/2)

- Gleiches Prinzip kann für größere Bit-Breiten verwendet werden ($CS_{n+1} = 3 \cdot CS_n + 2 \cdot MUX$, z.B. 32-Bit: $81 \cdot FA_2 + 80 \cdot MUX$)
- Verdopplung der Bit-Breite nur mehr zusätzliche Verzögerung um 1·MUX, z.B. bei 32 Bit nur 2·FA + 4·MUX Verzögerung

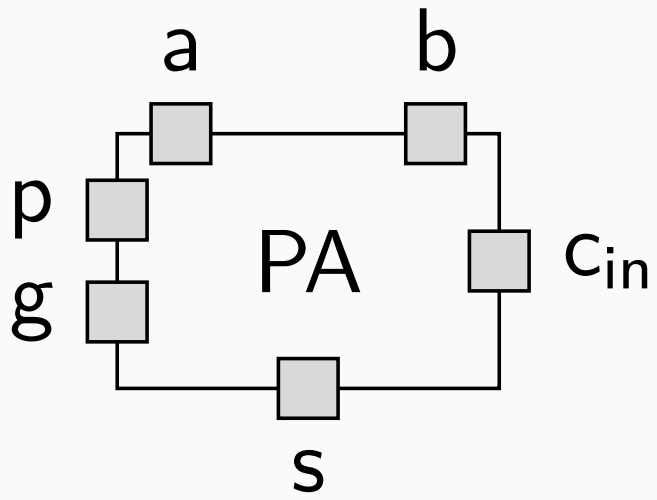


demos/alu.circ CS16

Addition mit Carry-Look-Ahead Adder (1/2)

- Carry wird für alle Stellen gleichzeitig berechnet
- Volladdierer wird zu “partial adder” (PA), gibt Information über potentiellen Übertrag (ohne Berücksichtigung von Carry In)
  - Carry wird immer generiert (**g**)
  - Carry wird propagiert (**p**), falls Carry hereinkommt

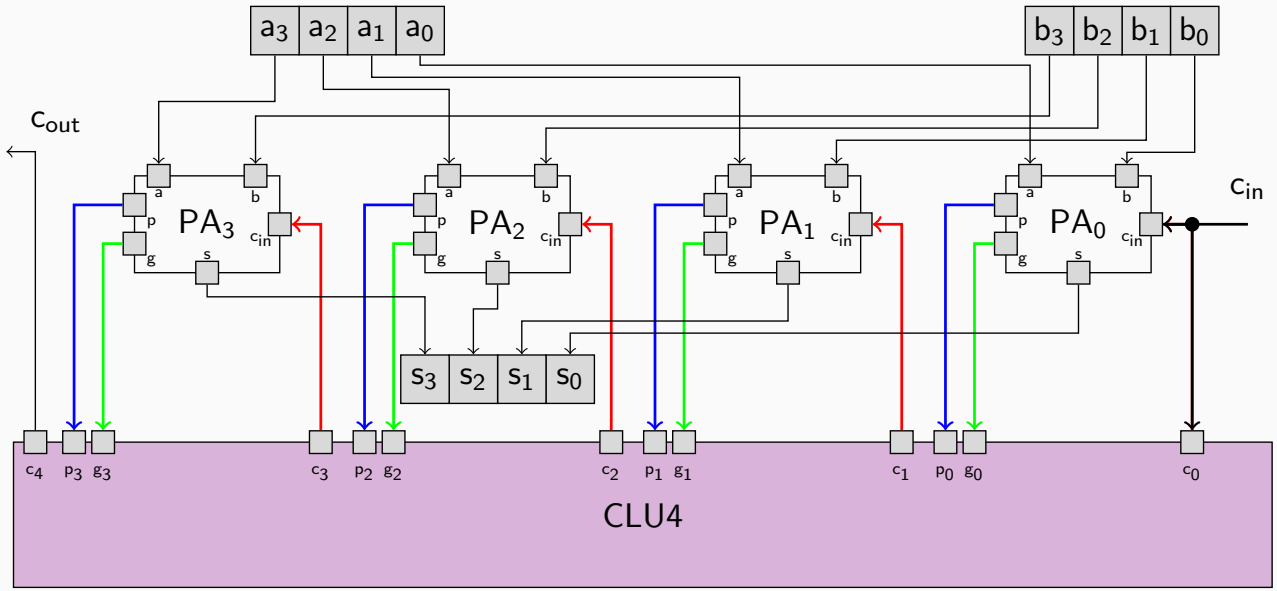
a	b	c_{in}	s	g	p
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	1	1	1



demos/alu.circ PA

Addition mit Carry-Look-Ahead Adder (2/2)

- Eigene Carry-Lookahead-Unit (CLU) berechnet Übertrag für alle Stellen gleichzeitig, z.B.
  - $c_1 = c_{in} \cdot p_0 + g_0$
  - $c_2 = c_{in} \cdot p_0 \cdot p_1 + g_0 \cdot p_1 + g_1$
- Für jede Bit-Breite immer nur drei Schritte (PA→CLU→PA)



demos/alu.circ LCU4, CLA4

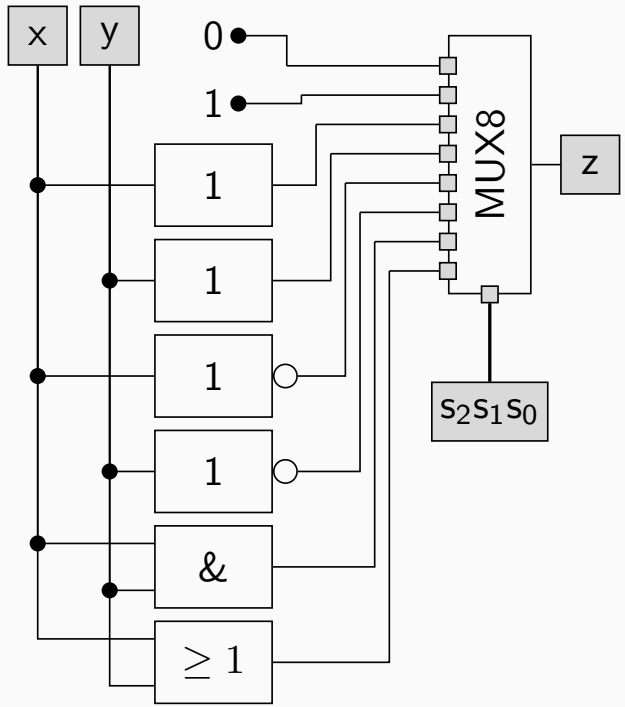
Additionswerke Vergleich

- mit zusätzlichen Gattern lassen sich schnellere Schaltungen bauen
- z.B. 32-Bit Addierer mit verschiedenen Implementierungen

Implementierung	Verzögerung	Bauelemente
Ripple-Carry	32	≈ 32
Carry-Select	6	≈ 160
Carry-Lookahead	3	≈ 560

ALU Aufbau: Logic Unit (LU) für ein Bit

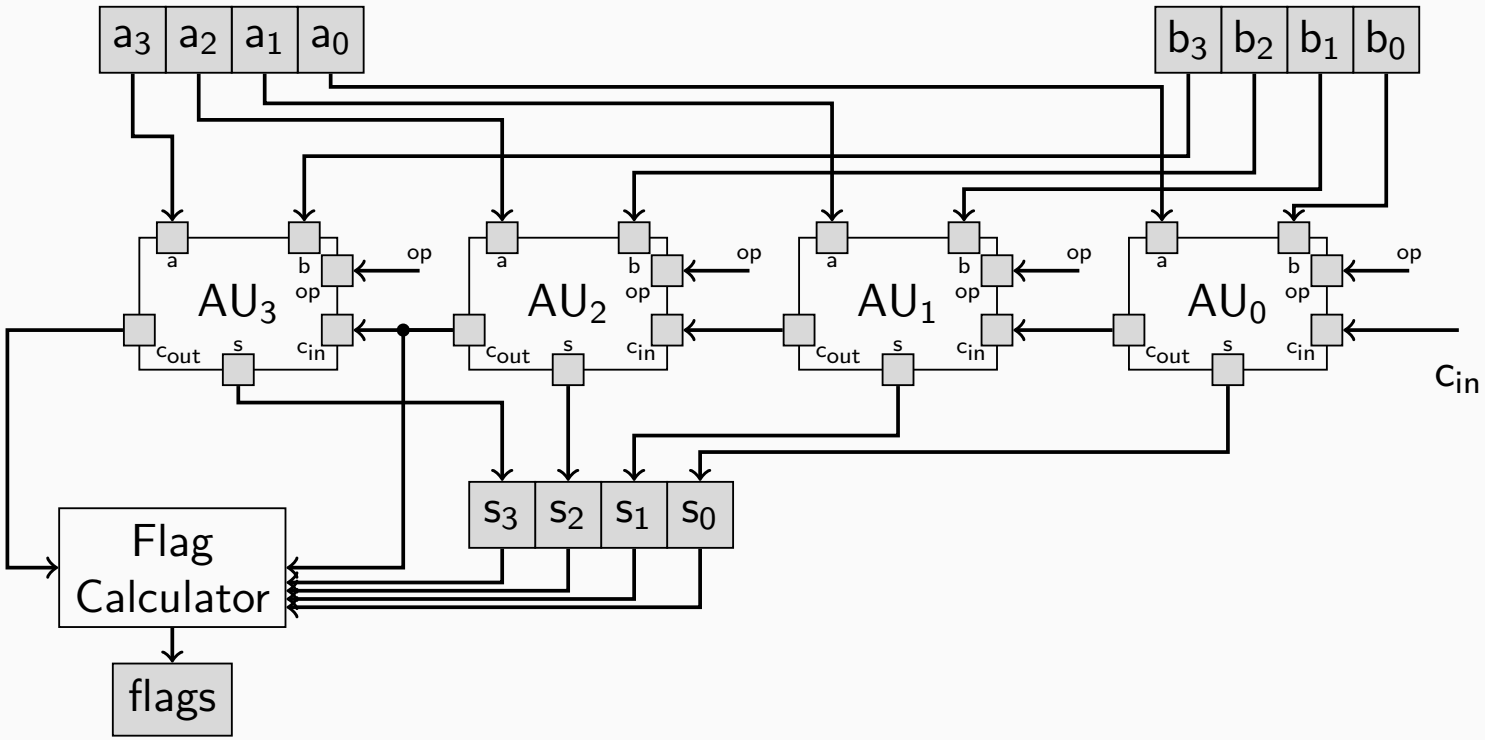
- Teil der ALU für logische Operationen
- Alle Operationen parallel berechnen
- Auswahl über Multiplexer
- Logic Unit hat z.B. folgende Funktionen
  - Erzeugen von Konstanten (0, 1)
  - Durchschleusen von x oder y
  - Negieren von x oder y
  - Konjunktion ($\&$) oder Disjunktion (≥ 1) von x und y
- Kann für beliebige Bit-Breite parallel geschaltet werden



`demos/alu.circ` LU1, LU8

ALU Aufbau: Arithmetic Unit (AU)

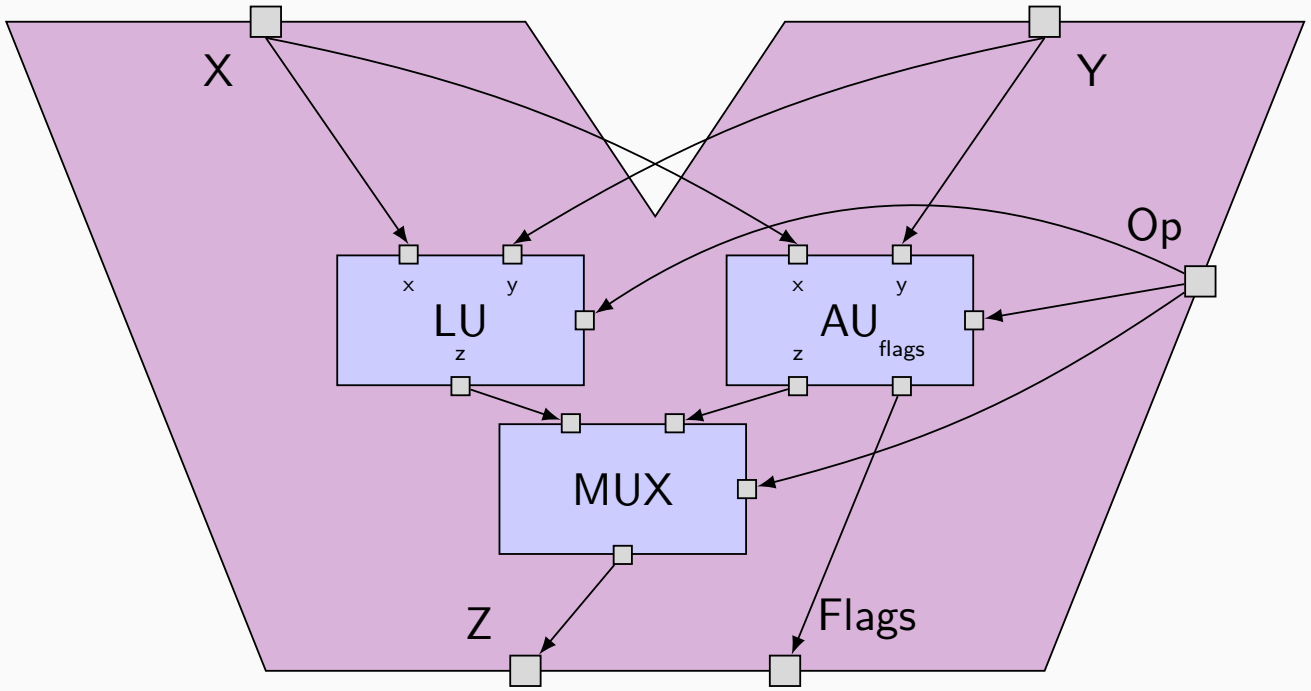
- Teil der ALU für arithmetische Operationen
- intern Übergabe des Übertrags
- Realisierung als Kaskade von 1-bit Arithmetic Units oder direkt für gewünschte Wortbreite, z.B. 32 Bit



demos/alu.circ AU8

ALU Aufbau: Implementierung

- Argumente und Teil des Operationscodes sowohl an LU als auch an AU
- Auswahl über MUX, gesteuert über restliche Bits der Operation



`demos/alu.circ` ALU8