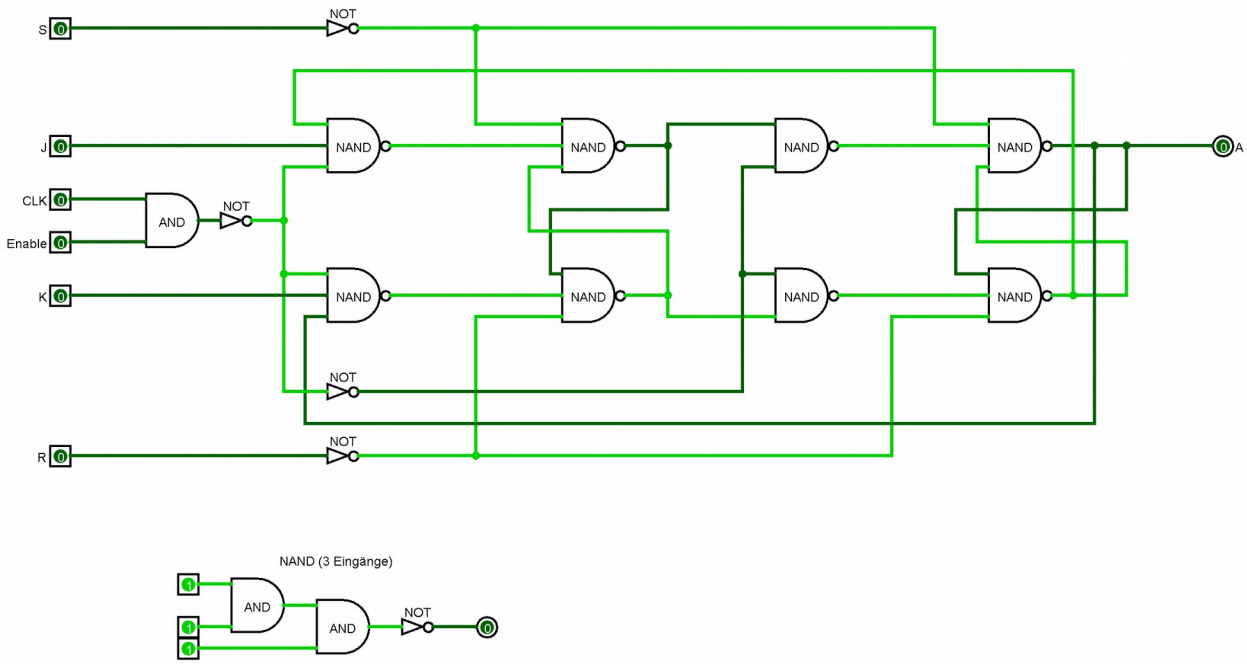
Do-It-Yourself CPU

4. Prozessor

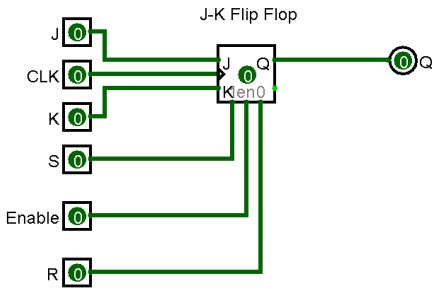
mail@AndreBetz.de

In diesem letzten Kapitel werden nun alle bisher dargestellten Bausteine zu einem Prozessor zusammengefügt. Da die Anwendung Logisim Probeleme hat die höheren Bausteine, die auf NAND Gattern beruhen zu simulieren, steige ich auf die internen Bausteine. Allerdings zeige ich, wie diese Bausteine intern aufgebaut sind.

1. J-K FlipFlop mit RS und Enable

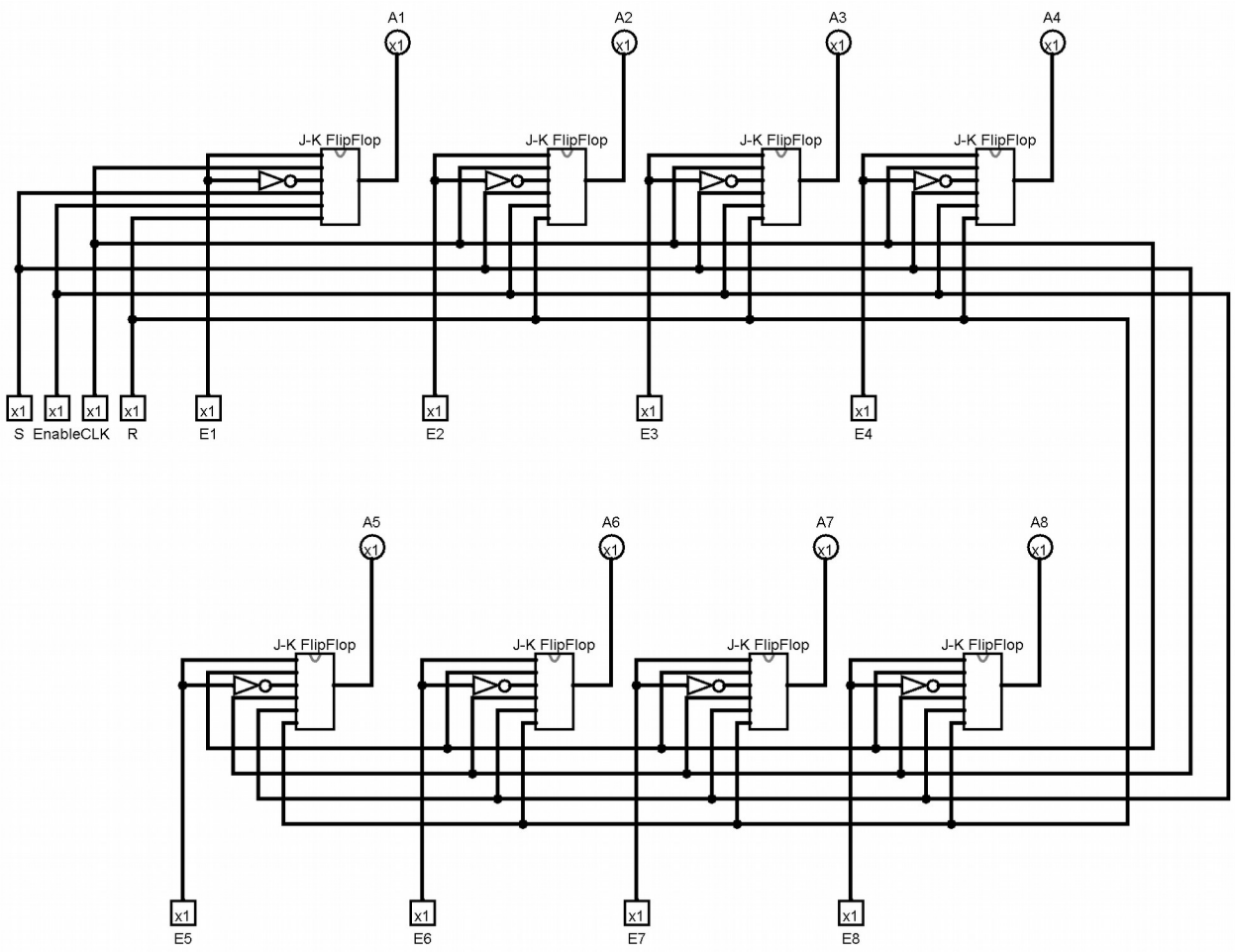


Das J-K FlipFlop bekommt noch eine Enable Leitung hinzu, die den Takt ein- bzw. ausschaltet. Im Bild ist noch einmal zur Verdeutlichung, wie ein 3-fach NAND Gatter aufgebaut ist.

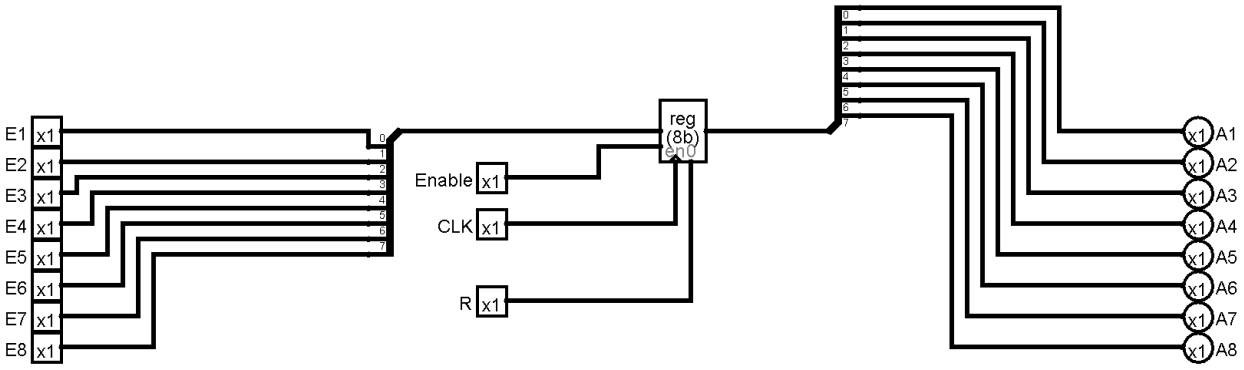


In Logisim gibt es ein eingebautes J-K FlipFlop mit der gleichen Funktionalität.

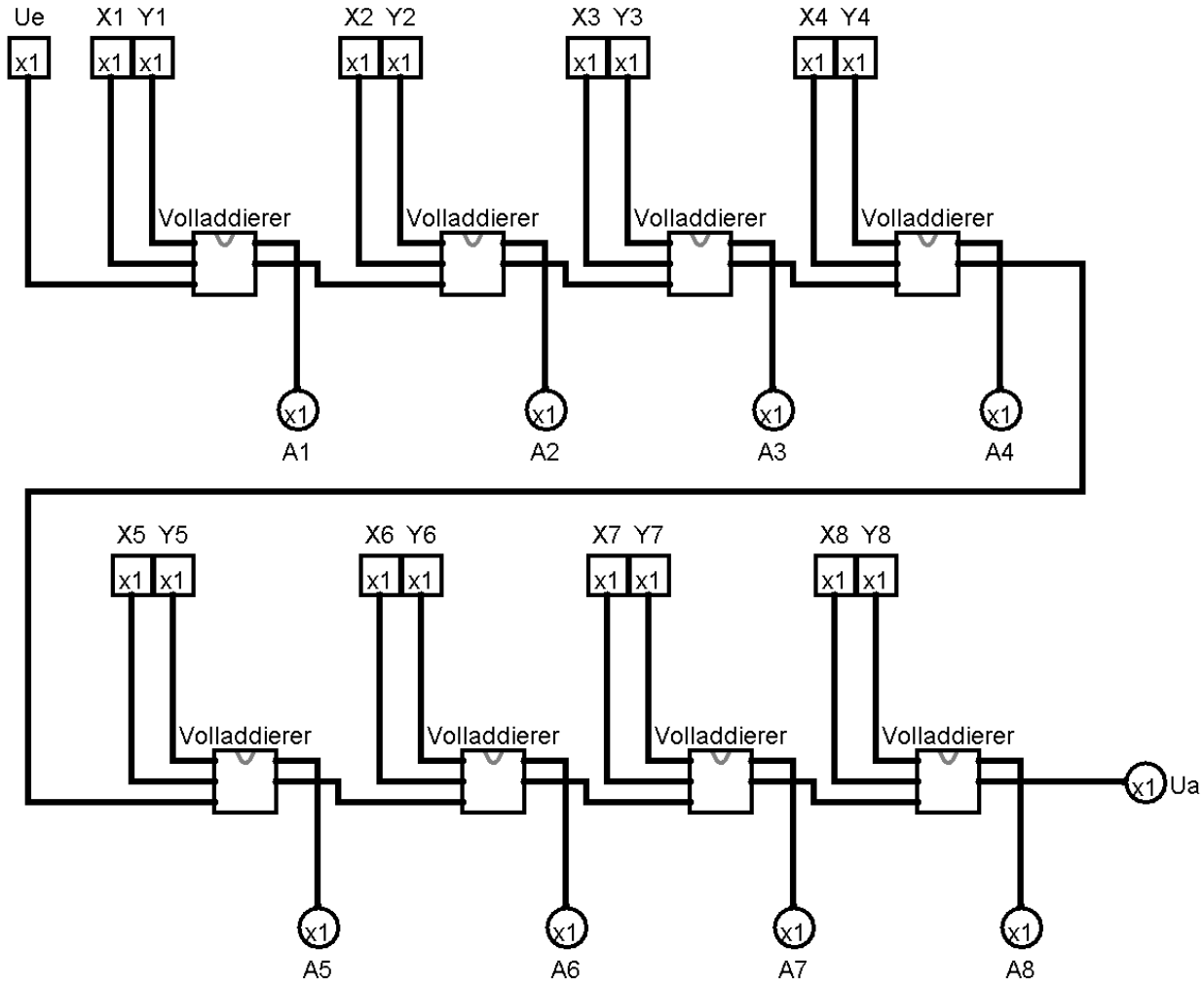
2. Register 8Bit



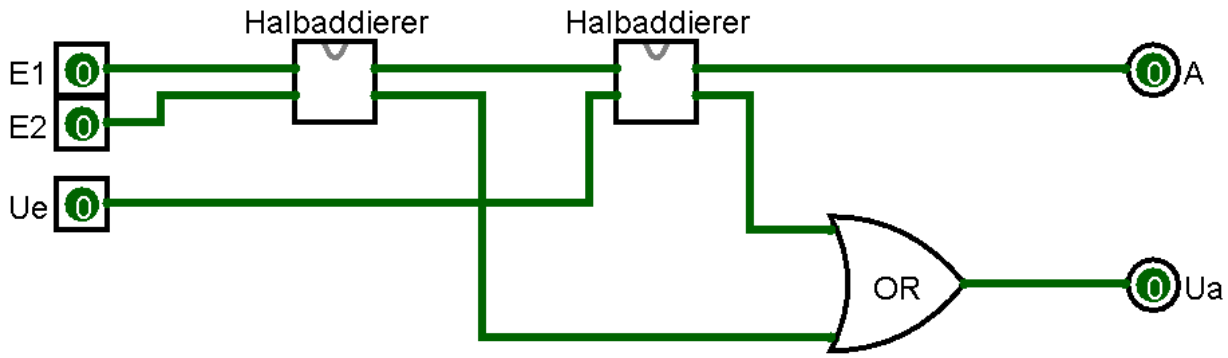
Ein Register, das 8Bit speichern kann ist in Logisim auch enthalten. Der interne Aufbau ist oben dargestellt.



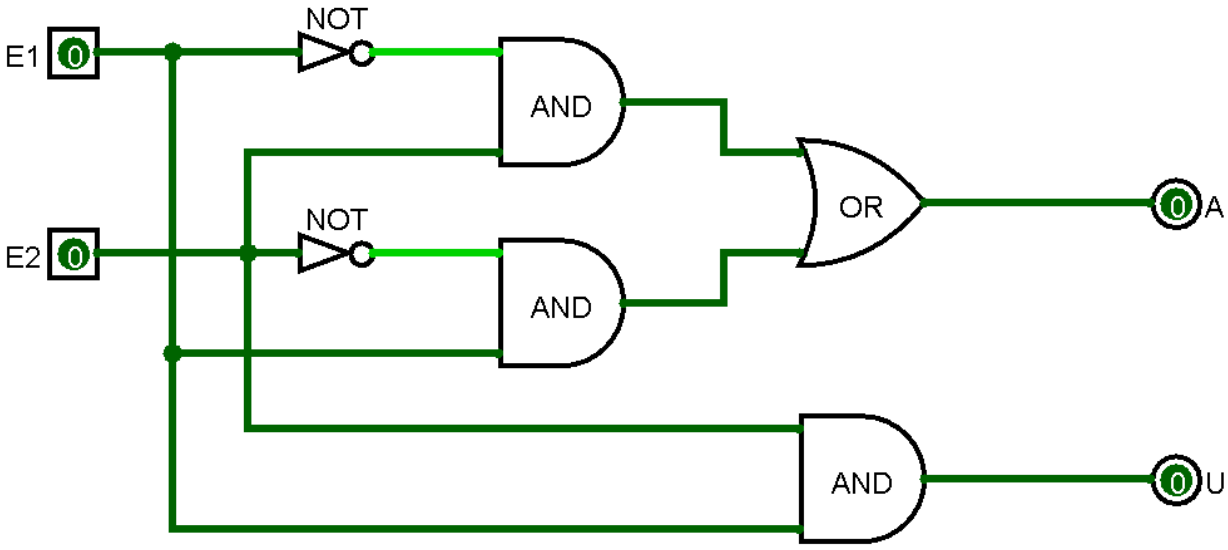
3. Addierer 8Bit

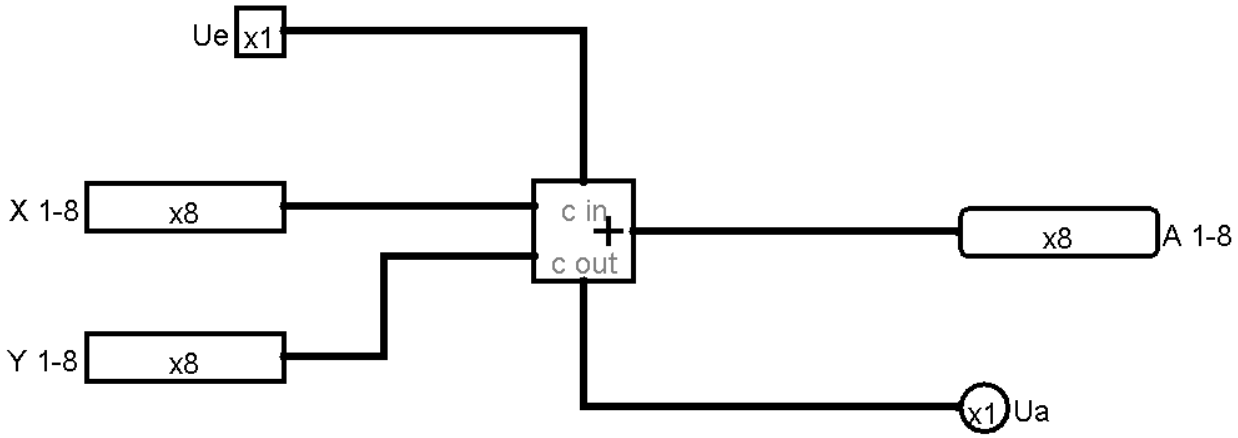


Oben ist die Funktionsweise eines 8Bit Addierers dargestellt. Die einzelnen Volladdierer bestehen wiederrum aus zwei Halbaddierern.



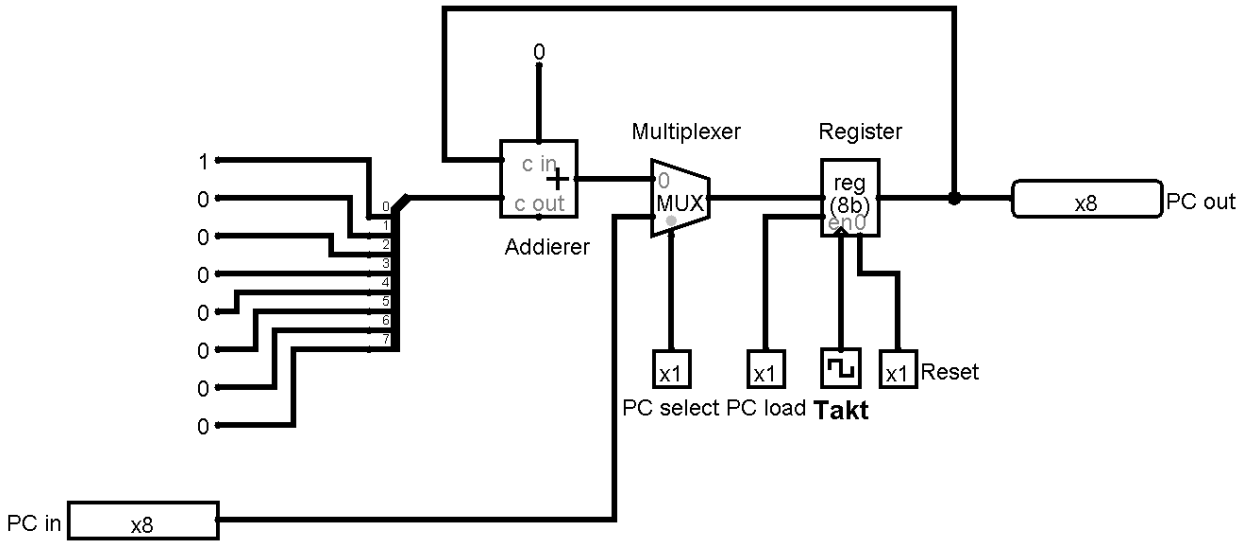
Und ein Halbaddierer ist wie folgt aufgebaut.





In Logisim ist ebenso ein 16Bit Addierer integriert

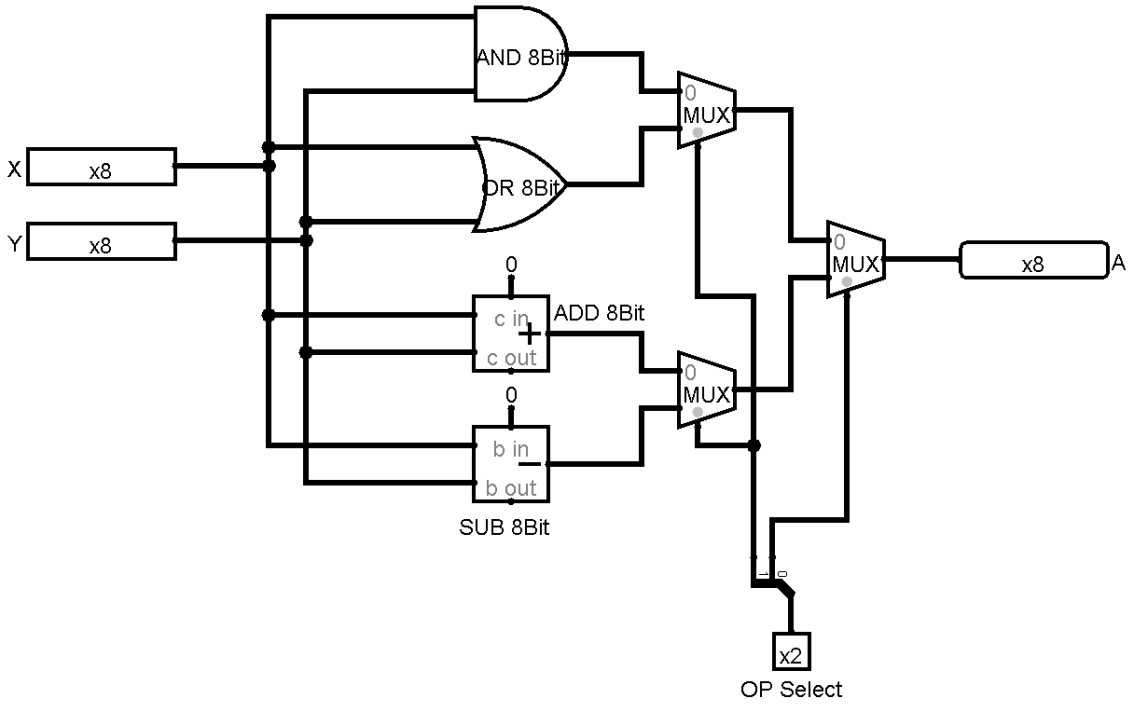
4. Program Counter



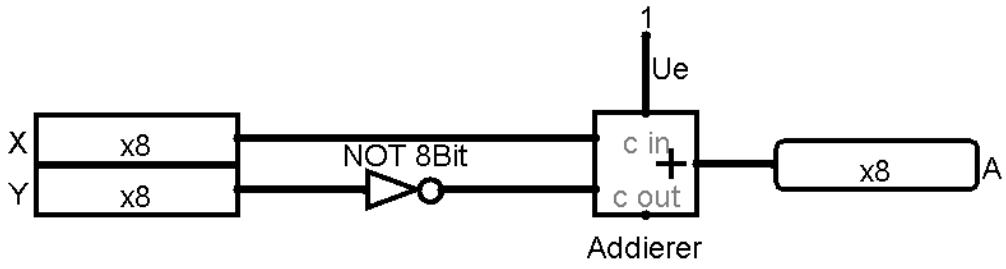
Der ProgramCounter (PC) beinhaltet die Adresse für den Speicher. Dieser wird nach jedem Schritt um eins hoch gezählt. Der PC kann auch direkt mit einer Adresse geladen werden für Sprünge. Dazu wird PC select auf eins gesetzt. Das Register wird nur geladen, wenn PC load auf eins ist.

5. ALU

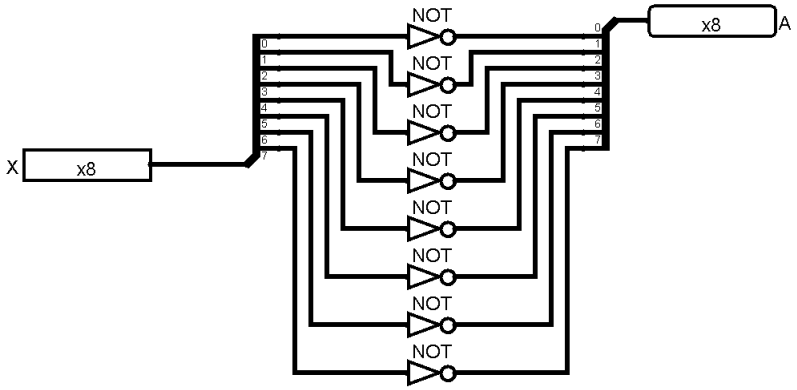
Die ALU ist die Einheit, die die Daten logisch bzw arithmetisch verknüpft. Für eine universelle CPU reichen vier arithmetische Befehle vollkommen aus. Mit AND, OR, ADD, SUB können alle mathematischen Operationen abgebildet werden.



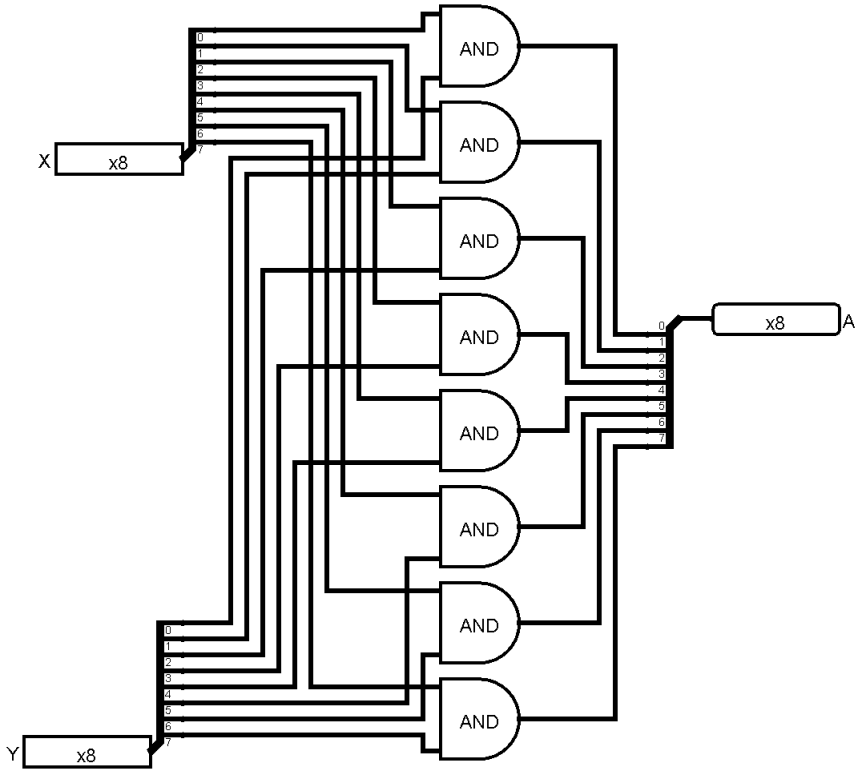
Über X und Y kommen die 8 Bit Werte in die ALU hinein. Alle vier Operationen werden gleichzeitig ausgeführt. Allerdings wird nur das Ergebnis an A ausgegeben, dass über die Multiplexer (MUX) mit OP Select eingestellt worden ist.



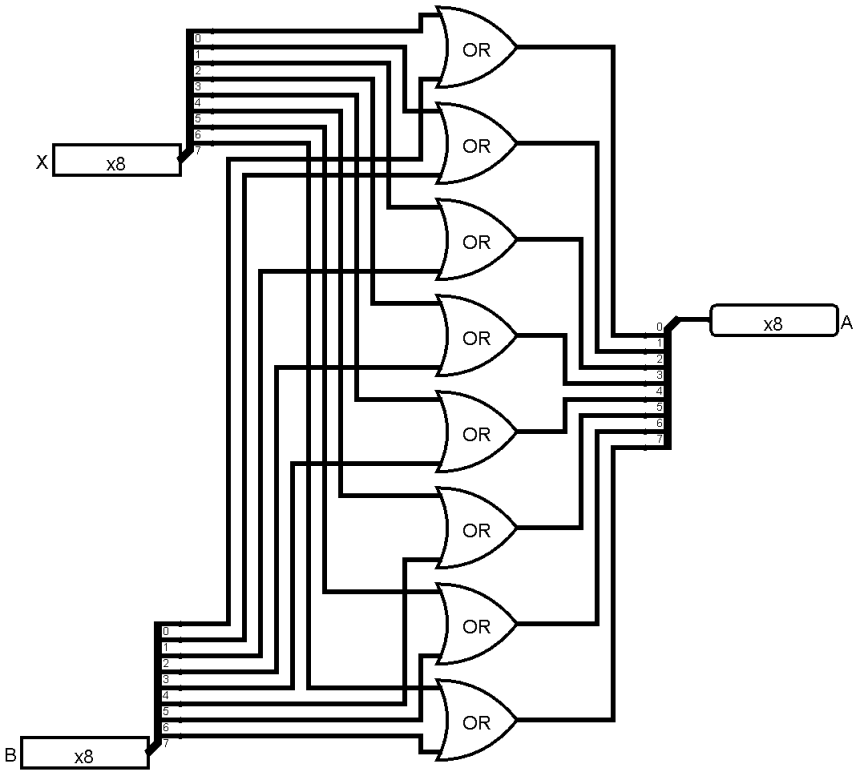
Die binäre Subtraktion kann auf eine binäre Addition zurückgeführt werden. Lediglich die Eingabe des abzuziehenden Operanden muss negiert und das eingehende Übertragsbit auf eins gesetzt werden, wie oben dargestellt.



Bei der 8Bit Negierung wird jedes einzelne Bit invertiert.

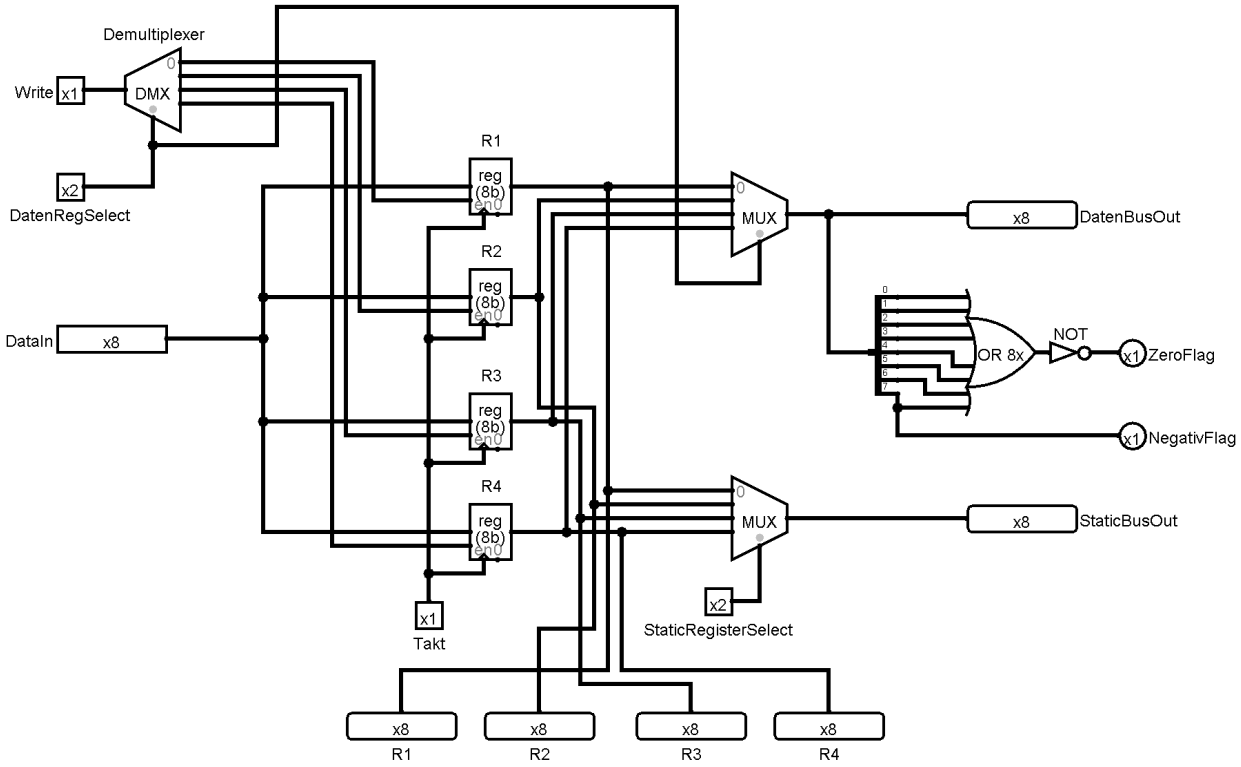


Beim 8Bit AND Gatter wird jede Datenleitung von X und Y verundet und an A ausgegeben. Das gleiche gilt für die 8Bit OR Funktion


X
x8
B
x8
OR
OR
OR
OR
OR
OR
OR
OR
x8
A

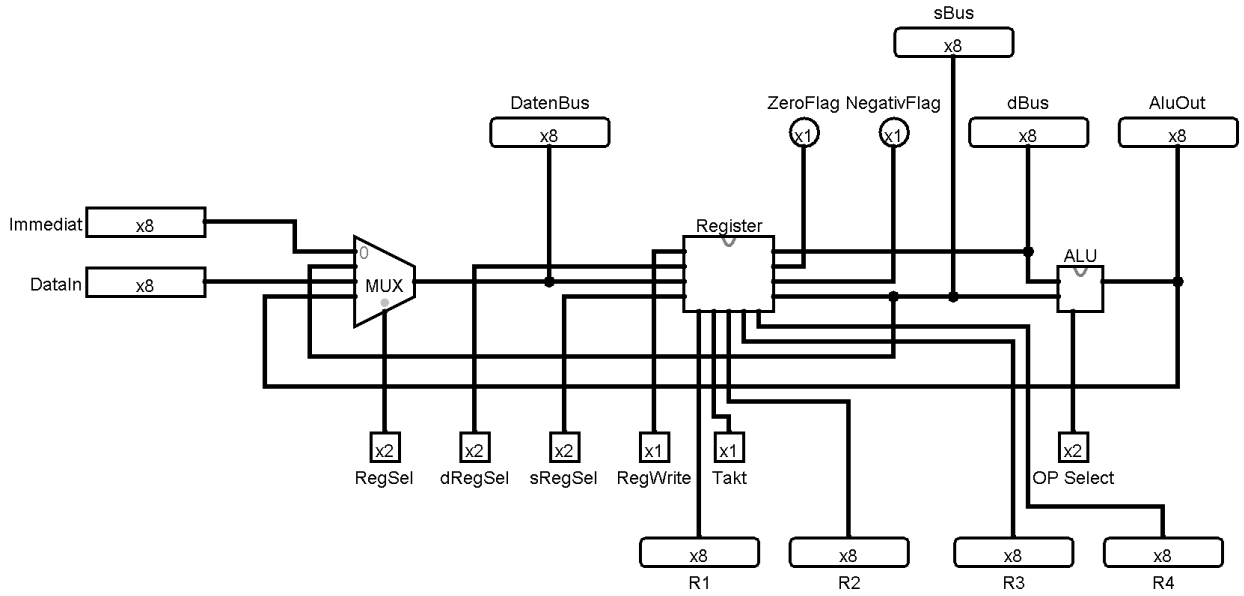
6. Register

Jede CPU hat Register, in denen sie Ausgaben von der ALU oder vom RAM zwischenspeichern kann.



In dieser CPU gibt es vier Register R1,R2, R3 und R4. Die Daten werden über DatenIn angelegt. Über RegSelect wird über den Demultiplexer (DMX) bestimmt, welches Register enabled wird aber nur, wenn Write auf eins gesetzt ist. Ist Write auf eins, so wird im nächsten Taktzyklus der DatenIn Wert ins das ausgewählte Regsiter geschrieben. Ausgelesen werden die Werte über zwei Multiplexer (MUX). Der obere MUX geht entscheidet welches Register an den DatenBusOut geht, das untere an StaticBusOut. Beide Busleitungen sind die Eingänge der beiden ALU Leitungen. Welches Register an welchen Bus ausgelesen wird bestimmt für den DatenBusOut das DatenRegisterSelect und für den StaticBusOut das StaticRegisterSelect. Zusätzlich werden am DatenRegisterOut noch das ZeroFlag und das NegativFlag bestimmt. Diese Flags sind entscheidend für Sprünge innerhalb Programme.

7. Register-ALU Verbindung



Die ALU und die Register werden über die Eingänge RegSel, dRegSel, sRegSel, RegWrite und OPSelect gesteuert. DataIn ist der Pfad. Über dem die Daten aus dem RAM kommen. Immidtiate ist ein zweiter Pfad über den ein zweiter Datensatz in die Register geladen werden kann. RegSel wählt dabei aus, woher die Daten in die Regsiter gelesen werden sollen. Diese können ausser DataIn und Immediat auch die Ausgabe der ALU sein oder der sBus. Folgende Befehlsfolge gilt für die Addition zweier Zahlen, die dann auch wieder im Register R1 gespeichert werden:

DataIn	**RegSel**	**dRegSel**	**sRegSel**	**RegWrite**	**OPSelect**	**R1**	**R2**
00000100	10	00	00	1	00	00000000	00000000
00000011	10	01	01	1	10	00000100	00000000
00000011	11	00	01	1	10	00000100	00000011
00000011	11	00	01	1	10	00000111	00000011

Jede Zeile repräsentiert einen Taktzyklus