

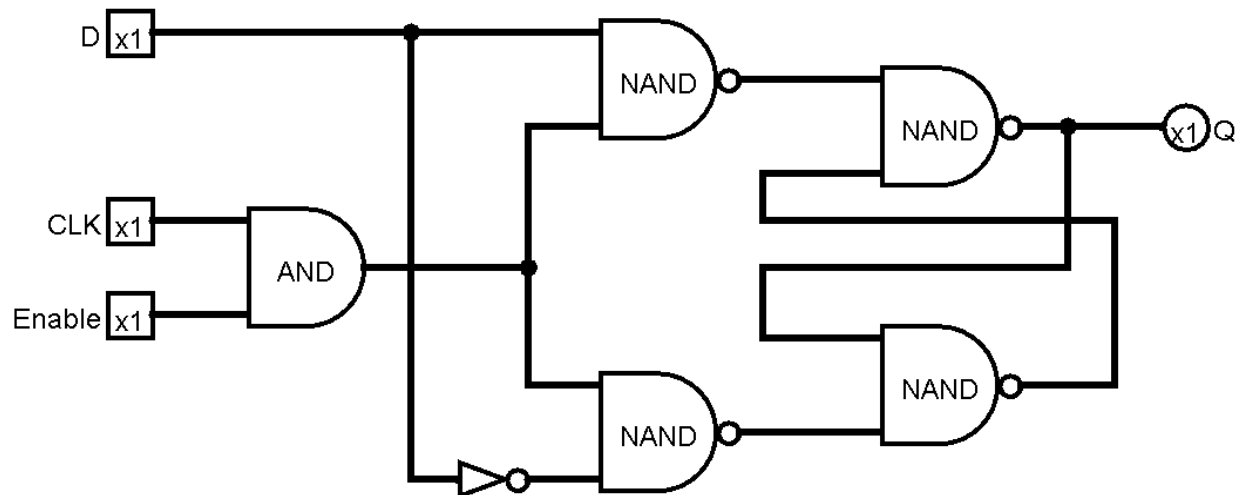
# Do-It-Yourself CPU

## 4. Prozessor

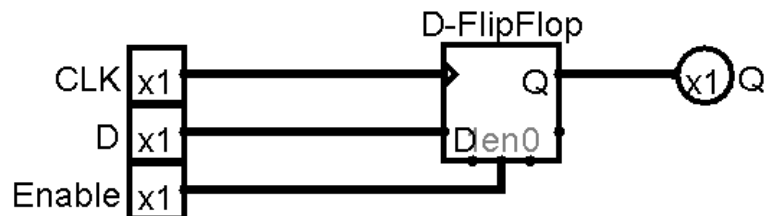
mail@AndreBetz.de

In diesem letzten Kapitel werden nun alle bisher dargestellten Bausteine zu einem Prozessor zusammengefügt. Da die Anwendung Logisim Probleme 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. D-FlipFlop mit Enable

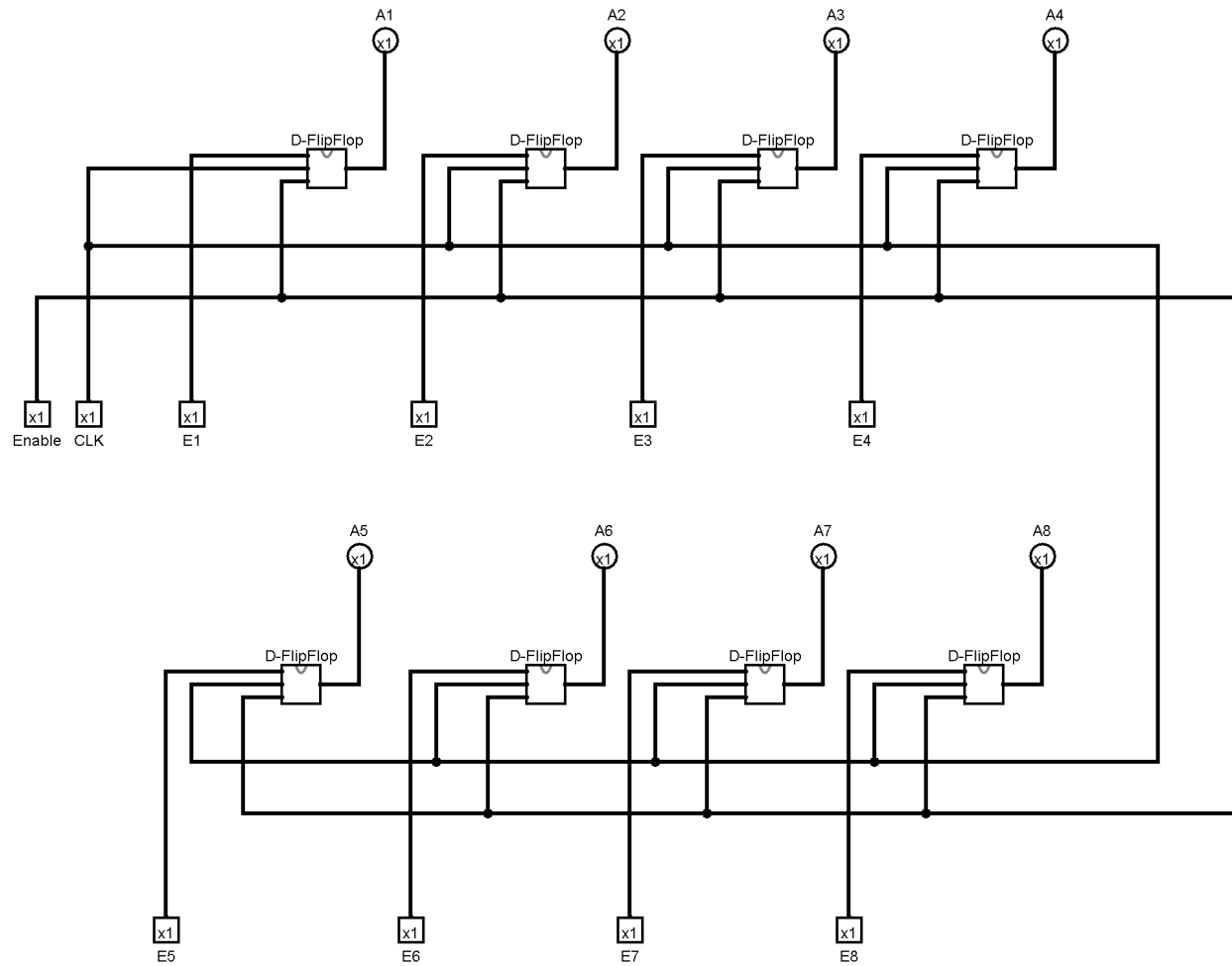


Für die Entwicklung der CPU reicht ein D-FlipFlop aus. Über den Eingang D kommt der Wert high oder low rein. Der Wert wird bei steigender Flanke (wechsel low auf high) am CLK gespeichert. Allerdings wird nur der Flankenwechsel weitergeleitet, wenn die Enable Leitung auf high liegt.



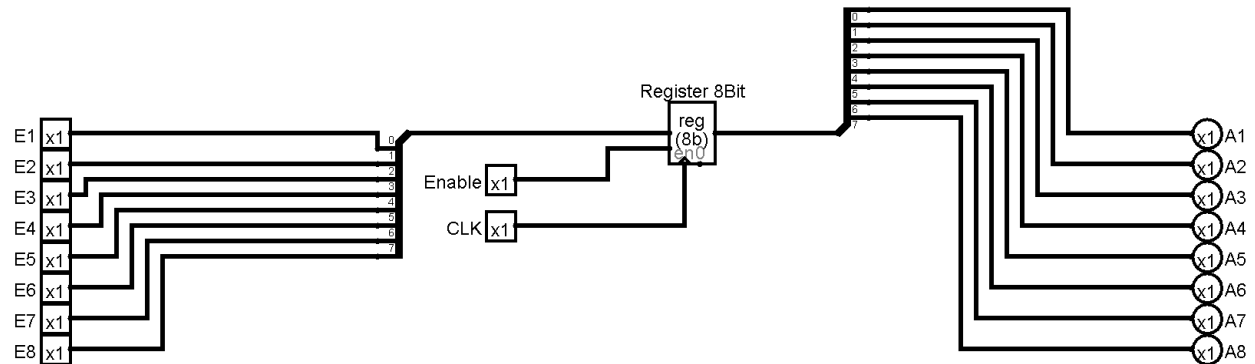
In Logisim sieht das Bauelement wie oben aus.

## 2. Register 8Bit

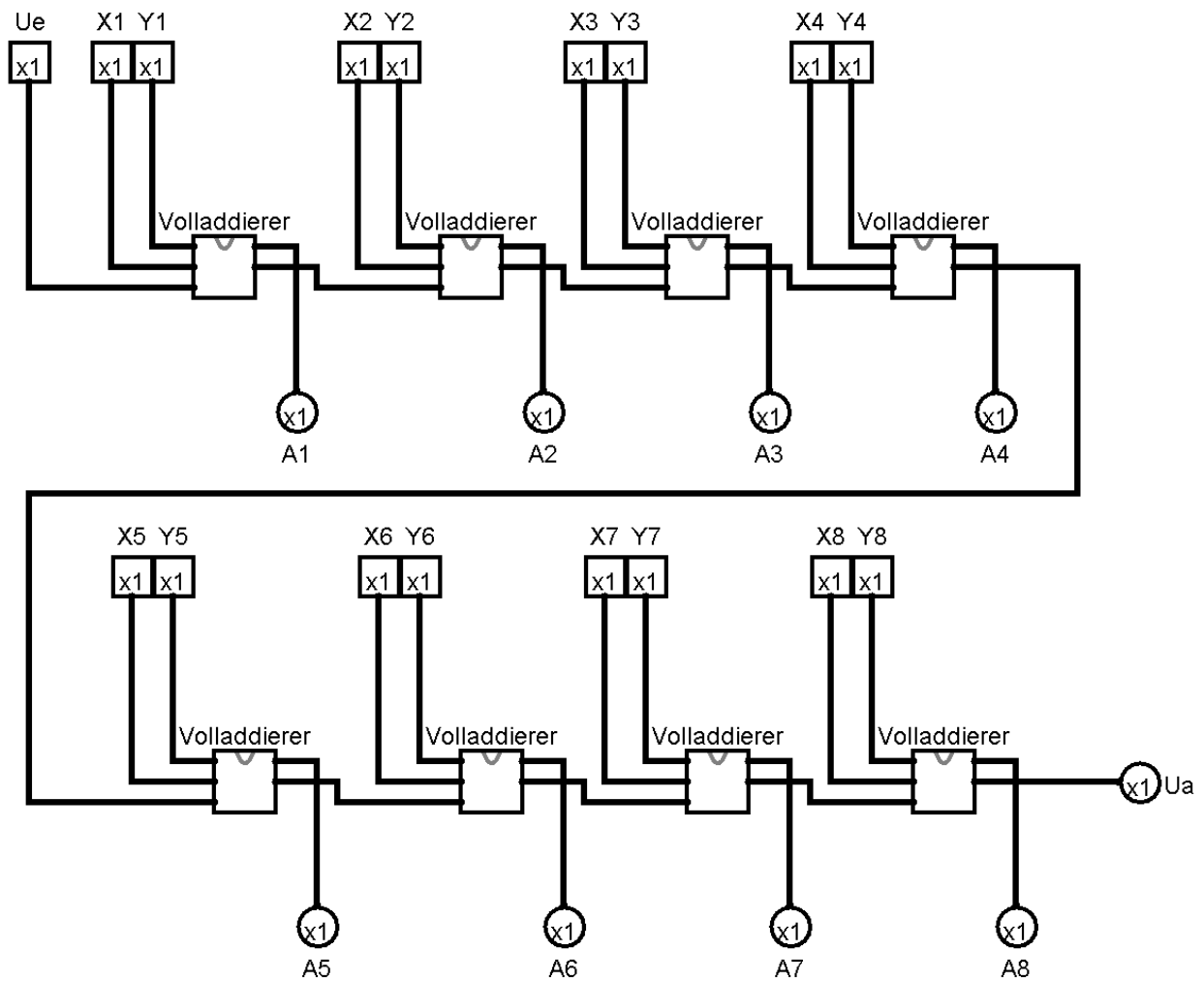


Ein 8Bit Register aufgebaut aus 8 D-FlipFlops

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

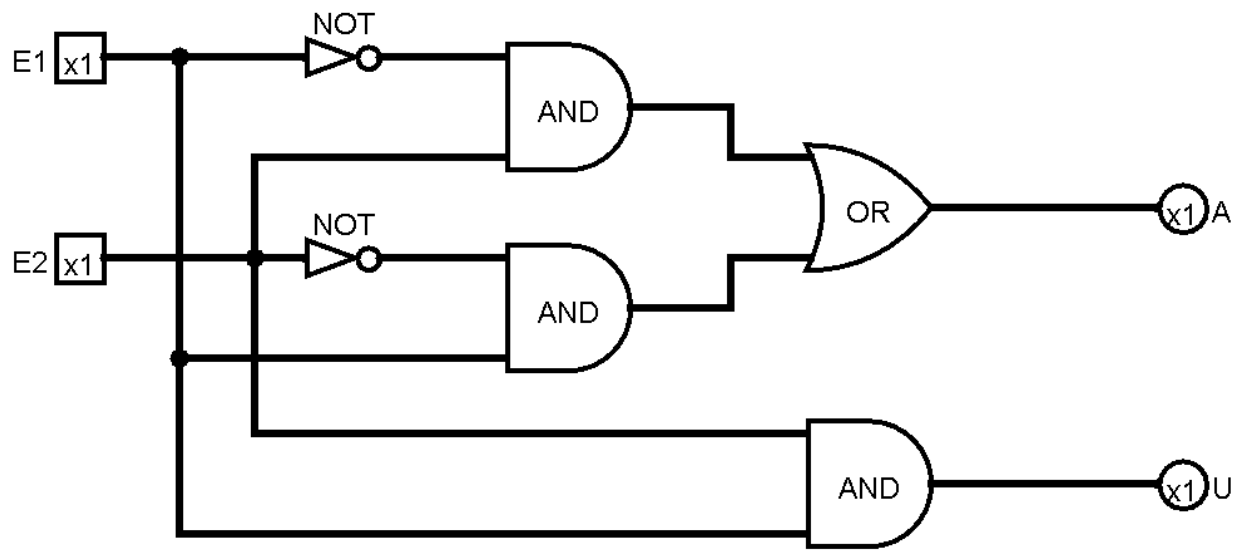


### 3. Addierer 8Bit

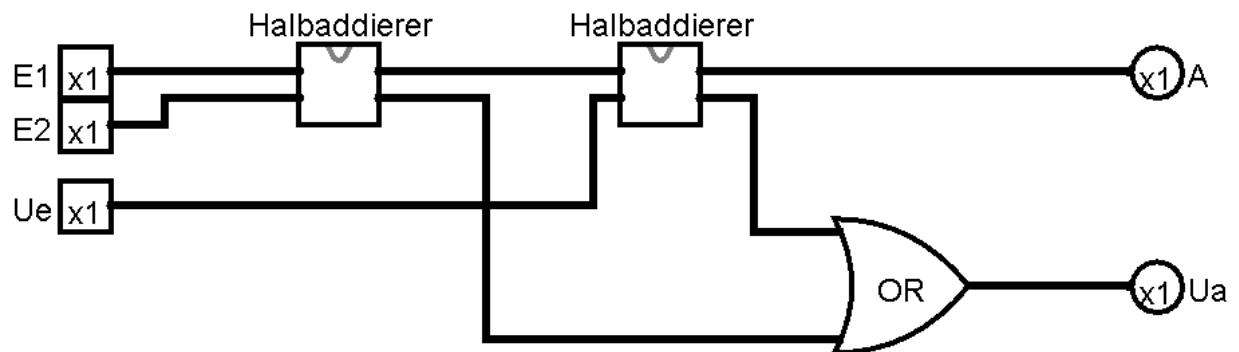


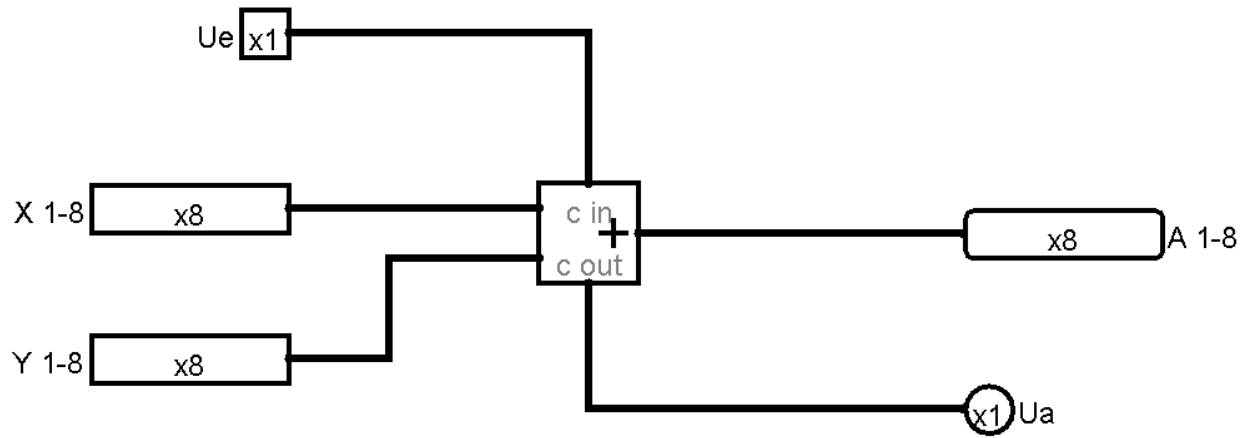
Oben ist die Funktionsweise eines 8Bit Addierers dargestellt. Die einzelnen Volladdierer bestehen wiederum aus zwei Halbaddierern.

Und ein Halbaddierer ist wie folgt aufgebaut.



Aus zwei Halbaddierern ergibt sich ein Volladdierer

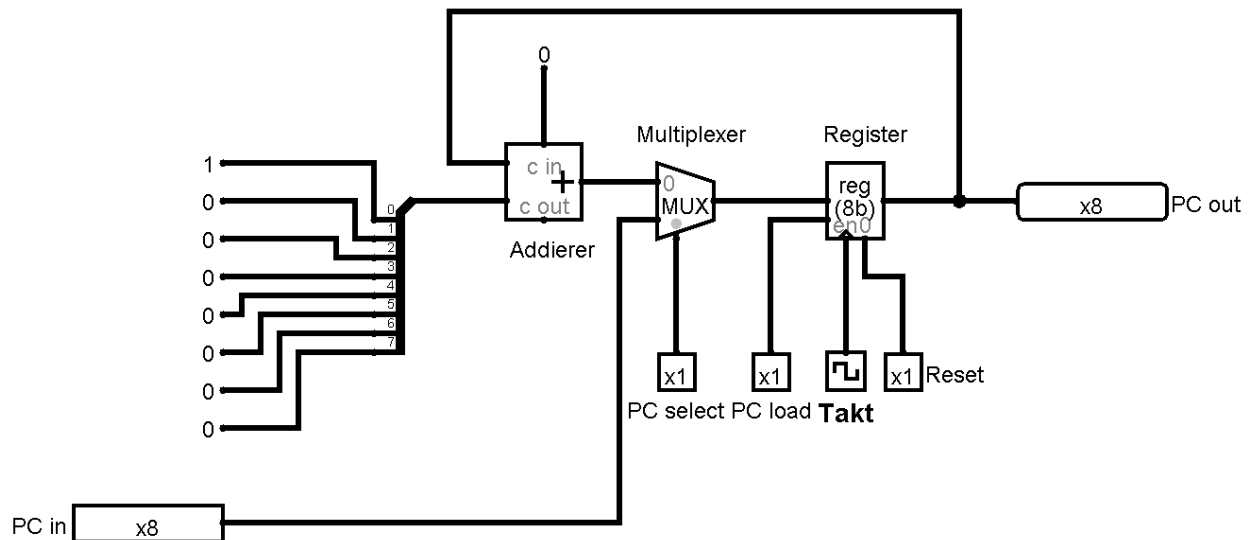




In Logisim ist ebenso ein 8Bit Addierer integriert



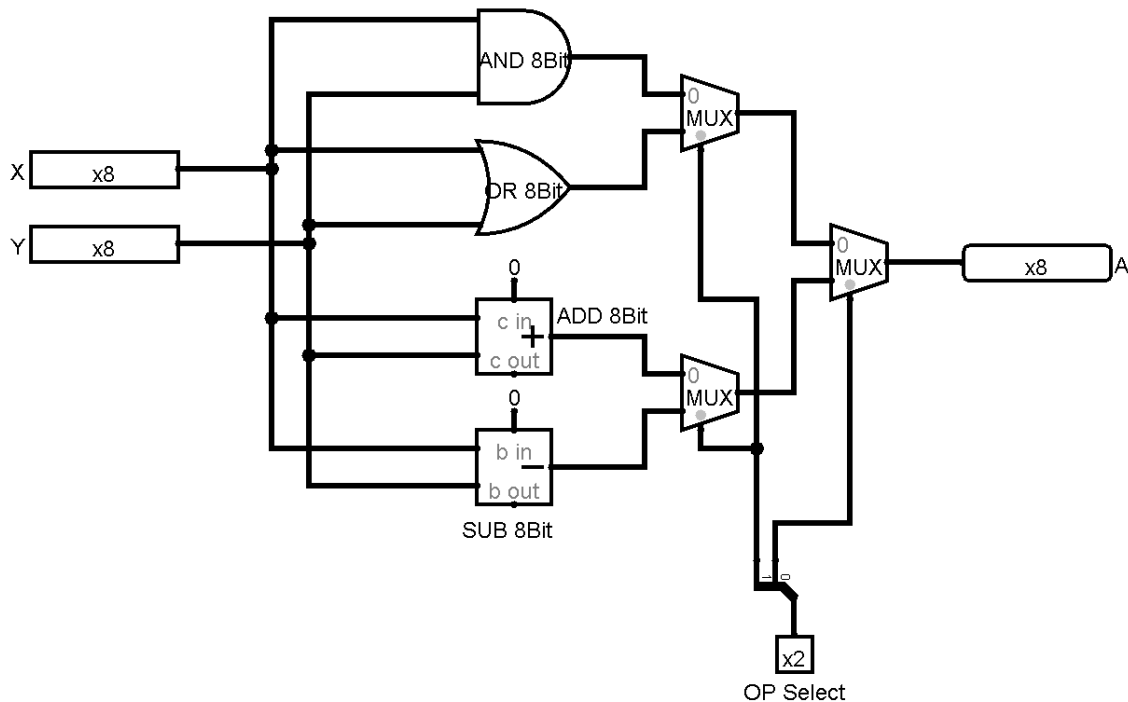
#### 4. Program Counter



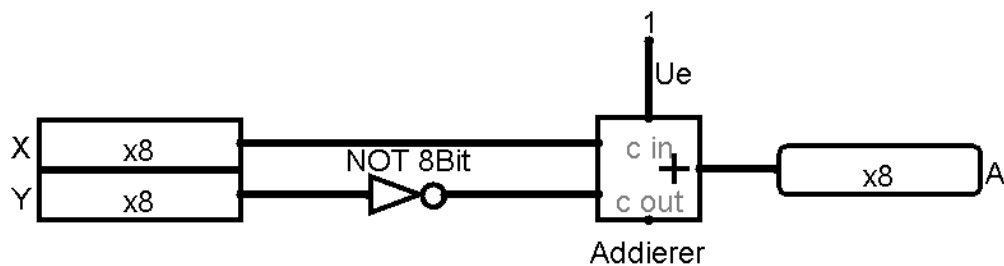
Der ProgramCounter (PC) beinhaltet die Adresse für den Speicher. Dieser wird nach jedem Schritt um eins hoch gezählt. Der PC kann auch direkt mit einer Adresse geladen werden für Sprünge. Dazu wird PC select auf eins gesetzt. Das Register wird nur geladen, wenn PC load auf eins ist.

## 5. ALU

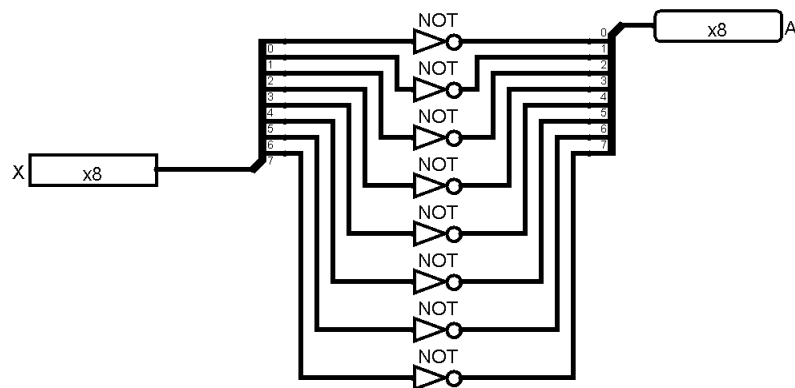
Die ALU ist die Einheit, die die Daten logisch bzw arithmetisch verknüpft. Für eine universelle CPU reichen vier arithmetische Befehle vollkommen aus. Mit AND, OR, ADD, SUB können alle mathematischen Operationen abgebildet werden.



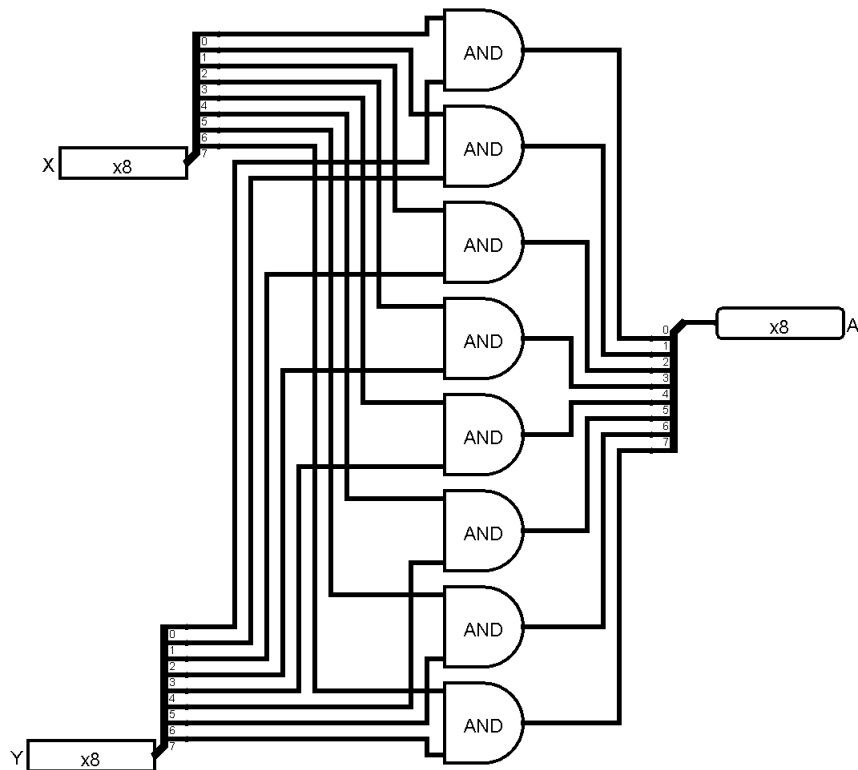
Über X und Y kommen die 8 Bit Werte in die ALU hinein. Alle vier Operationen werden gleichzeitig ausgeführt. Allerdings wird nur das Ergebnis an A ausgegeben, dass über die Multiplexer (MUX) mit OP Select eingestellt worden ist.



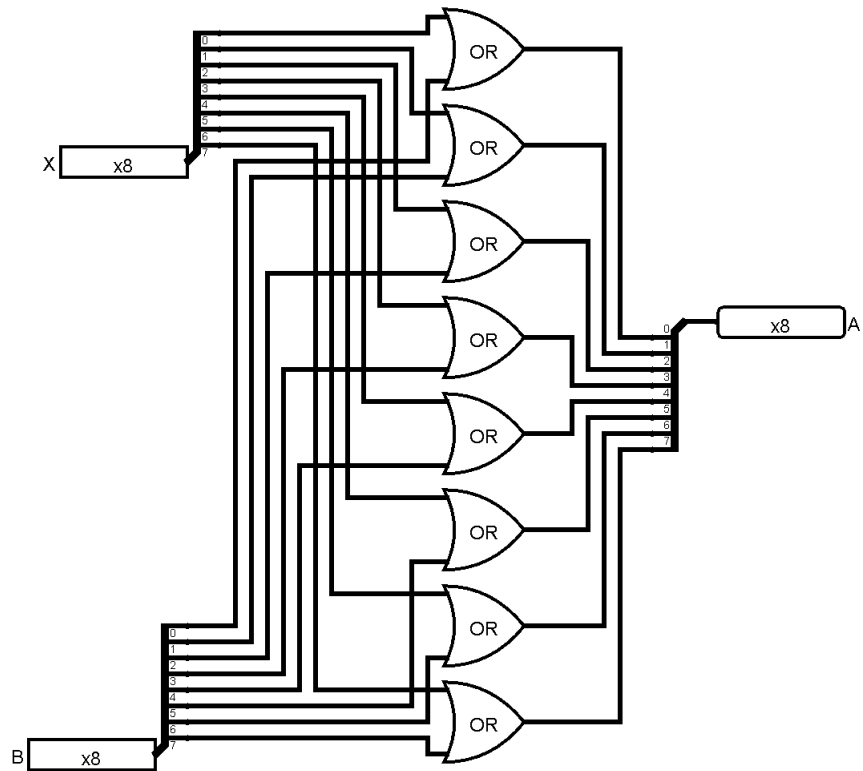
Die binäre Subtraktion kann auf eine binäre Addition zurückgeführt werden. Lediglich die Eingabe des abzuziehenden Operanden muss negiert und das eingehende Übertragsbit auf eins gesetzt werden, wie oben dargestellt.



Bei der 8Bit Negierung wird jedes einzelne Bit invertiert.

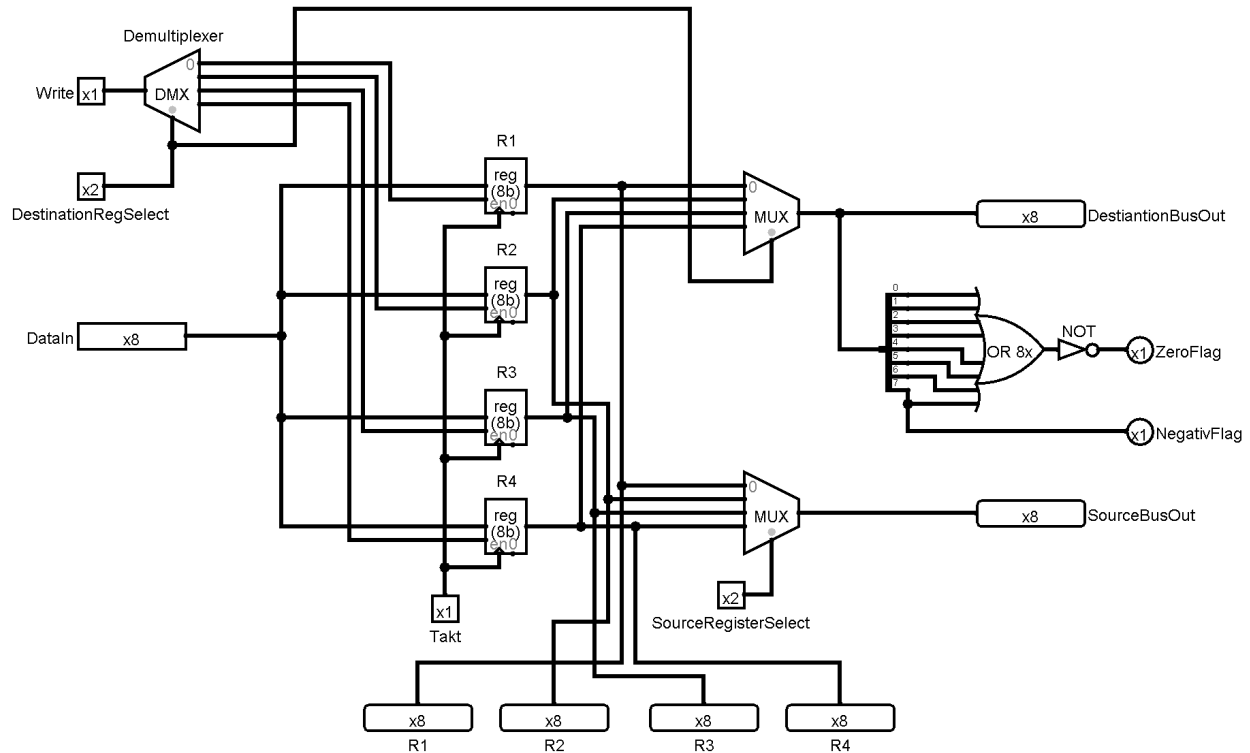


Beim 8Bit AND Gatter wird jede Datenleitung von X und Y verundet und an A ausgegeben. Das gleiche gilt für die 8Bit OR Funktion



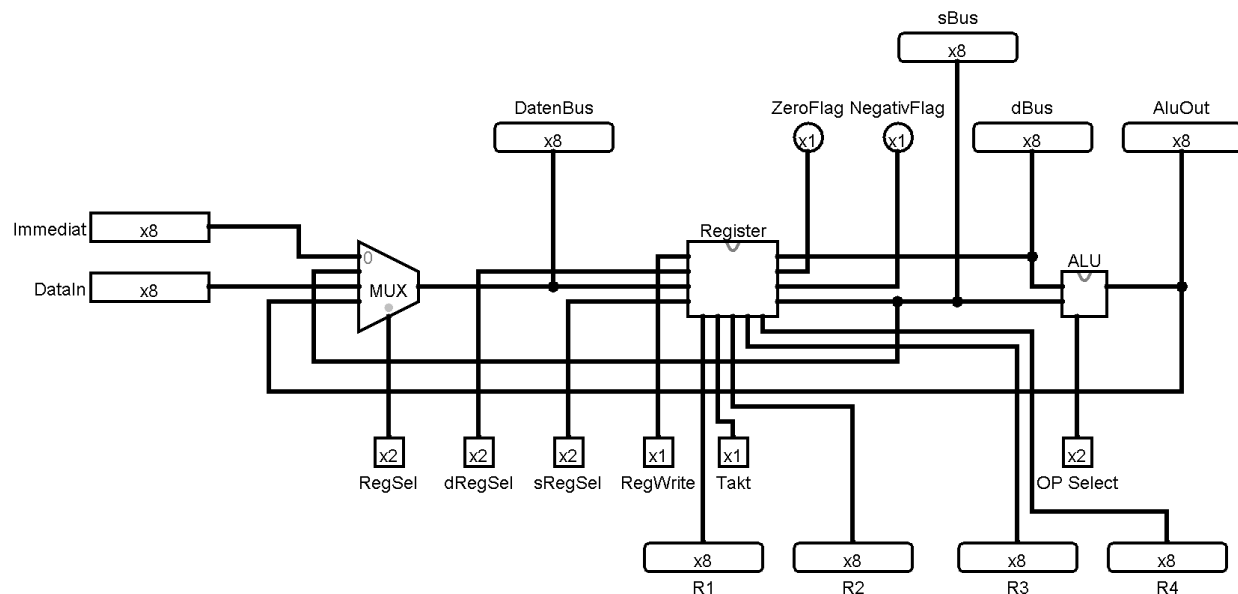
## 6. Register

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 Register 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



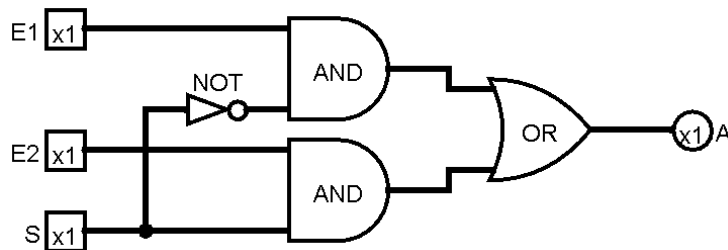
Die ALU und die Register werden über die Eingänge RegSel, dRegSel, sRegSel, RegWrite und OPSelect gesteuert. DataIn ist der Pfad, über dem die Daten aus dem RAM kommen. Immediat ist ein zweiter Pfad, über den ein zweiter Datensatz in die Register geladen werden kann. RegSel wählt dabei aus, woher die Daten in die Register 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

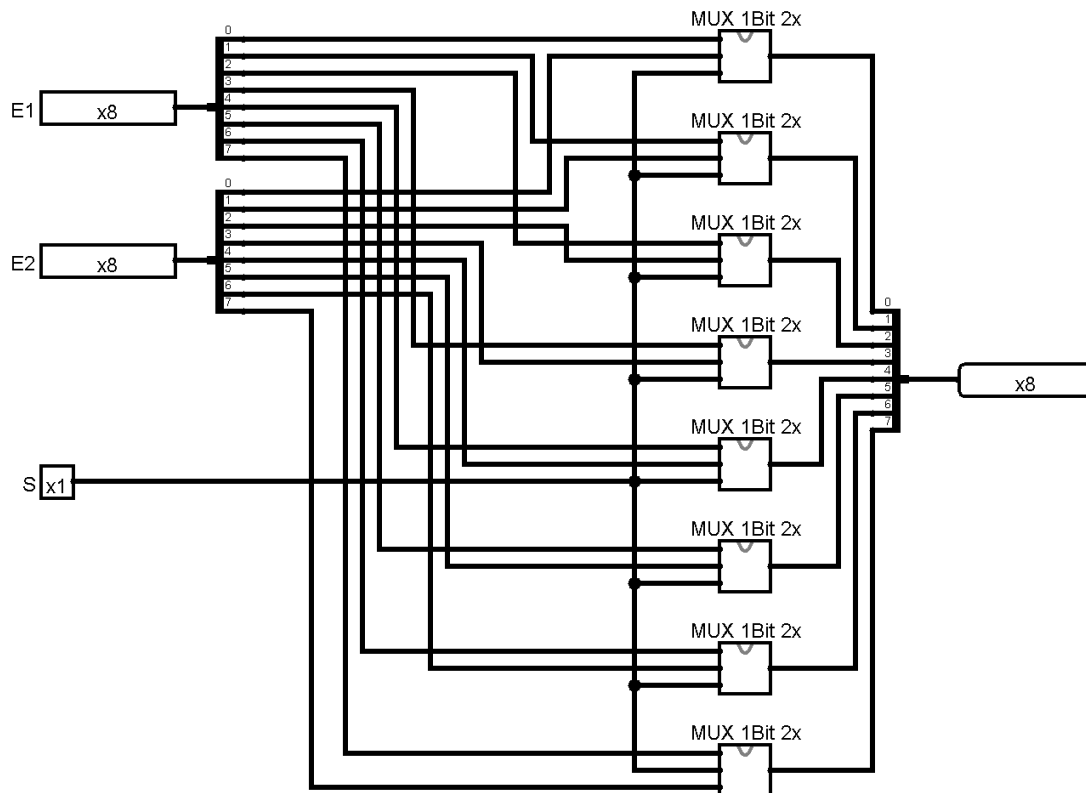
Jede Zeile repräsentiert einen Taktzyklus

## 8. Multiplexer

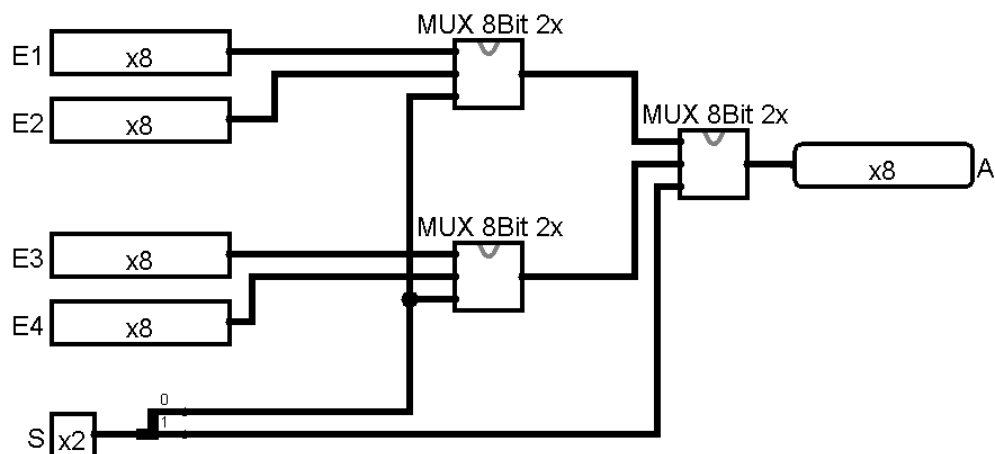
Um den Zugriff auf RAM Speicher zu verstehen, muss auch die Funktionalität der Multiplexer (MUX) und Demultiplexer (DeMUX) verstanden werden. Dabei handelt es sich um Bausteine, die wie Weichen funktionieren. Mehrere Datenstränge führen zu einem Strang (Multiplexer) oder ein Strang wird aufgefächert auf mehrere Stränge. Dabei entscheidet eine Steuerleitung S, von welchem Eingangsstrang E(1-n) die Daten genommen werden, um diese auszugeben A (Multiplexer) oder die Steuerleitung S entscheidet, an welchen Ausgang A(1-n) ein Strang gehen soll (Demultiplexer). Dabei wird beim Multiplexer einmal zwischen der Anzahl der Datenleitungen, die geschaltet werden unterschieden und die Anzahl der Verzweigungen.



Oben ist ein 1Bit 2x Multiplexer (MUX 1Bit). Er legt eine der zwei 1Bit Eingangsleitung abhängig des Schalters S auf A.

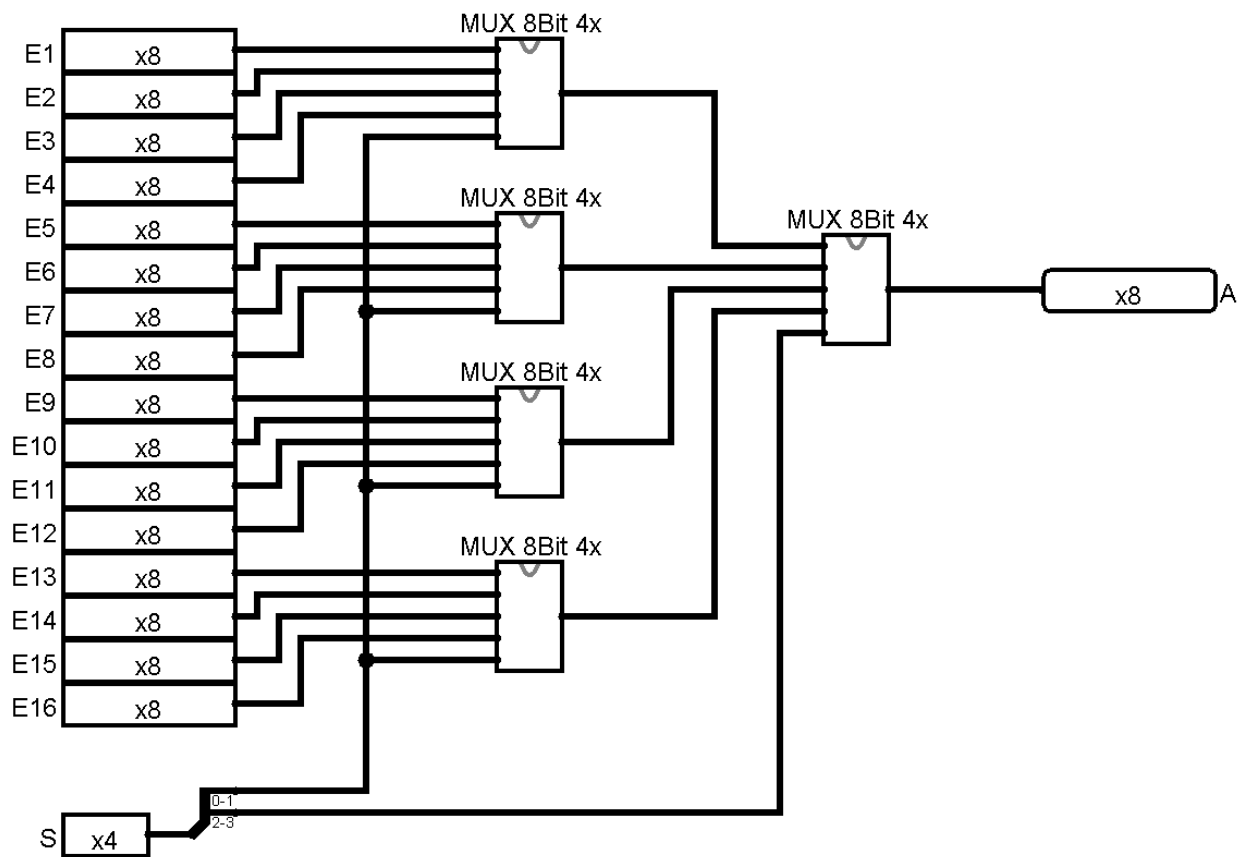


Aus 8 MUX 1Bit 2x kann man einen MUX 8Bit 2x herstellen. Dieser schaltet abhängig vom Schalter S die zwei 8Bit Eingänge auf einen 8Bit Ausgang A. Daraus lässt sich wiederum ein MUX 8Bit 4x generieren. Sie Steuerleitung benötigt nun 2 Bits.

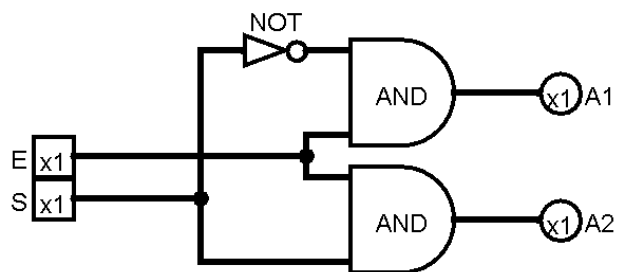


Daraus lässt sich dann ein MUX 8Bit 16x erzeugen und 4Bit Steuerleitungen. Dies kann beliebig so weiter geführt werden.

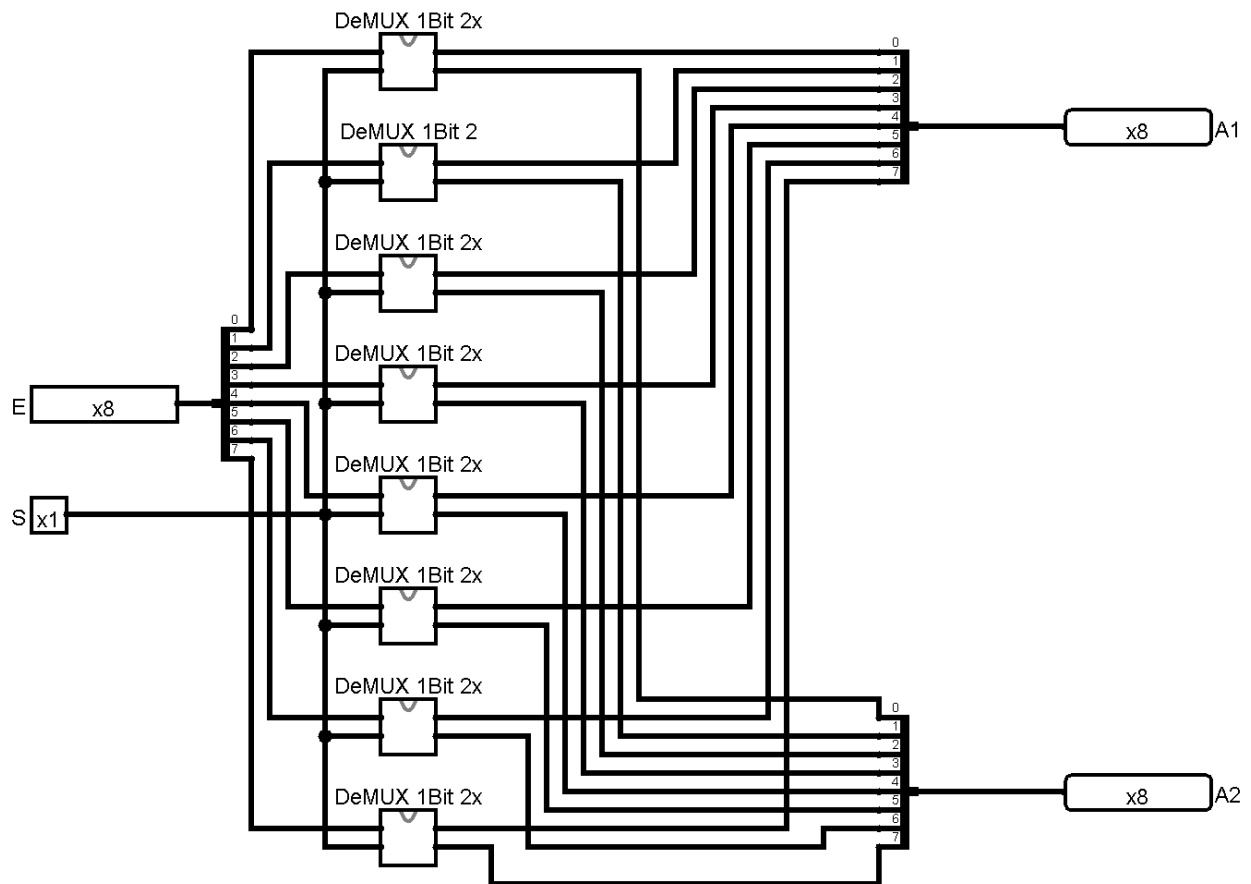




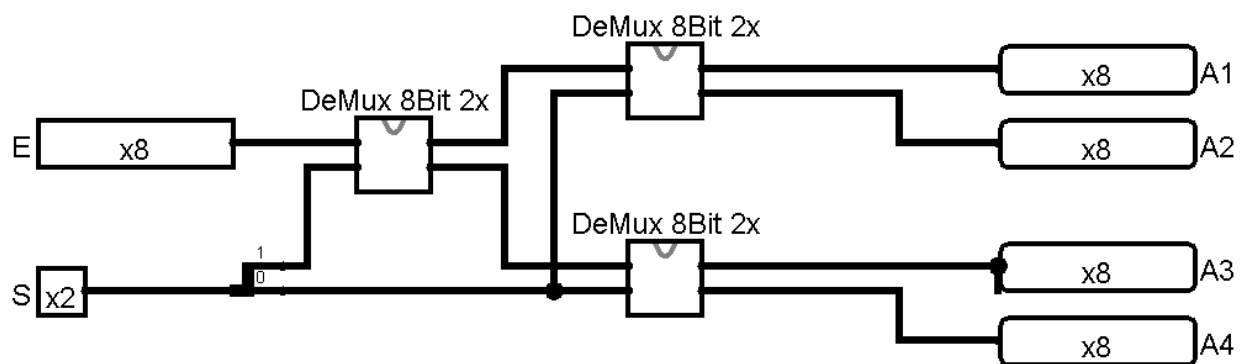
Ähnlich ist der Aufbau beim DeMultiplexer. Unten ist ein DeMUX 1Bit 2x. Er sendet ein Eingangssignal E abhängig vom Steuersignal S an den Ausgang A1 oder A2. Hier benötigt das Steuersignal lediglich 1Bit.



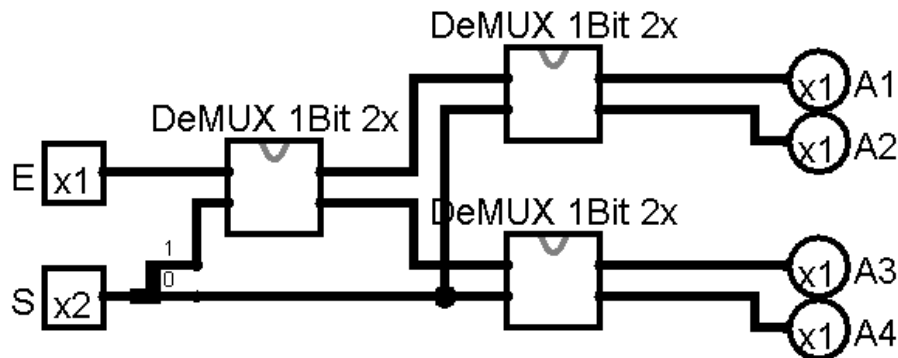
Der DeMUX 8Bit 2x besteht aus 8 DeMUX 1Bit 2x Bausteinen.



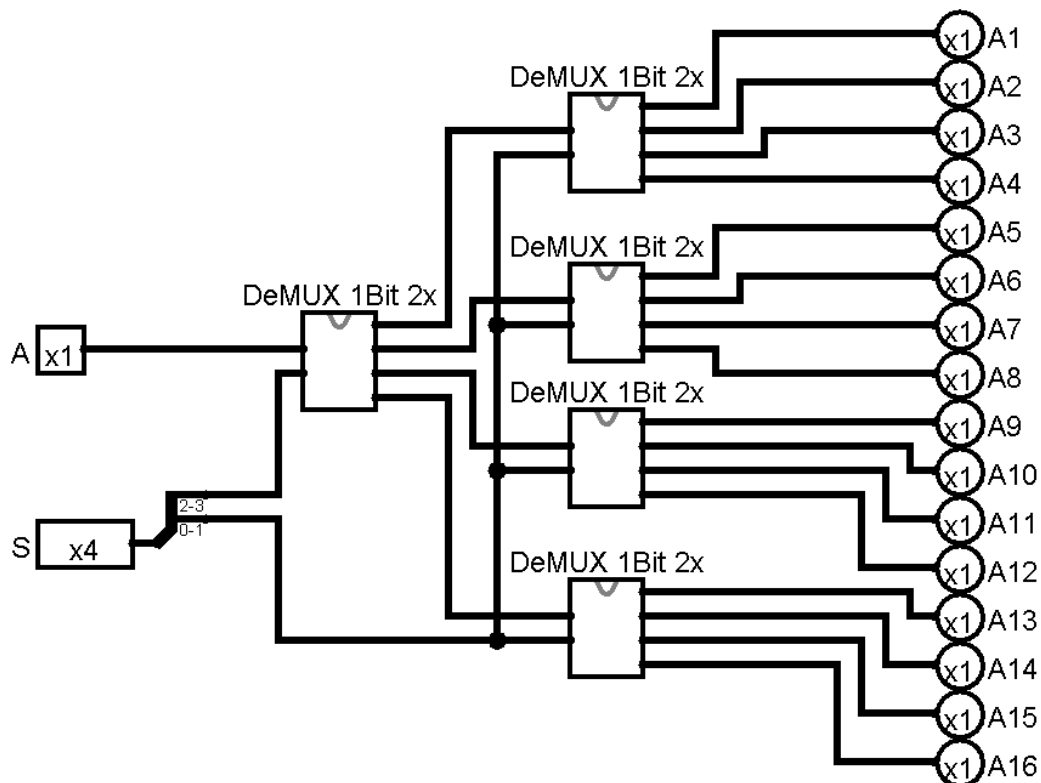
Aus drei DeMUX 8Bit 2x lässt sich ein DeMux 8Bit 4x herstellen mit 2Bit Steuerleitung



Bei DeMultiplexern benötigen wir allerdings eher die 1Bit Variante mit einer höheren Ausfächerung. Im Speicherbaustein wird darüber das FlipFlop eingeschaltet, in dem der Wert gespeichert werden soll. In dem unteren Beispiel wird eine 1Bit Eingangsleitung E auf 4 Ausgangsleitungen A1-A4 abhängig der 2Bit Steuerleitung S geschaltet.

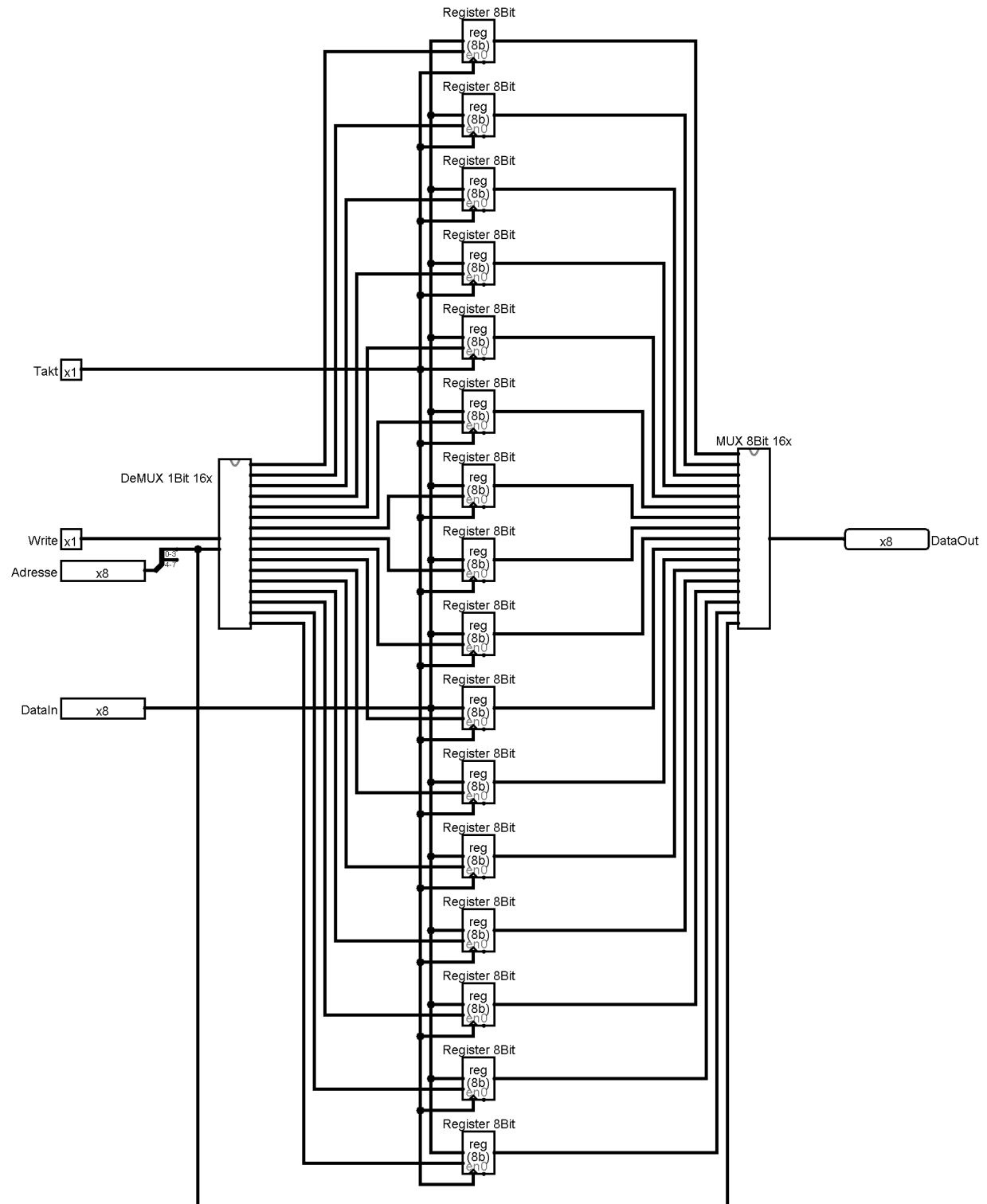


Und daraus ein DeMUX 1Bit 16x erstellen.

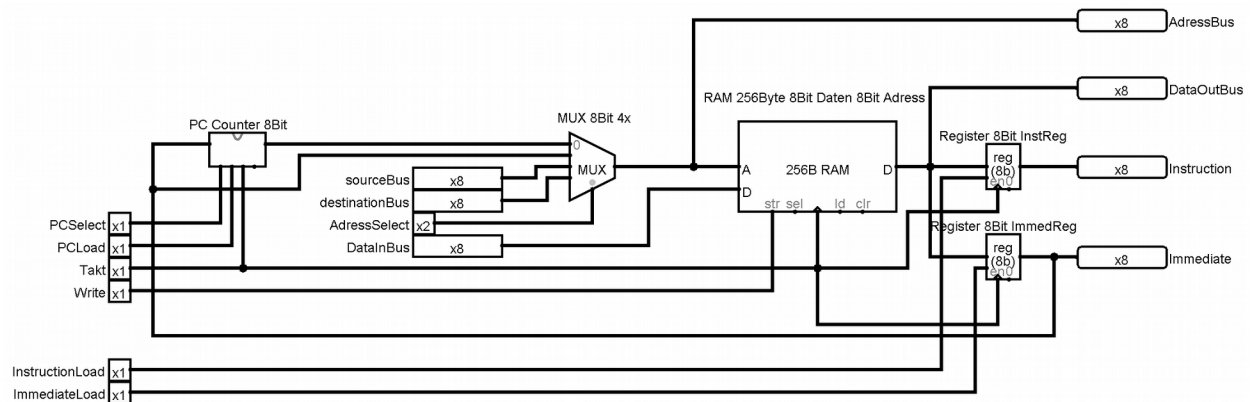


Auch dies lässt sich beliebig weiterführen. Bei 8Bit Steuerleitungen ergibt sich 256 Ausgangsleitungen.

## 9. RAM



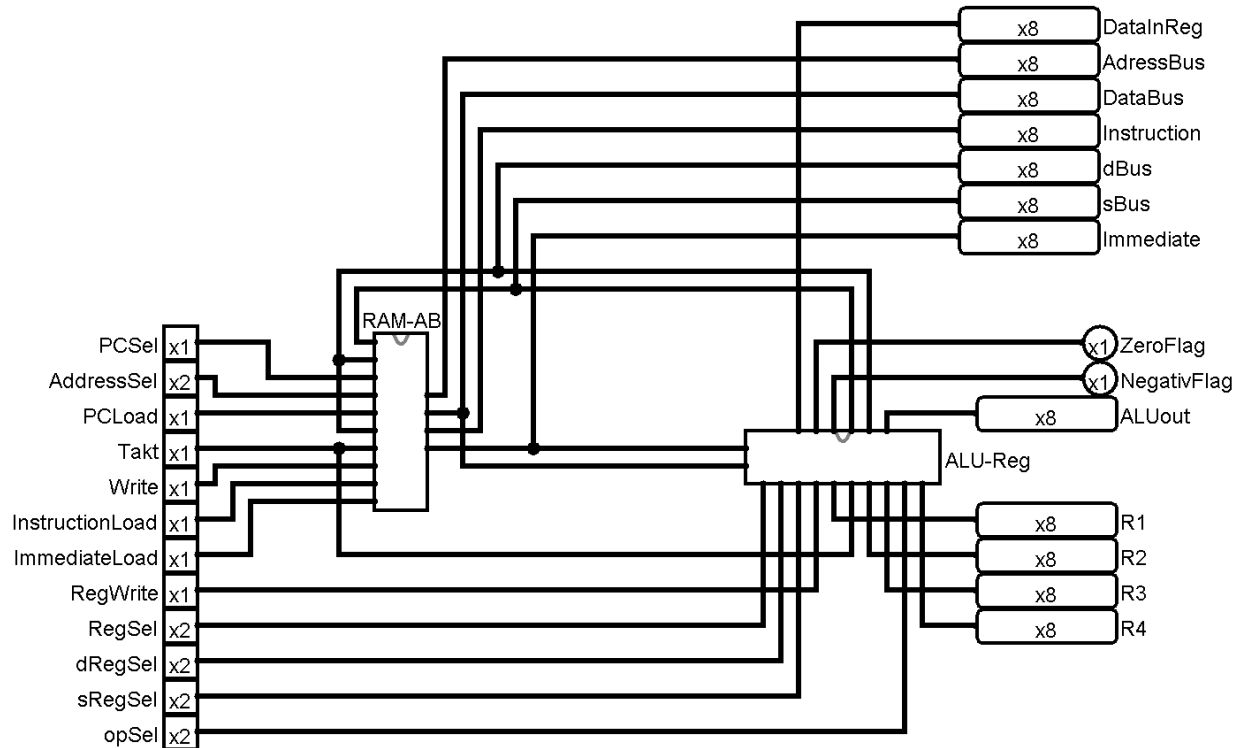
Das RAM (Random Access memory) wird definiert durch die breite der Datenleitung und Adressleitung. Im obigen Beispiel können 16 Bytes gespeichert werden. In welchem Register das Datum DataIn gespeichert wird hängt von dem DeMUX 1Bit 16x bestimmt, der beim Register die entsprechende Enable Eingang auf 1 legt, abhängig von der Steuerleitung, die beim RAM die Adressleitung ist. Bei 16 Register reichen eigentlich 4 Adressleitungen aus. Allerdings wird im Folgenden im Logisim ein fertiger 256Byte RAM verwendet und somit werden 8 Adressleitungen benötigt. Allerdings würde es hier den Rahmen sprengen ein komplettes 256Byte RAM aufzumalen. Auch macht es keinen Sinn, da es generisch aus den Vorinformationen konstruiert werden kann.



In der obigen Schaltung kommt die Adresse über den AdressBus in das 256Byte grosse RAM von einem MUX 8Bit 4x. Mit AdressSelect wird ausgewählt, ob vom sourceBus (sBus), destinationBus (dBus), vom PCCounter oder direkt aus dem ImmediateRegister, die Speicheradresse kommen soll.

Die Daten kommen über den DataInBus in das RAM, werden aber nur abgespeichert, wenn das Write Bit gesetzt ist. Gleichzeitig liegt am DataOutBus der ausgelesene RAM Registerwert an. Je nachdem, welches Bit im InstructionLoad oder ImmediateLoad gesetzt ist wird der DataOutBus Wert in einem der Register gespeichert. Der PCCounter wird gesteuert über PCSelect, ob eine Adresse über das Immediate Register gelesen werden soll oder der interne Register Wert hochgezählt werden soll.

## 10. Datenpfad



In dem obigen Bild ist der komplette Datenpfad aufgezeigt. ALU und RAM sind miteinander verbunden. Auf der linken Seite befinden sich alle Steuerleitungen und die rechte Seite zeigt die Daten, die in der Vorstufe der CPU laufen. Prinzipiell fehlt noch eine Steuereinheit, die je nach Zustand die entsprechenden Steuerleitungen schaltet

Takt	PCSel	AdressSel	PCLoad	Write	InstLoad	ImmLoad	RegWrite	RegSel	DregSel	SregSel	opSel	Befehl: LDR1 [Adr] lädt Wert von Adr (Adresse) ins Register R1
1	0	0	1	0	1	0	0	0	0	0	0	Befehl holen
2	0	0	1	0	0	1	0	0	0	0	0	Adresse holen
3	0	1	0	0	0	0	1	10	0	0	0	Wert laden

Takt	PCSel	AdressSel	PCLoad	Write	InstLoad	ImmLoad	RegWrite	RegSel	DregSel	SregSel	opSel	Befehl: STR1 [Adr] speichert Wert Register R1 nach Adr (Adresse)
1	0	0	1	0	1	0	0	0	0	0	0	Befehl holen
2	0	0	1	0	0	1	0	0	0	0	0	Adresse holen
3	0	1	0	1	0	0	1	0	0	0	0	Wert schreiben

Takt	PCSel	AdressSel	PCLoad	Write	InstLoad	ImmLoad	RegWrite	RegSel	DregSel	SregSel	opSel	Befehl: MR1R2 R2 <= R1
1	0	0	1	0	1	0	0	0	0	0	0	Befehl holen
2	0	0	0	0	0	0	1	1	1	0	0	kopieren

Takt	PCSel	AdressSel	PCLoad	Write	InstLoad	ImmLoad	RegWrite	RegSel	DregSel	SregSel	opSel	Befehl: AND R1 = R1 & R2
1	0	0	1	0	1	0	0	0	0	0	0	Befehl holen
2	0	0	0	0	0	0	1	11	0	1	0	kopieren

Takt	PCSel	AdressSel	PCLoad	Write	InstLoad	ImmLoad	RegWrite	RegSel	DregSel	SregSel	opSel	Befehl: OR $R1 = R1 \mid R2$
1	0	0	1	0	1	0	0	0	0	0	0	Befehl holen
2	0	0	0	0	0	0	1	11	0	1	1	kopieren

Takt	PCSel	AdressSel	PCLoad	Write	InstLoad	ImmLoad	RegWrite	RegSel	DregSel	SregSel	opSel	Befehl: ADD $R1 = R1 + R2$
1	0	0	1	0	1	0	0	0	0	0	0	Befehl holen
2	0	0	0	0	0	0	1	11	0	1	10	kopieren

Takt	PCSel	AdressSel	PCLoad	Write	InstLoad	ImmLoad	RegWrite	RegSel	DregSel	SregSel	opSel	Befehl: SUB $R1 = R1 + R2$
1	0	0	1	0	1	0	0	0	0	0	0	Befehl holen
2	0	0	0	0	0	0	1	11	0	1	11	kopieren

Takt	PCSel	AdressSel	PCLoad	Write	InstLoad	ImmLoad	RegWrite	RegSel	DregSel	SregSel	opSel	Befehl: JPZ [Adr] Springt an Adr wenn z-Flag null ist
------	-------	-----------	--------	-------	----------	---------	----------	--------	---------	---------	-------	--



1	0	0	1	0	1	0	0	0	0	0	0	Befehl holen
2	0	0	1	0	0	1	0	0	0	0	0	Adresse holen
3	1	0	0	1	0	0	0	0	0	0	0	wenn z-Flag gleich high → PCLoad ist 1 ansonsten 0

## 11. Program Controller

Im vorhergehenden Kapitel hab ich nun acht Maschinenbefehle vorgestellt LD, STR, MR1R2, AND, OR, ADD, SUB und JPZ und die entsprechenden Einstellungen für jeden Takt, um diese durchzuführen. Damit das nun automatisch abläuft, braucht man einen Program Controller Logik, die das für einen übernimmt. In grösseren Prozessoren ist diese programmierbar und nennt sich Mikrocode. Da es sich hier nur um wenige Befehle handelt und die Schaltung nicht komplexer sein sollte, zeige ich eine verdrahtete Version.

Bei den Befehlen gibt es zwei Kategorien: 1 Byte Befehle, AND, OR, ADD, SUB und MR1R2 und 2 Byte Befehle, LDR, STR und JPZ. Bei letzteren ist das 2. byte immer die Adresse, wo noch Information mitverarbeitet werden soll. Die 2-Byte Befehle benötigen lediglich zwei Taktzyklen und die 1-Byte Befehle drei Taktzyklen zur Abarbeit. Ein Grundbaustein des Program Controllers ist somit ein Zyklenzähler, der angibt, in welchem Zyklus die Befehlsabarbeitung sich befindet.