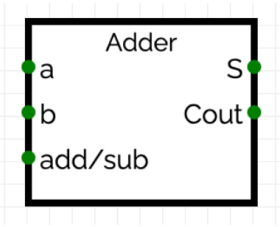
Lernziele:

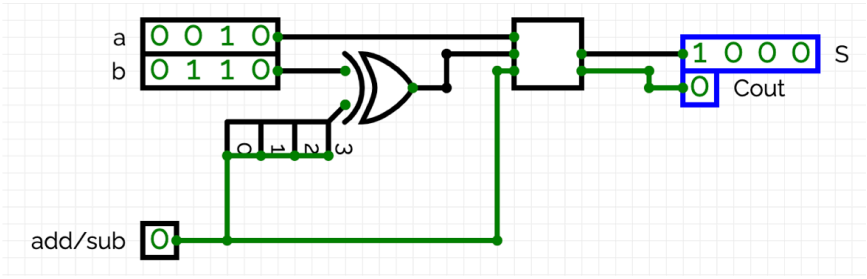
- Einführung der sogenannten Status Flags am Beispiel Addition/Subtraktion.
- Anwendung von Registern

Ausgangspunkt

Für dieses Aufgabenblatt benötigen wir einen 4-Bit-Addierer, der auch Subtraktionen durchführen kann. Der Addierer hat folgende Eigenschaften:

- Eingänge **a** und **b** sind 4 Bit breit.
- Der Eingang **add/sub** (1 Bit) wählt mit 0 die Addition und mit 1 die Subtraktion.
- Der 4-Bit-Ausgang **s** zeigt das Ergebnis der Operation.
- Der Ausgang **Cout** zeigt den Überlauf an und ist direkt mit Cout des 4-Bit-Addierers von CircuitVerse verbunden. Bei einer Subtraktion wird der Wert nicht negiert.





Schritt 1

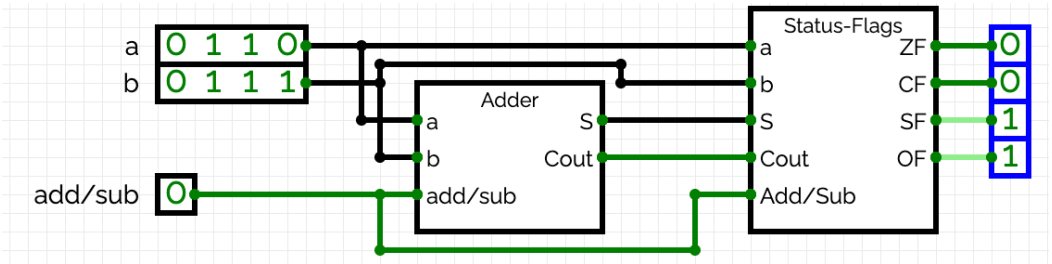
In der Vorlesung wurde gezeigt, dass ein Bitmuster $(a_3 a_2 a_1 a_0)$ entweder als Unsigned-Integer oder Signed-Integer interpretiert werden kann. Je nach Interpretation kann es bei einer Addition/Subtraktion zu einem Überlauf kommen. Um Überläufe zu erkennen, sollen folgende Status-Flags verwendet werden:

- **CF (Carry-Flag):** Wert 0 bei keinem Überlauf als Unsigned-Integer, sonst 1.
- **OF (Overflow-Flag):** Wert 0 bei keinem Überlauf als Signed-Integer, sonst 1.

Zusätzlich sind folgende Flags wichtig:

- **ZF (Zero-Flag):** Hat den Wert 1, wenn ausschließlich Null-Einträge in s vorhanden sind, sonst 0.
- **SF (Sign-Flag):** Wert entspricht dem Most-Significant-Bit von s.

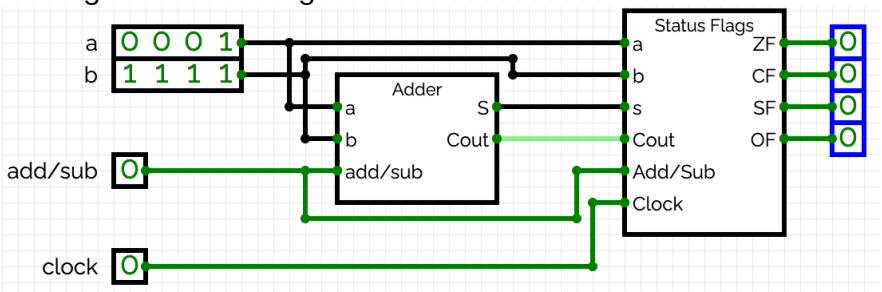
Erstellt in CircuitVerse ein Modul namens "Status Flags", das abhängig von **a**, **b**, **s**, **Cout** und **add/sub** diese Flags bestimmt. Diese Komponente kann dann wie folgt dargestellt zusammen mit dem Addierer verwendet werden:



Schritt 2

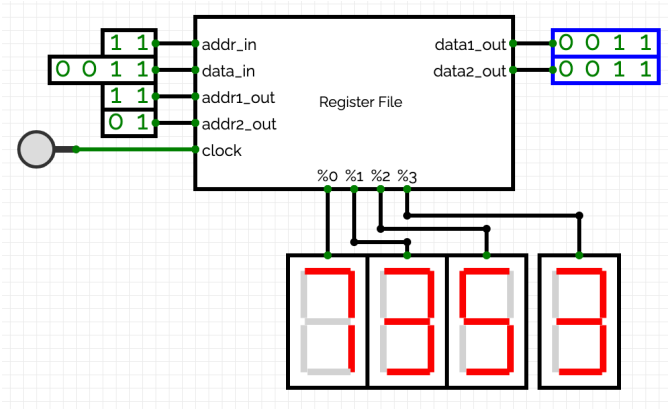
Im nächsten Schritt sollen die Status-Flags erst dann aktualisiert werden, wenn ein Clock-Signal von Null auf Eins wechselt (Aufsteigende Flanke). Ergänze dazu dein Modul "Status Flags" um einen Eingang namens **clock** und verwende D-Flipflops, um die Werte von **ZF**, **CF**, **SF** und **OF** zu speichern.

Mit folgender Schaltung kannst du das Modul testen:



Schritt 3

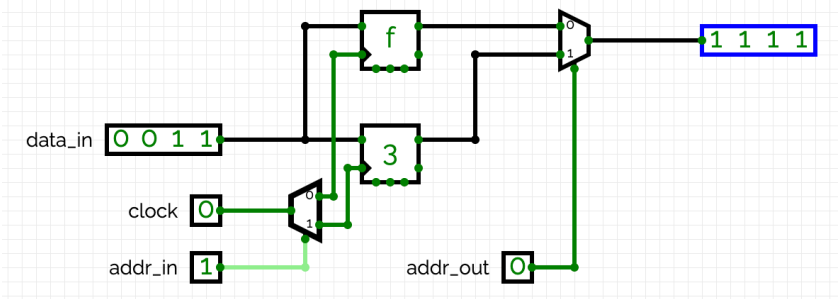
In diesem Schritt sollen D-Flipflops verwendet werden, um 4-Bit-breite Register zu realisieren. Das Modul "Register File" soll intern vier solche Register besitzen und es ermöglichen, Daten in ein Register zu schreiben oder Daten aus zwei Registern zu lesen. Ein mögliches Layout für dieses Modul ist in folgender Abbildung dargestellt:



Hier eine kurze Beschreibung, wie dieses Modul benutzt werden kann:

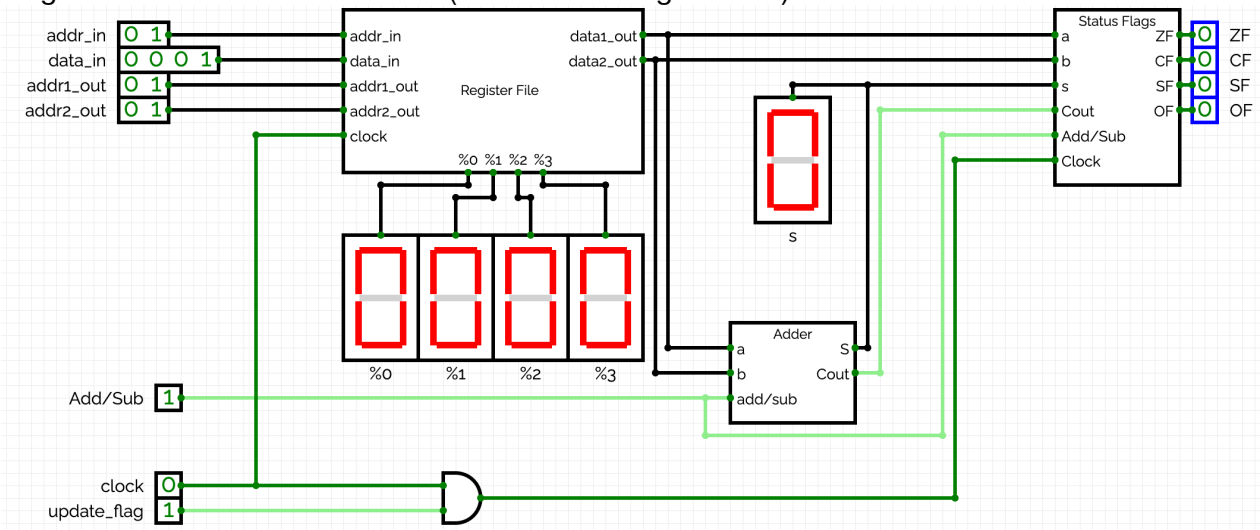
- Mit dem Eingang **addr_in** kann ein Ziel-Register (**%0**, **%1**, **%2** oder **%3**) ausgewählt werden. Bei einer aufsteigenden Flanke des Clock-Signals wird der Wert von data_in in dieses Register geschrieben.
- Mit **addr1_out** kann festgelegt werden, welcher Registerinhalt mit **data1_out** verbunden ist und somit ausgelesen werden kann.
- Mit **addr2_out** kann festgelegt werden, welcher Registerinhalt mit **data2_out** verbunden ist und somit ausgelesen werden kann.

Als Hilfestellung findest du hier eine Schaltung, die ein Register-File mit zwei 4-Bit-Registern zeigt, bei denen nur ein Register gleichzeitig ausgelesen werden kann Diese zeigt, wie prinzipiell Multiplexer und Demultiplexer in dieser Aufgabe verwendet werden können:



Schritt 4

Die folgende Schaltung zeigt, wie das Register-File zusammen mit dem Addierer und dem Status-Flag-Modul für eine einfache ALU (Arithmetical-Logical-Unit) verwendet werden kann:



Aufgaben:

- Erstelle diese Schaltung in CircuitVerse.
- Mache dich mit der Funktionsweise der Schaltung vertraut. Versuche dazu folgende Operationen durchzuführen:
  - Schreibe das Bitmuster **1111** in das Register **%0**. Die Status-Flags sollen dabei nicht verändert werden.
  - Schreibe das Bitmuster **0001** in das Register **%1**. Die Status-Flags sollen dabei nicht verändert werden.
  - Addiere den Inhalt der Register **%0** und **%1**, und aktualisiere gleichzeitig die Status-Flags.
  - Subtrahiere den Inhalt des Registers **%1** von **%0**, und aktualisiere gleichzeitig die Status-Flags.

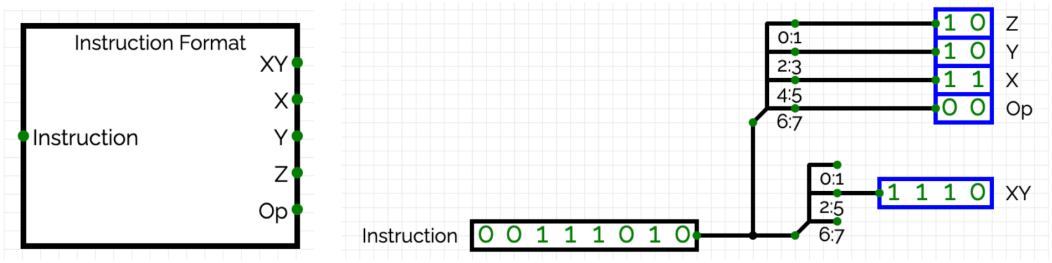
Schritt 5

Die ALU soll durch 8 Bit breite Befehle gesteuert werden. Für einen Befehl der Form $(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$ führen wir die Bezeichnungen **Op**, **X**, **Y**, **Z** und **XY** wie folgt ein:

- Mit Op (für Operation-Code) bezeichnen wir die Bits (a_7, a_6).
- Mit X die Bits (a_5, a_4), mit Y die Bits (a_3, a_2), mit Z die Bits (a_1, a_0) und
- mit XY die Bits (a_5, a_4, a_3, a_2)

a_7, a_6	a_5, a_4	a_3, a_2	a_1, a_0
Op	X	Y	Z
	XY		

Realisiert in CircuitVerse folgende Schaltung, um auf diese Bereiche des Bitmusters einfach zugreifen zu können:



Schritt 6 (Steuerung der ALU)

Im (für heute) letzten Schritt kann nun eine Schaltung erstellt werden, so dass abhängig vom Op-Code und den Operanden **X**, **Y**, **Z** und **XY** folgende Befehle zur Steuerung der ALU unterstützt werden:

Op	Operation
00	Addiert den Inhalt der Register %X und %Y und aktualisiert die Status-Flags.
01	Subtrahiert den Inhalt des Registers %Y von %X und aktualisiert die Status-Flags.
10	Schreibt das Bitmuster XY in Register %Z
11	Noch keine Verwendung (aktuell identisch mit Op = 10)

Bei einem Op-Code 00 (Addition) oder 01 (Subtraktion) ist **data_in** mit dem Ausgang **s** der ALU verbunden. Das Ergebnis **s** wird bei der nächsten aufsteigenden Flanke des Clock-Signals somit in das Ziel-Register **%Z** geschrieben. Bei einem Op-Code 10 oder Op-Code 11 ist data_in mit **XY** verbunden. Bei der nächsten aufsteigenden Flanke des Clock-Signals wird somit das Bitmuster **XY** in das Ziel-Register **%Z** geschrieben.

Die Status-Flags werden nur nach einer Addition oder Subtraktion aktualisiert.