Verslag

Titel: Project 3: ALU

Dit verslag werd opgesteld door:

- Naam: Caluwé Jonas
  Studentennummer: s0210051
  Email adres: jonas.caluwe@student.uantwerpen.be
- Naam: Van pellicom Gilles
  Studentennummer: s0211449
  Email adres: gilles.vanpelicom@student.uantwerpen.be

Aantal man-uren besteed: 30 uur

Moeilijkheidsgraad: 6 / 10 (1 is heel makkelijk, 10 is heel moeilijk)

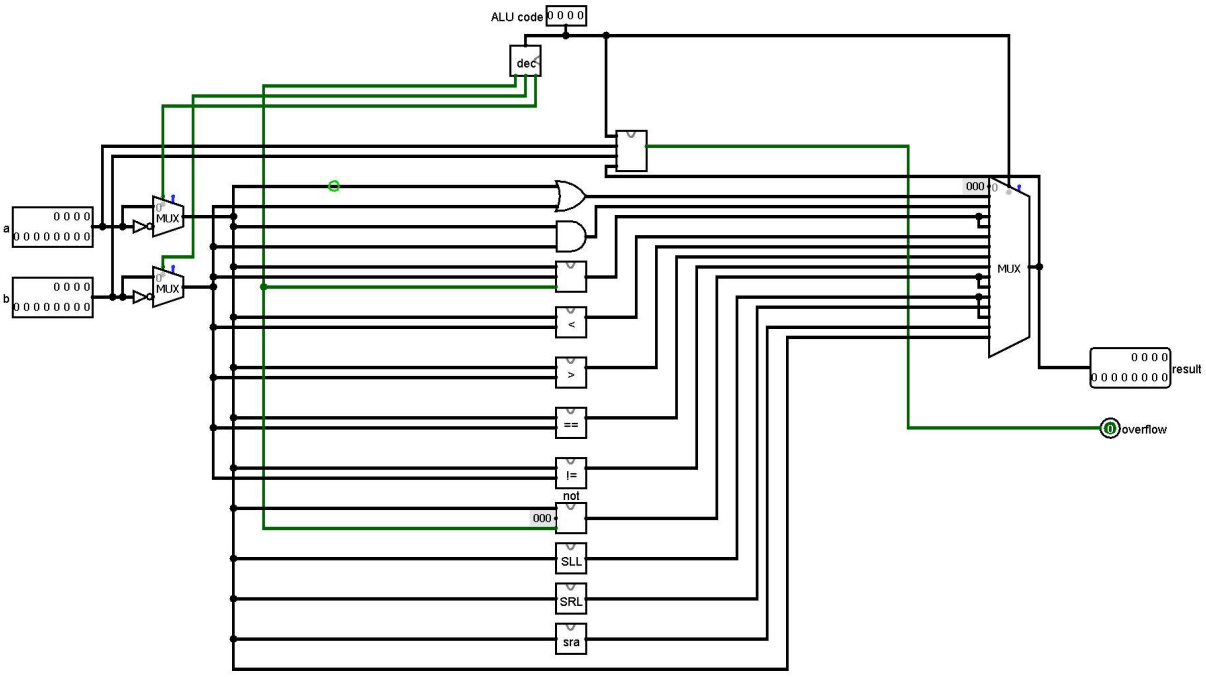
Inhoud van de oplossing

De oplossing bestaat uit de volgende bestanden (geef alle bestanden op):

- Adders.circ
- ALU_Group11.circ

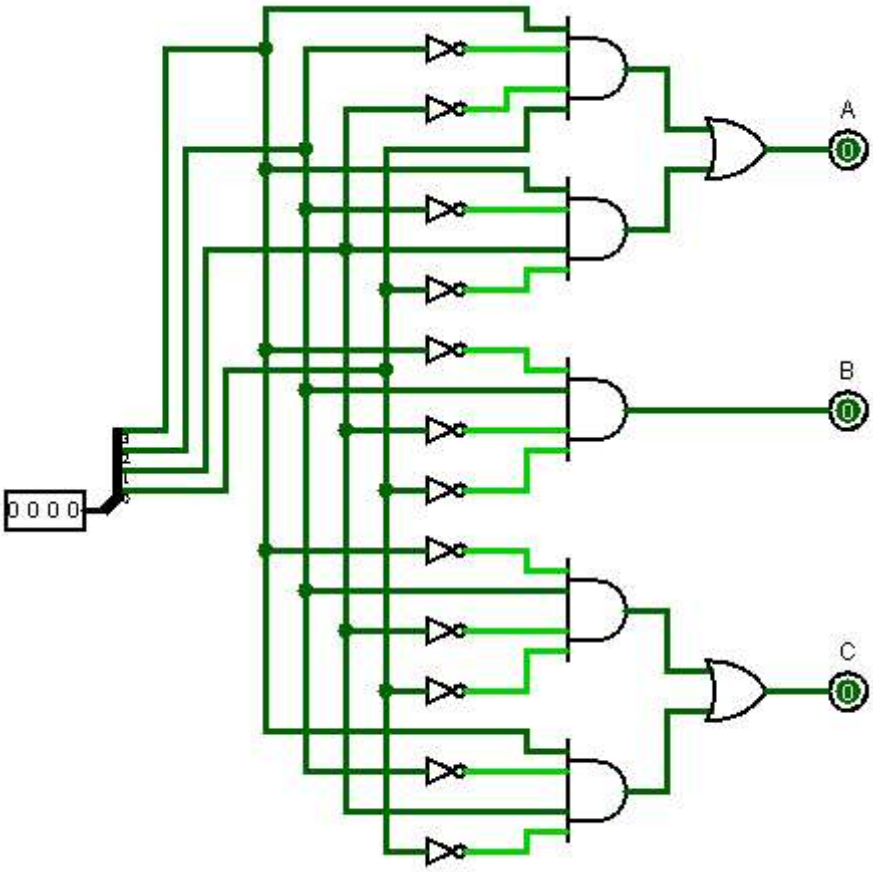
Inhoud

Overzicht Arithmemtic Logic Unit



Hierboven een overzicht van de ALU. De operaties in de ALU worden gemaakt door de rij van modules te zien in het midden van het bovenstaande diagram. Deze berekenen elk hun resultaat en door middel van de 4-bit operatie code (in wat volgt geschreven als “opcode”), zal de multiplexer de juiste input als output geven. Zo stelt opcode 0011 (3) ADD (+) voor. De multiplexer zal in dit geval de 3de input als output kiezen. Bij sommige operaties, zoals 0100 (4) SUB (-) hergebruiken we bepaalde modules, in dit geval de ADD-module. Aangezien we in 2’s complement werken, is de som van A en het complement van B plus 1 hetzelfde als A min B. Hier zal de opcode decoder (de module links onder de opcode input aan te top van de foto) beslissen of voor deze specifieke opcode het complement van A, B en/of de carry nodig zal zijn. Zo kunnen we voor SUB de datalijn van ADD tijdelijk kapen en verbinden met de input bestemd voor SUB aan de multiplexer. Aangezien er er altijd iets mis kan lopen bij bewerkingen, berekenen we de overflow. We regelen dit met de overflow module (te zien rechts onder de opcode input). Hier zal later nog worden over uitgebreid.

Operation decoder

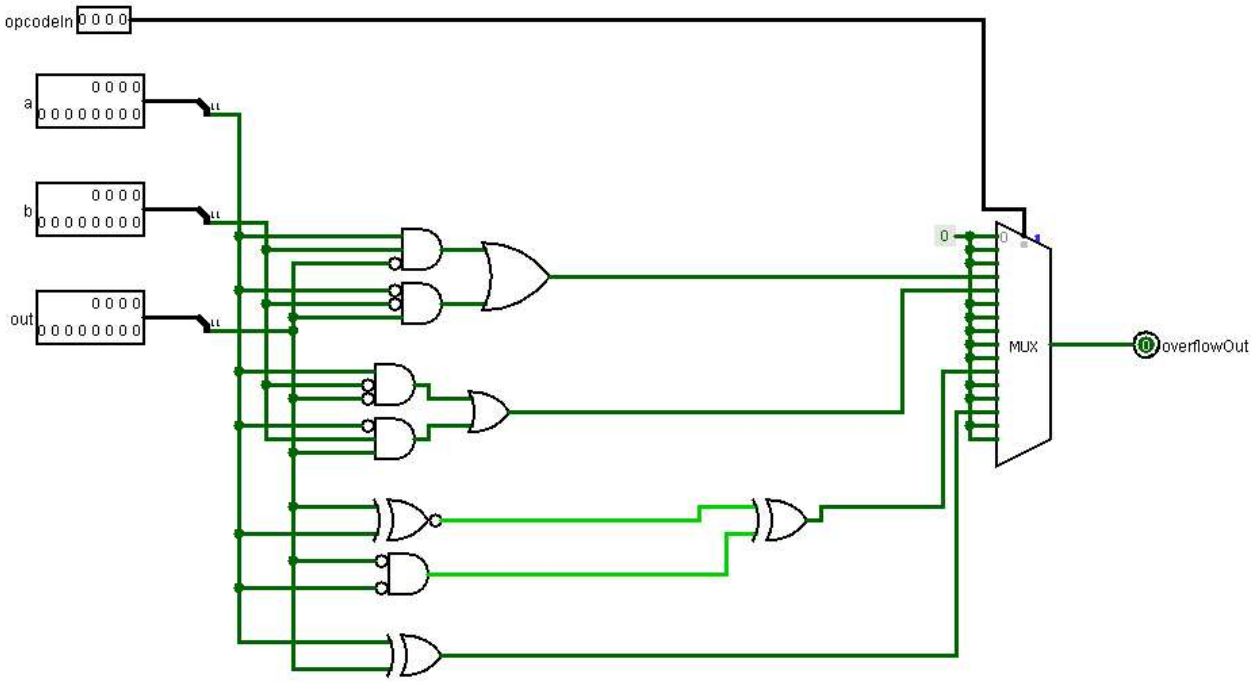


De decoder is een circuit dat als input de opcode neemt, en als output het complement van A, B en/of carry signaal activeert. Dit is een redelijk onorthodox component maar we vonden de decoder veruit de meeste elegante oplossing voor bepaalde problemen. Dit circuit is gemaakt op basis van de onderstaandeformules en waarheidstabel. Bij de waarheidstabel stelt abcd de opcode voor met a als MSB. A, B en C stellen de controle outputs voor: A, B complement en carry.

a	b	c	d	A	B	C
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	1	0	0
1	0	1	0	1	0	1
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0

$$A = a \cdot \overline{b} \cdot \overline{c} \cdot d + a \cdot \overline{b} \cdot c \cdot \overline{d}$$
$$B = \overline{a} \cdot b \cdot \overline{c} \cdot \overline{d}$$
$$C = \overline{a} \cdot b \cdot \overline{c} \cdot \overline{d} + a \cdot \overline{b} \cdot c \cdot \overline{d}$$

Overflow Detection Unit



In vorige verslag (crf. Project 2: Adders) spraken we al over een overflow unit, maar deze bevatte enkel de nodige circuits om overflow te berekenen bij de som. Echter zijn er ander regels nodig voor bijvoorbeeld subtraction, bitshiften, etc. We hebben de Overflow Detection Unit geimplementeerd als een soort van voorop gedefinieerde set van regels. De multiplexor doet hier dienst als selectiemenu. De outputs die geen overflow mochten genereren hebben we gehardwired aan een 0 ofwel false constante. Voor addition geldt: wanneer een positieve respectievelijk negatieve getallen met elkaar worden opgeteld en een negatief respectievelijk positief getal het resultaat is, dan spreken we van overflow. Voor subtraction geldt (in pseudo code): negatief – pos = pos betekend overflow of positief – neg = pos betekend overflow. Voor de invert (two's complement) spreken we van overflow wanneer een postief respectievelijk negatief getal positef respectievelijk negatief wordt, tenzij het om 0 gaat. Bij de shift left arithmatic is er overflow waneer de MSB voor en na niet hetzelfde zijn.

Operaties

Generate zero (ZERO 0000)

Generate zero is een simpele operatie. Er zal geen rekening gehouden worden met input A of B. Een 12-bit constante geeft 0 door aan de overeenkomende input van de multiplexer.

OR (OR 0001)

Bij OR worden A en B naar de inputs van een OR-gate geleid. De OR-gate ingebouwd in Logisim or'd elke bit apart dus had het geen zin meer om het wiel opnieuw uit te vinden. De output hiervan gaat naar de overeenkomende input van de multiplexer.

AND (AND 0010)

Zelfde principe als bij OR met enige aanpassing dat er gebruik gemaakt wordt van een AND-gate.

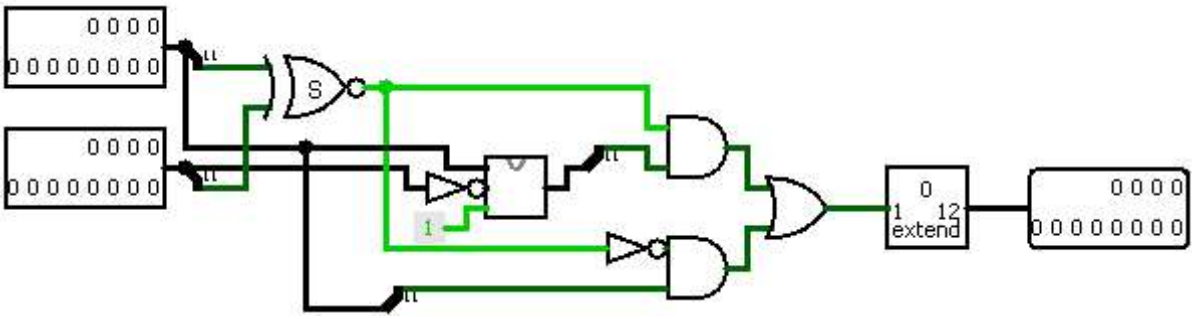
Addition (ADD 0011)

De ADD-module werd grondig besproken in verslag 2.

Subtraction (SUB 0100)

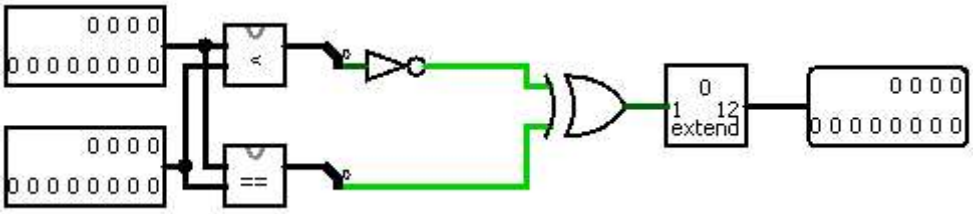
Bij subtraction hergebruiken we de ADD-module. De decoder beslist bij deze opcode dat het B-invert en carry signaal op 1 worden gezet waardoor de ADD-module zich gedraagt als een SUB module. De ADD-datalijn verbinden we ook met de SUB input op de multiplexer.

Less than (LT 0101)



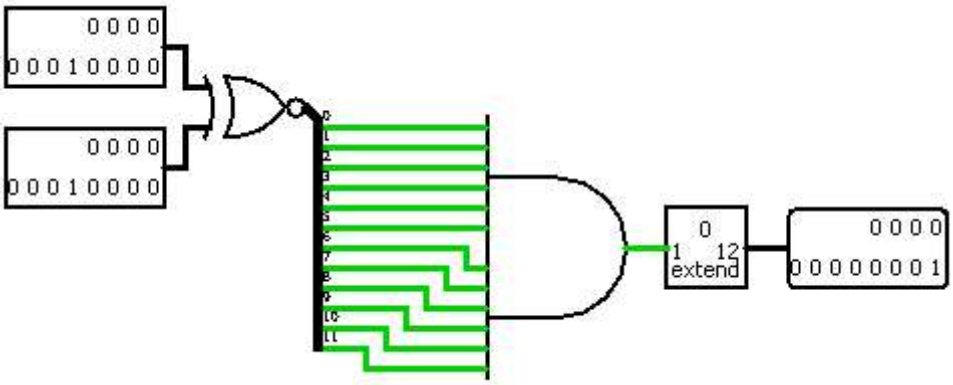
De less then lijkt een ingewikkelde operatie, maar is dit helemaal niet. Getal 1 is kleiner dan getal 2 wanneer het verschil tussen getal 1 en getal 2 negatief is. Deze berekening is echter niet nodig wanneer de getallen een verschillende sign bit hebben (in dit geval geven we sign bit van A als output). Om te “kiezen tussen deze opties, gebruiken we een selector bit. We gaan eerst na of de bits gelijk zijn, dan gaan we uit van optie 1. In het geval dat ze verschillend zijn gebruiken we optie 2.

Greater than (GT 0110)



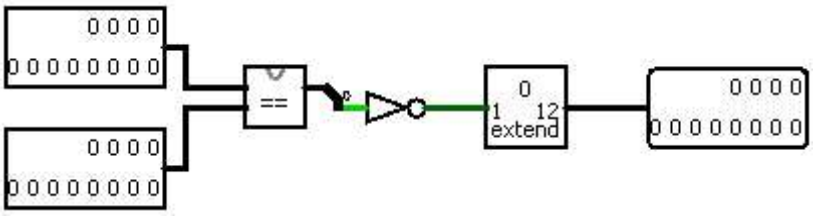
Getal 1 is groter dan getal 2 als getal 1 niet kleiner is dan getal 2, maar ook niet gelijk (zie verder). Dit lijkt misschien triviaal, maar dit is exact de implementatie die we hier hebben gekozen. Omdat true gelijk is aan: 0000 0000 0001, moeten we alleen de LSB inverteren. Vanwege de manier waarop we een resultaat geven bij verschillende sign bits, genereerden we ook een true wanneer de getallen gelijk zijn. We hadden in dit geval ook een custom circuit kunnen maken dat de sign bit van B teruggeeft, maar omdat we het circuit “eq” en “lt” al hadden geimplementeerd, zijn we gegaan voor de deze keuze.

Equals (EQ 0111)



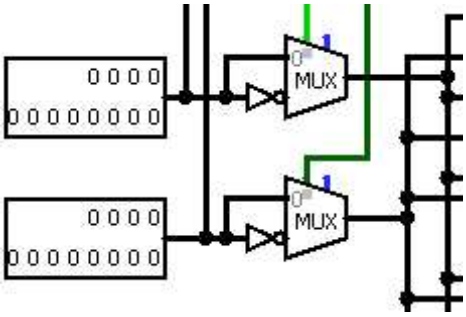
Bij equals nemen we de XNOR van A en B. De ingebouwde XNOR-gate vergelijkt elke bit apart waardoor de output zou bestaan uit 12 keer 1 als A en B hetzelfde zijn. Dit kunnen we checken door van de 12 outputs de AND te nemen. De output hiervan is wat we nodig hebben, maar is maar 1-bit. Om dit op te lossen gebruiken we een bitextender dat 11 keer 0 toevoegt.

Not equals (NEQ 1000)



Bij not equals kunnen we simpelweg de equals nemen van A en B, dan de NOT van de LSB nemen en we hebben het resultaat. Voor dezelfde reden als bij equals ook een bitextender.

NOT (NOT 1001)

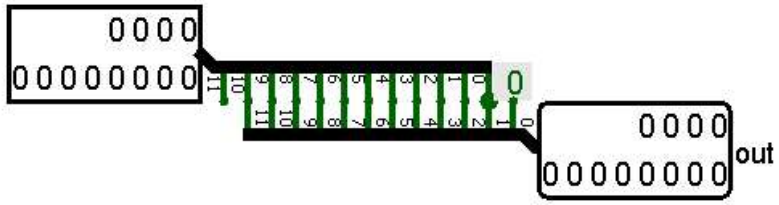


Voor deze operatie is geen module nodig. De decoder zet bij deze opcode het complement van A aan en dit signaal versturen we dan naar de output.

Inverse (INV 1010)

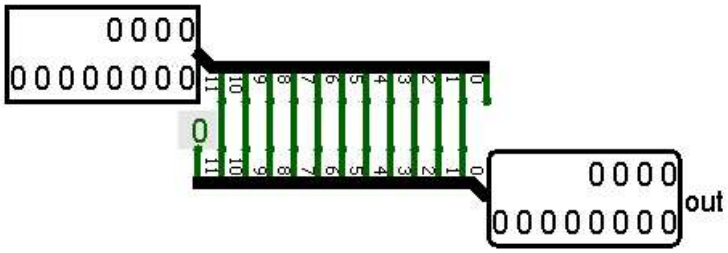
Inverse komt op hetzelfde neer als NOT, met als enig verschil dat het een wiskundige operatie is en er dus rekening gehouden moet worden met 2's complement. We beginnen met dezelfde stappen als NOT maar de decoder zet ook het carry signaal aan en we sturen het signaal naar een ADD module. Deze module voegt A toe bij een nulconstante plus één van het carry signaal.

Shift left logical (SLL 1011)



Voor SLL gebruiken we simpelweg twee splitters om de bits één positie op te schuiven. De bit die niet aangesloten is maken we dan 0 door middel van een constante.

Shift right logical (SRL 1100)

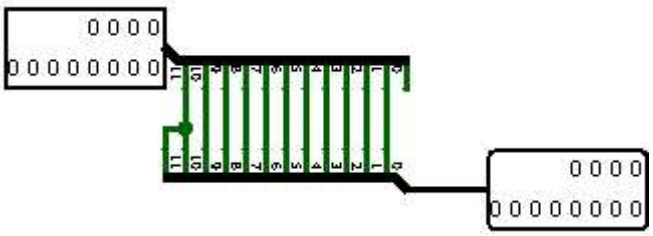


Bij SRL gebruiken we hetzelfde concept als bij SLL, met als enige verschil dat de posities naar de tegenovergestelde kant verplaatsen samen met de constant.

Shift left arithmetic (SLA 1101)

SLA is praktisch hetzelfde als SLL met als enige verschil dat er rekening moet worden gehouden met overflow. Er moet rekening gehouden worden met overflow aangezien dit een wiskundige bewerking is dat maal 2 implementeert. Aangezien de overflow module de overflow calculaties behandelt kunnen we hier simpelweg de SLL-module en datalijn hergebruiken.

Shift right arithmetic (SRA 1110)



SRA is ook bijna hetzelfde als zijn logische tegenhanger. Alleen moet hier altijd de MSB behouden worden aangezien we werken met 2’s complement. SRA implementeert gedeeld door 2. Bij delen door 2 kan de sign bit veranderen en dus is er ook geen overflow mogelijk

No operation (NOOP 1111)

Noop is veruit de simpelste operatie. We geven input a door aan de multiplexer en zo naar de output. Er worden geen bewerkingen uitgevoerd.