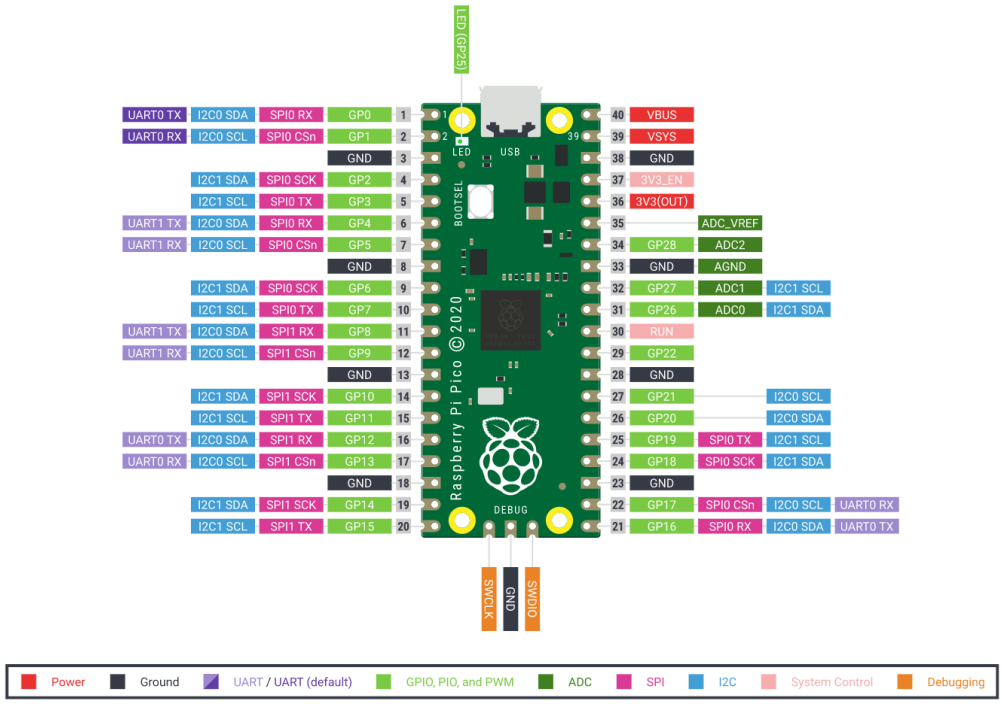
# 2 DIL logische poorten tester




## 2.1 Einddoel

Je gaat een logische poorten tester maken die aangeeft welke logische poort je hebt aangesloten.

Als je klaar bent met de opdracht lever je in It's learning in:

- De definitieve python code.
- Een mp4-filmpje met een demo van je programma.

Als dit gedaan is laat je de opdracht zien aan de docent zodat deze de opdracht kan goedkeuren.

## 2.2 Kennis

Voor deze opdracht heb je kennis nodig van microPython.

## 2.3 Benodigdheden

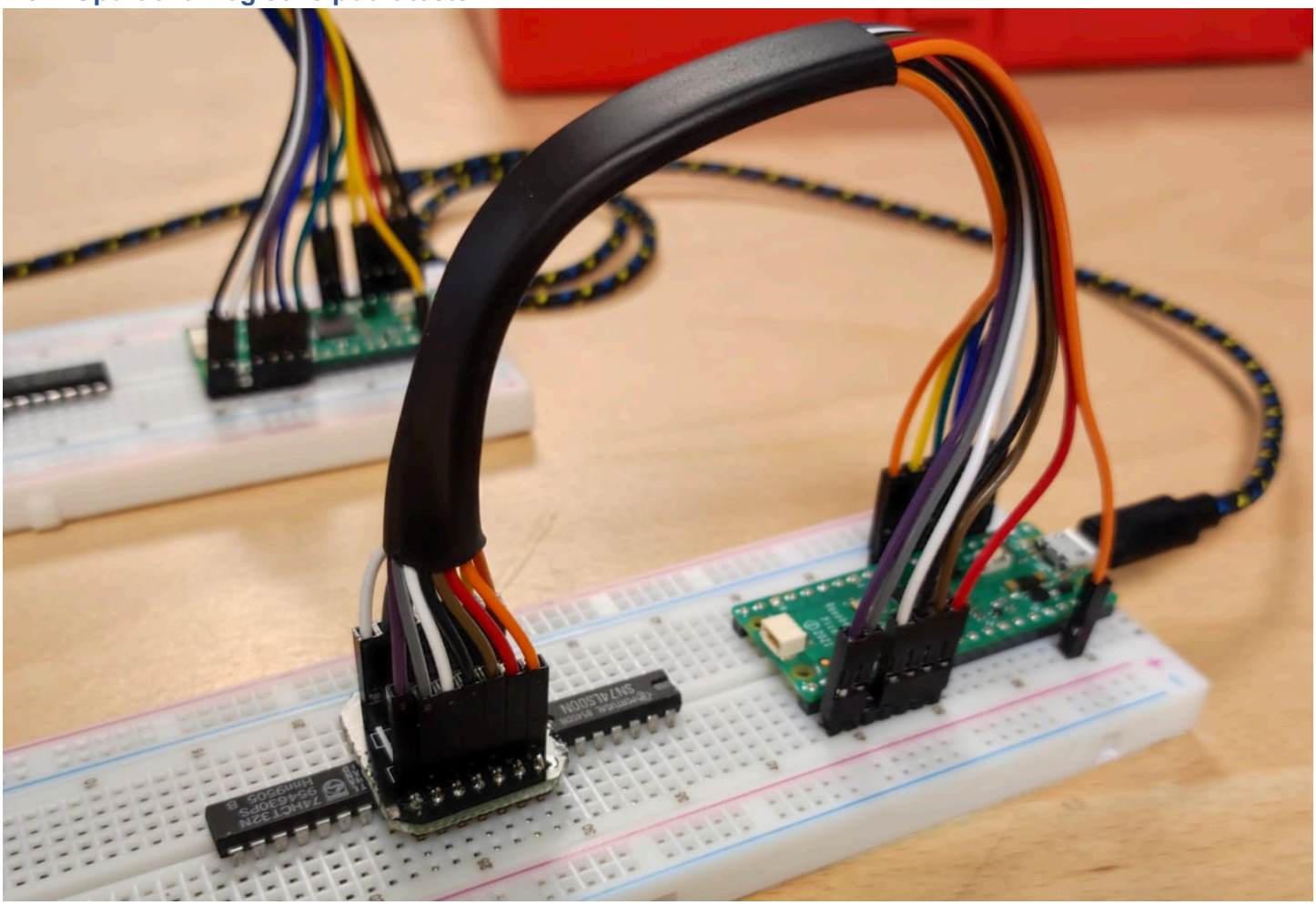
Voor deze opdracht heb je de onderstaande materialen nodig. Controleer aan het begin of al deze spullen aanwezig zijn. Bij het opruimen dien je weer te controleren of alles aanwezig is. Indien er iets defect is geraakt moet je de docent op de hoogte brengen.

- Raspberry Pi Pico – H
- UBS-usb mini kabel

## 2.4 Voorbeeldcode

De voorbeeldcode is in dit document opgenomen. Dit moet je zelf overtypen.

## 2.5 Opdracht: Logische poort tester



### 2.5.1 Wat is het probleem

We hebben een hele bak met zowel de 74HC00 (AND) als de 74HC32(OR) DIL-componenten.  Er is alleen een groot probleem. Alle nummers zijn ervan af, en we willen deze componenten wel gebruiken. We gaan daar een tester voor maken met de Raspberry pi Pico. We weten ook dat sommige componenten kapot zijn, dus we moeten alle poorten testen. Als de DIL defect is of verkeerd aangesloten is krijgen we op het scherm de tekst “Unknown” te zien. Als de DIL herkend wordt dan wordt het type van de DIL weergegeven bijvoorbeeld 74HC00 of 74HC32. De elektronische opstelling is hieronder gegeven. We gaan in de volgende paragrafen de software ontwikkelen voor deze tester.

### 2.5.2 Logische poorten

We gebruiken een *if*-vergelijking als we iets willen uitvoeren als een conditie geldig is. Bijvoorbeeld indien een waarde groter is dan 6 dan willen we iets printen.

Het kan ook zo zijn dat we meerdere condities tegelijk willen bekijken. Oftewel we willen kijken of de waarde groter is dan 6, maar het moet wel kleiner zijn dan 20. Dit kunnen we in een regel schrijven met de *AND*-operator.

```
# C
if ( x > 6 && x < 20)
{
    Serial.println("Ja");
}
# python

if x > 6 and x < 20:
    print("Ja")

```

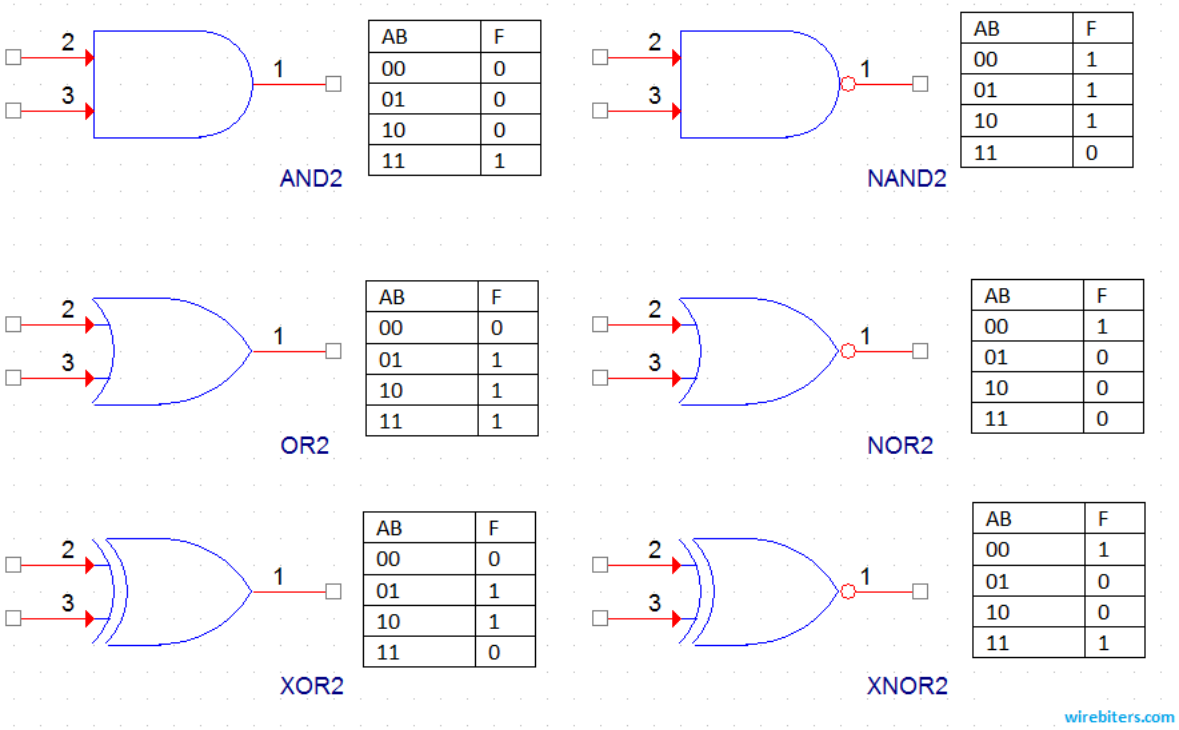
Of we willen dat indien de waarde kleiner is dan 6 of groter dan 20 de output neerzetten, dat doen we dan met de *OR*-operator.

```
# C
if ( x < 6 || x > 20)
{
    Serial.println("Ja");
}
# python

if x < 6 or x > 20:
    print("Ja")

```

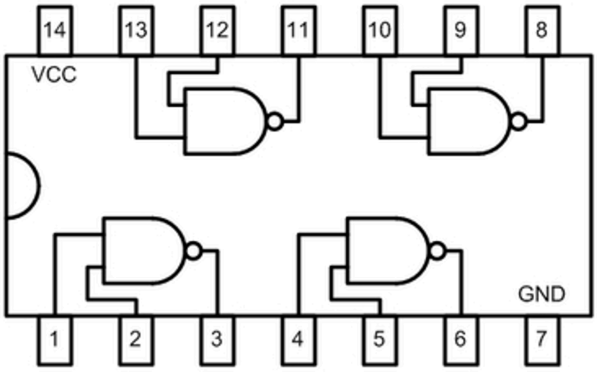
Deze logica kunnen we ook in elektronica nabouwen en hiervoor zijn er logische poorten. Dit zijn DIL-componenten met daarin meerdere van dezelfde vergelijkers. Alle onderstaande vergelijkers hebben 2 ingangen (nummer 2 en 3) en 1 uitgang (nummer 1).

Naast het symbool zie je ook de waarheidstabel. Ik bespreek de waarheidstabel van de AND.
Als de ingangen A en B beide 1 zijn is de uitgang F ook 1, in alle andere gevallen is de F de waarde 0.

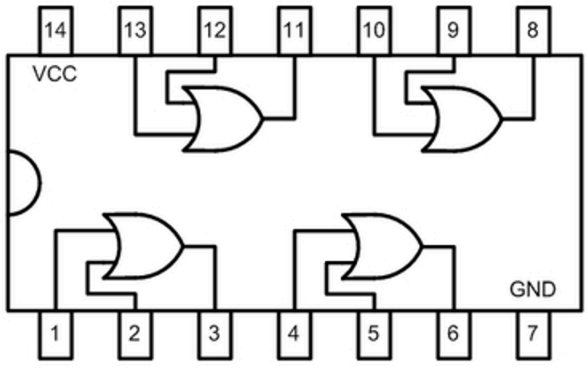
Naast de AND staat de NAND, dit is een AND maar dan alles net andersom.



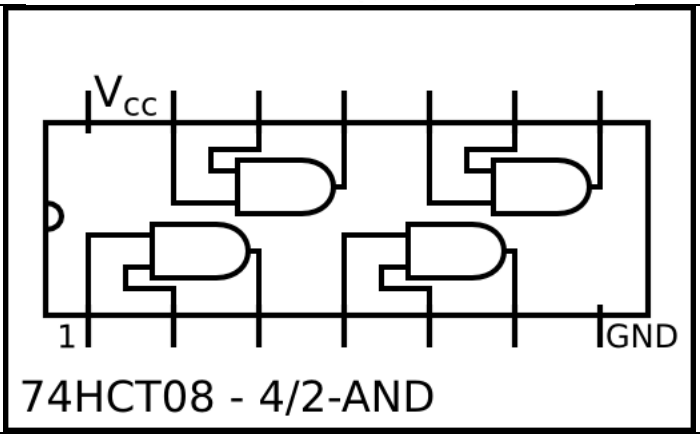

### 2.5.3 Aansluitingen van de componenten




7400 Quad 2 Input NAND




7432 Quad 2 Input OR



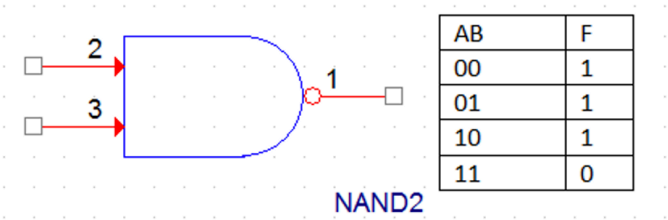

We zien op bovenstaande aansluitschema dat pin 14 de VCC heeft, en pin 7 de GND. Verder zijn alle ingangen en uitgangen van de logische poorten op dezelfde plek. We sluiten al die poorten dan aan op GP0 t/m GP11.

We beginnen klein, en gelukkig kunnen we kijken op de DIL wat voor poorten erop zitten. Zoek en sluit de 7400 aan. We zien dat dit een NAND-poort is.

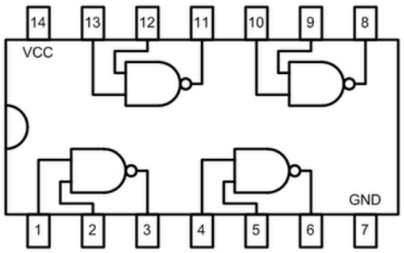
Maak een functie met de naam isNand(pinA, pinB, pinOut) die True terug geeft als de aangesloten poorten een NAND is.

```
1  from machine import Pin
2
3  CHECK       = ((0,0),(0,1),(1,0),(1,1))
4  RESULT_NAND  = (1,1,1,0)
```

In check staan alle verschillende combinaties die we op de ingangen zetten. Als we naar het eerste element in de lijst kijken zie we dat we op pinA een 0 zetten en op pinB ook een 0. Als resultaat verwachten we dan het eerste element in de constante RESULT_NAND. En zo naar de volgende.




| AB | F |
|---|---|
| 00 | 1 |
| 01 | 1 |
| 10 | 1 |
| 11 | 0 |



7400 Quad 2 Input NAND

We gaan dan de pinnen definiëren die we nodig hebben om deze test te doen. 2 pinnen die we zetten, en een derde pin die we dan uitlezen.

```
6  pin01 = Pin(1, Pin.OUT)
7  pin02 = Pin(2, Pin.OUT)
8  pin03 = Pin(3, Pin.IN)
```

```
10 #pinA and pinB are of type Pin.IN
11 #pinOut is of type Pin.OUT
12 #if the pins are set according to the CHECK we check if
13 #pinOUT is of the NAND logic
14 def isNand(pinA: Pin, pinB: Pin, pinOut: Pin) -> bool:
15     for index, check in enumerate(CHECK):
16         pinA.value(check[0])
17         pinB.value(check[1])
18         if pinOut.value() != RESULT_NAND[index]:
19             return False
20     return True
21
22 print( isNand(pin01, pin02, pin03) )
```

In bovenstaande code zitten een paar nieuwe programmeerdingen. Op regel 14 zie je: Pin staan. Dit is ervoor om te zorgen dat je alleen Pin als invoer mag geven voor de functie voor die parameter. Aan het einde staat -> bool, dit geeft aan dat het resultaat van de functie een bool zal zijn.
Op regel 15 staat enumerate. Deze functie zorgt ervoor dat zowel het element als de index van het element teruggegeven wordt uit de lijst.

```
1  CHECK = ((0,0),(0,1),(1,1),(1,0))
2
3  for pos, element in enumerate(CHECK):
4      print(pos, "A=", element[0], "B=", element[1])
```

Shell

```
>>> %Run -c $EDITOR_CONTENT

  0 A= 0 B= 0
  1 A= 0 B= 1
  2 A= 1 B= 1
  3 A= 1 B= 0
```

We gaan nu eerst proberen of bovenstaande werkt op de 7400. We gaan dit niet alleen proberen voor de DIL-pinnen 1,2 en 3, maar voor alle poorten die op de DIL. Maak de code zodat gekeken wordt of alle poorten van het type NAND zijn. Dit kan je in eerste instantie proberen door 3 draadjes steeds te verplaatsen. In tweede instantie door de probe te gebruiken die alle poorten bekijkt.

### 2.5.4 Code algemener maken

We kunnen bovenstaande code nu ook kopiëren en herschrijven voor een AND, OR, XOR, NOR en NXOR maar dan hebben we 6 functies die bijna hetzelfde doen. We gaan de code aanpassen en dan als eerste kijken of alles nog goed werkt.

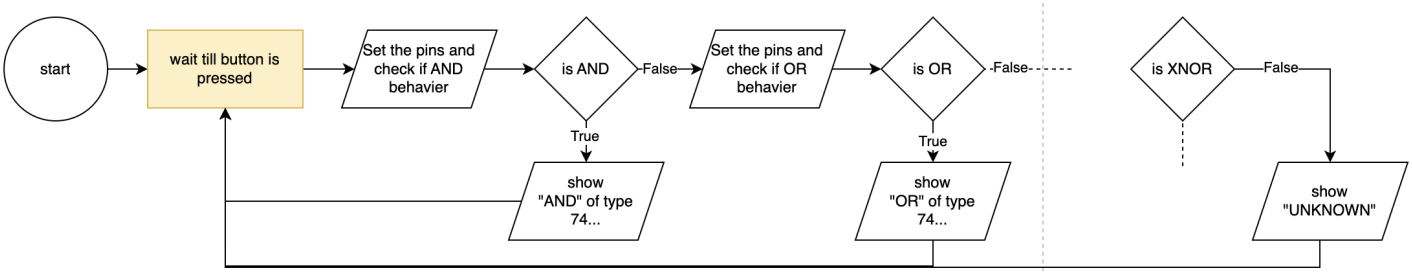
```
1  from machine import Pin
2
3  CHECK       = ((0,0),(0,1),(1,0),(1,1))
4  RESULT_NAND  = (1,1,1,0)
5  RESULT_OR    = (0,1,1,1)
6  pin01 = Pin(1, Pin.OUT)
7  pin02 = Pin(2, Pin.OUT)
8  pin03 = Pin(3, Pin.IN)
9
10 #pinA and pinB are of type Pin.IN
11 #pinOut is of type Pin.OUT
12 #if the pins are set according to the CHECK we check if
13 #pinOUT behaves like the given logic
14 def checkLogical(pinA: Pin, pinB: Pin, pinOut: Pin, logic : tuple) -> bool:
15     for index, check in enumerate(CHECK):
16         pinA.value(check[0])
17         pinB.value(check[1])
18         if pinOut.value() != logic[index]:
19             return False
20     return True
21
22 print( checkLogical(pin01, pin02, pin03, RESULT_NAND) )
23 print( checkLogical(pin01, pin02, pin03, RESULT_OR) )
```

Maak de code nu af voor de missende AND, XOR, NOR en NXOR. Deze logica is al eerder getoond in een plaatje.
Maak de code vervolgens af zodat alle poorten op de DIL gecontroleerd worden.

### 2.5.5 Programma afmaken

We gaan nu ten slotte het programma afmaken zodat deze aangeeft welke DIL we aan het testen zijn.

Maak eerst de flowchart af hieronder is al een opzet gegeven. We gaan als volgt te werk. We sluiten de poorten van de DIL aan op de Raspberry Pi Pico. We drukken vervolgens op een knop. Er wordt gekeken of de poort een AND is, als dat zo is wordt dat op het scherm getoond. Is dat niet zo dan wordt er gekeken of het een OR is… etc. etc. etc. totdat het helemaal nergens op slaat en dan wordt “Unknown” getoond. Let op dat er DIL-componenten zijn waar de ingangen en de uitgang op een andere plek zitten als in het voorbeeld. Zorg eerst dat je de hier besproken ondersteund en daarna de andere. De anderen zijn degene waarbij je tot nu toe allemaal 0-en terugkrijgt als resultaat.





### 2.5.6 Ultieme uitdaging

Als je hier bent aangekomen en je hebt alles makkelijk kunnen maken dan kan je de gemaakte code ook nog omschrijven dat er nog minder regels code zijn. Hierboven wordt steeds de waarheidstabel uitgevoerd en dan gekeken or die gelijk is aan een bepaald verwacht resultaat. Je kunt ook de poort eenmaal testen en vervolgens kijken welke van de verwachte resultaten het gemeten resultaat mee overeen komt. Kan je dit oplossen dan behoor je echt tot de hele goede slimme programmeurs. Je zult zien dat je code veel kleiner wordt.