Les circuits devront dorénavant contenir des entrées/sorties multibits et des splitters. Ainsi les squelettes des circuits `shad1` (exercice **2**, question 2) et `2comp3bits` (exercice **3**, question 3b) devraient ressembler respectivement à :

Exercice 1 On considère la formule ξ de l'exercice **5** de TD01. Déduire de sa forme normale disjonctive (calculée dans l'exercice **1** de TD02) un circuit pour ξ n'utilisant que des portes `NAND`.

Exercice 2 Les circuits de cet exercice pourront être rassemblés dans un même projet `shadok`.

1. Construire un circuit `shad1` avec une entrée `a1a0` qui permet d'afficher le chiffre Shadok correspondant via une matrice led quatre sur quatre (ou plus grande éventuellement).
2. Modifier le circuit précédent pour qu'il ait une sortie sur seize bits.
3. Construire alors un circuit `shad4` avec une entrée sur 8 bits qui permet d'afficher l'écriture Shadok correspondante via quatre matrices led.

Exercice 3 Les circuits de cet exercice pourront être rassemblés dans un même projet `2comp`.

1. Construire un circuit `nzp4` avec une entrée `a3a2a1a0` et trois sorties `n`, `z`, `p` de sorte que seule `n` (*resp.* seule `z`, *resp.* seule `p`) vaut `1` si l'entier dont l'écriture `a3a2a1a0` en complément à 2 est strictement négatif (*resp.* est nul, *resp.* est strictement positif).
2. Construire un circuit `ext4v8` avec une entrée `a3a2a1a0` et une sortie `b7b6b5b4b3b2b1b0` de sorte qu'elles soient les écritures en complément à 2 d'un même entier.

3a. En s'appuyant sur une table de vérité (à six colonnes), construire un circuit `2comp3` avec une entrée `a2a1a0` et une sortie `c2c1c0` de sorte qu'elles soient compléments à 2 l'une de l'autre.

3b. Ajouter une deuxième sortie `o` (à un seul bit) à `2comp3` signalant une retenue.

3c. En s'appuyant ou non sur une table de vérité, construire un circuit `2comp4`.

3d. Sans s'appuyer sur une table de vérité, construire des circuits `2comp8` et `2comp16`.

Exercice 4 L'outil `bc` permet de faire des calculs plus ou moins sophistiqués (voir `man bc`). Ses variables spéciales `ibase` et `obase` permettent de définir la base associée aux nombres en entrée (*input base*) et en sortie (*output base*).

```
> bc -q
ibase
??
obase
??
quit
>
```

1. Lancer `bc` comme ci-contre et vérifier la valeur par défaut de `ibase` et `obase`.
2. Utiliser `bc` pour convertir $(314)_{10}$ en base octale, puis $(413)_8$ en base décimale.
3. Expliquer l'exécution de commande comme `echo 'ibase=8; ibase=12; 431'| bc` ou bien `echo 'ibase=4; ibase=ibase+10; obase=ibase+10; 11'| bc`

Exercice 5 Écrire une fonction `addition` qui, selon l'exemple d'exécution ci-contre, décrit l'addition posée des deux entiers en argument.

```
public class AdditionPosee{
  public static void add(int a, int b){
    //...
  }
  public static void main(String[] args){
    int x=6656, y=46283;
    add(x,y);
  }
}
```

```
> java AdditionPosee
Décrivons l'addition posée de 6656 et de 46283.
La somme de 6 et 3 fait 9: on pose 9.
La somme de 5 et 8 fait 13: on pose 3 et on retient 1.
La somme de 6, de 2 et de la retenue 1 fait 9: on pose 9.
La somme de 6 et 6 fait 12: on pose 2 et on retient 1.
La somme de 0, de 4 et de la retenue 1 fait 5: on pose 5.
On obtient la somme 52939.
>
```