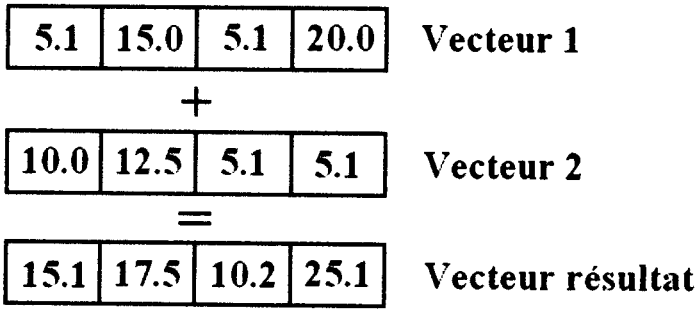
Partie 1 : Streaming SIMD Extensions, généralement abrégé **SSE**, est un jeu de 70 instructions supplémentaires pour microprocesseurs x86, apparu en 1999 sur le Pentium III. Le fonctionnement est de type SIMD. Ces instructions SIMD sont des instructions qui peuvent effectuer plusieurs opérations en parallèle, sur des données différentes.

Les opérations en question peuvent être :

- des opérations bit à bit, comme des *et*, *ou*, *not* ;
- des additions des soustractions ;
- des multiplications éventuellement des divisions ;
- ou des opérations mathématiques plus complexes.

Une instruction SIMD va traiter chacune des données du vecteur indépendamment des autres. Par exemple, une instruction d'addition SIMD va additionner ensemble les données qui sont à la même place dans deux vecteurs, et placer le résultat dans un autre vecteur, à la même place. Quand on exécute une instruction sur un vecteur, les données présentes dans ce vecteur sont traitées simultanément.



Question 1 (10 points)

Vous allez réaliser un additionneur entier non signé SIMD et générique. Vous disposez pour cela **de 4 additionneurs** 8 bits avec retenu Cin retenue en entrée et Cout retenue en sortie. Voici l'entity VHDL du composant additionneur 8 bits:

```
entity ADD8 is
          Port (
               A,B  : in  STD_LOGIC_VECTOR (7 downto 0);
               Cin  : in  STD_LOGIC;
               Cout : out STD_LOGIC;
               S    : out STD_LOGIC_VECTOR (7 downto 0)
          );
end ADD8;
```

A. Vous allez construire en VHDL un premier composant ***SSE_4add8*** qui réalise l'addition de 4 mots de 8 bits en parallèle. Il reçoit 2 mots de 32 bits en entrée et produit un mot de 32 bits en sortie.

B. Construisez ensuite un ***SSE_2add16*** avec les mêmes entrées et sorties qui réalise la somme de deux mots de 16 bits en parallèle. Ne donnez que l'architecture ici.

C. Enfin proposez un composant ***SSE_add*** avec une entrée supplémentaire ***mode*** de type `std_logic_vector(1 downto 0)` produisant

- Une addition 32 bits pour les modes 00 ou 11
- 2 additions 16 bits pour le mode 01
- 4 additions 8 bits pour le mode 10

D. Intégrez ce dernier composant ***SSE_add*** dans un IP au format HoMade en lui associant les IPcodes X010, X011 X012 et X013. Les bits de poids faibles de l'IPcode serviront pour sélectionner le mode de calcul, les bits de poids forts à sélectionner l'IP lui-même. Donnez le code VHDL.

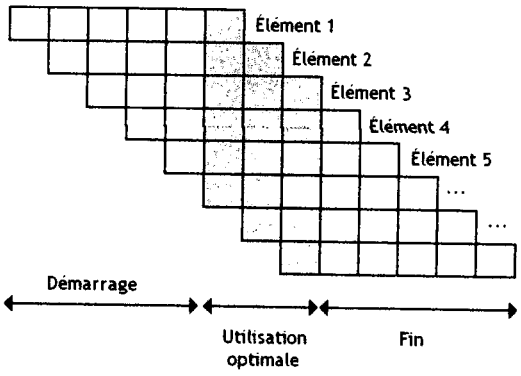
E. Ecrire un code test.fsh qui saisit successivement 4 pixels de 8 bits sur les switchs, crée un mot de 32 bits par concaténation des 4 pixels, crée un second mot de 32 bits par inversion de tous les bits et enfin réalise la somme des deux mots ainsi obtenus par tranche de 8 bits en un seul cycle.

Partie 2 : Pipeline de calcul

Contrairement aux processeurs scalaires, les processeurs vectoriels sont spécialement conçus et optimisés pour exécuter la même instruction sur un ensemble de données.

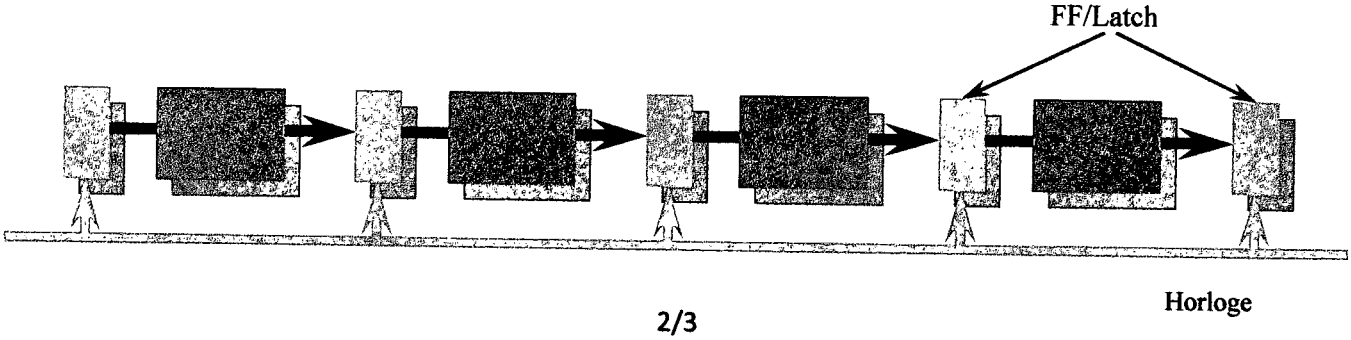
L'exécution d'une opération par l'unité de calcul est ***pipelinée***. Par pipelinée, on veut dire que l'exécution de chaque instruction sera découpée en plusieurs étapes, indépendantes les unes des autres. Cela ressemble un peu au fonctionnement d'une chaîne de montage, dans laquelle on découpe la fabrication d'un objet en plein de sous-étapes qu'on effectue les unes après les autres dans des boxes différents.(par exemple des voitures)

Au lieu d'attendre que l'exécution complète d'une opération sur une donnée soit terminée avant de passer à la suivante, on peut ainsi commencer le traitement d'une nouvelle donnée sans avoir à attendre que l'ancienne soit terminée. Cela permet ainsi d'exécuter plusieurs instructions simultanément dans notre unité de calcul. Toutes ces instructions en cours de calcul sont alors dans des états d'avancement différents.



Dans cet exemple à gauche il y a 6 étages à traverser pour une réalisation complète du calcul. Ici en gris on voit que toutes les 6 unités de traitement sont utilisées en parallèles. C'est mieux que de les laisser ne rien faire la plus part du temps !

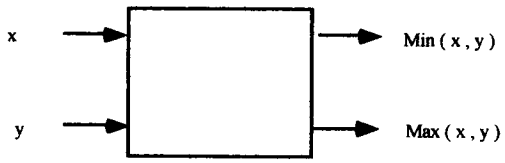
On peut construire la chaine de traitements synchrone en plaçant des registres entre chaque étape, ici 4 étapes sont dessinées...



Question 2 (10 points)

Nous allons nous intéresser aux algorithmes de tri et de fusion parallèles de mots de 8 bits: Algorithmes de Batcher. Vous proposerez au final un traitement pipeliné de cet algorithme.

La fusion de 2 listes triées X et Y permet d'obtenir une liste résultat triée Z telle que chaque élément z_i de Z appartienne à X ou Y et que chaque x_i et y_i apparaissent exactement une fois dans Z. On dispose d'un comparateur élémentaire 2 entrées et 2 sorties.



Voici l'entity VHDL de ce comparateur, on obtient les 2 mots de 8 bits : Z1 le min et Z2 le max:

```
entity FUS1 is
          Port (
               X1, Y1 : in  STD_LOGIC_VECTOR (7 downto 0);
               Z1, Z2 : out STD_LOGIC_VECTOR (7 downto 0)
          );
end FUS1;
```

A. Construisez un schéma structurel du circuit à base de comparateurs élémentaires, donnez ensuite le code VHDL d'un composant ***FUS2*** qui fusionne 2 listes triées de 2 éléments de 8 bits, X1 X2 et Y1 Y2 et produit 4 éléments triés Z1 Z2 Z3 Z4 . Vous n'avez droit qu'à trois composants ***FUS1***.

B. Proposer un code VHDL ***FUS4*** qui fusionne deux listes triées de 4 éléments X1.. X4 et Y1 .. Y4 et produit 8 mots de 8 bits triés Z1..Z8. Vous n'avez droit qu'à trois ***FUS2***. Vous pouvez aussi dessiner le schéma à base de composants élémentaires pour vous aider.

C. Proposer un composant VHDL ***SORT8*** qui trie 1 liste ***non triée*** de 8 éléments rangées dans 8 mots de 8 bits U1..U8 et produit en sortie Z1..Z8. Combien de FUS1, FUS2 et FUS4 utilisez-vous ? Vous pouvez aussi dessiner le schéma.

D. Dans cette version combinatoire on place 8 entrées sur le composant et on traverse tout un ensemble de comparateur avant de produire le résultat. En référence au mode de calcul pipeline, proposez une solution synchrone avec un découpage de votre trieur de telle sorte que chaque étape ne traverse au plus qu'un seul comparateur. Entre chaque étape il est nécessaire de placer des registres tous synchrones afin d'assurer l'avancement en même temps de toute étape vers la suivante. Dessinez un tel schéma. Combien de registres sont nécessaires et de quelle taille ? Identifiez toutes les instances nécessaires en VHDL.

E. Comment pourrait-on prendre en compte ce fonctionnement pipeline sur un processeur à pile comme HoMade ? d'où viennent les données et où vont-elles ? Qu'est-ce que vous ajouteriez à cet IP pour pouvoir le rendre utilisable et efficace dans une telle configuration.