INF3500 : Conception et réalisation de systèmes numériques

Examen final

17 décembre 2007, 13h30-16h00

Durée: 2h30.
Pondération: 40%.
Documentation: Toute permise.
Calculatrice: Programmable permise.
Directives particulières: Ordinateurs interdits, répondre à toutes les questions, la valeur de chaque question est indiquée.
Ne posez pas de questions. En cas de doute sur le sens d'une question, énoncez clairement toute supposition que vous faites.

Question 1. (8 points)

Faites la conception d'un circuit numérique combinatoire qui accepte en entrée trois nombres signés A, B, et C exprimés avec *W* bits et qui a pour sortie la médiane des trois nombres.

Dans une liste de nombres, la médiane est au milieu de la liste classée par ordre de grandeur. Par exemple, la médiane de {10, -1, 7} est 7. La médiane de {15, 12, 12} est 12.

Vous pouvez utiliser des opérations logiques et arithmétiques (dont la comparaison), des multiplexeurs, décodeurs et encodeurs, et toute porte logique. Votre circuit doit être purement combinatoire, il ne doit pas inclure d'éléments à mémoire.

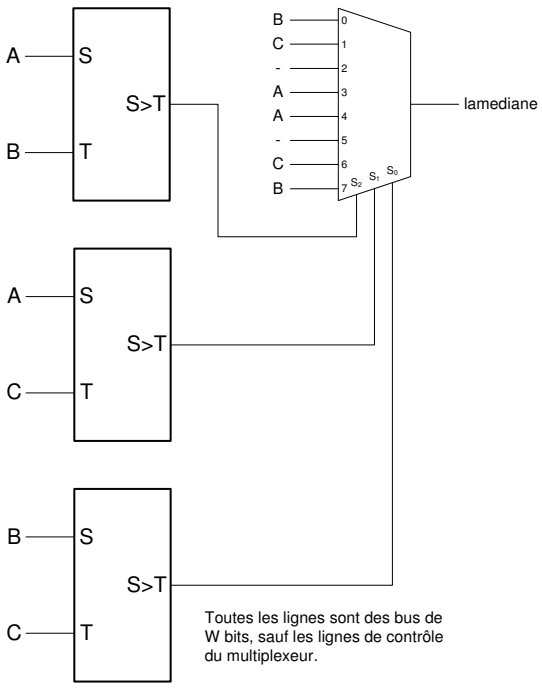
a. Donnez un diagramme de votre circuit identifiant clairement les entrées, les sorties, les blocs de traitement et leurs interconnexions.

Solution :

Une façon d'aborder le problème consiste à reconnaître qu'il y a 6 possibilités pour l'ordre de A, B, C : ABC, ACB, BAC, BCA, CAB et CBA.

Il y a plusieurs solutions possibles. Au minimum, il faut trois comparaisons, mais on peut imaginer plusieurs autres solutions avec plus de comparaisons.

Une autre solution intéressante utilise uniquement trois comparaisons mais cascade leurs résultats, ce qui réduit la complexité du multiplexeur montré ici. Cependant, le circuit montré à droite serait plus rapide.



b. Donnez une description de votre circuit en VHDL, en composant une architecture pour la déclaration d'entité suivante.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity mediane is
    generic (
        W : positive := 16 -- nombre de bits des opérandes
    );
    port (
        A, B, C : in signed(W - 1 downto 0);
        lamediane : out signed(W - 1 downto 0)
    );
end mediane;
```

Solution :

Encore une fois, il y a une multitude de solutions acceptables. Celle à droite correspond au diagramme de la partie a. On peut aussi imaginer d'utiliser un processus avec des conditions if-else.

```
architecture arch of mediane is
signal ApgB : std_logic := '0';
signal ApgC : std_logic := '0';
signal BpgC : std_logic := '0';
begin
    ApgB <= '1' when A > B else '0';
    ApgC <= '1' when A > C else '0';
    BpgC <= '1' when B > C else '0';
    with ApgB & ApgC & BpgC select
        lamediane <=
            B when "000",
            C when "001",
            A when "011",
            A when "100",
            C when "110",
            B when "111",
            (others => 'X') when others;
end arch;
```

Question 2. (8 points)

Considérez le problème de la vérification d'un circuit combinatoire qui doit rencontrer les spécifications de la question 1.

Complétez le squelette de banc d'essai suivant en VHDL en incluant une génération algorithmique de vecteurs de tests ainsi qu'une évaluation automatisée des réponses du circuit. Indiquez clairement dans quelle partie du code existant votre code doit être placé.

```
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity medianeTB is
    generic (W : INTEGER := 16 );
end medianeTB;

architecture archTB of medianeTB is
    component mediane
        generic (W : INTEGER := 16 );
    port (
        A, B, C : in SIGNED(W-1 downto 0);
        lamediane : out SIGNED(W-1 downto 0)
    );
```

```
    end component;
    signal A : SIGNED(W-1 downto 0);
    signal B : SIGNED(W-1 downto 0);
    signal C : SIGNED(W-1 downto 0);
    signal lamediane : SIGNED(W-1 downto 0);

    -- vous pouvez ajouter du code ici si nécessaire ...

begin

    UUT : mediane generic map (W => W) port map (A, B, C, lamediane);

    -- vous pouvez ajouter du code ici si nécessaire ...

end archTB;
```

Il y a plusieurs solutions possibles. La vérification par tableau de valeurs pourrait être acceptable si elle inclut suffisamment de cas possibles. Il y en a 6 de base, en plus des possibilités où deux des trois valeurs sont identiques. On imagine donc au moins une trentaine de tests. La vérification exhaustive n'est pas généralement possible, surtout pour de grandes valeurs de W. Un choix intéressant consiste à utiliser une génération aléatoire. Il faut prendre garde d'inclure des nombres négatifs.

Pour la vérification automatisée, le banc d'essai doit inclure une façon de vérifier que le résultat produit par l'unité à vérifier est bien correct. Idéalement, on devrait utiliser une description distincte de celle utilisée dans l'unité à vérifier.

```
-- dans la partie déclarative de l'architecture
function donnemediane(A, B, C: integer) return integer is
begin
    if (A >= B and B >= C) or (C >= B and B >= A) then
        return B;
    elsif (B >= A and A >= C) or (C >= A and A >= B) then
        return A;
    else
        return C;
    end if;
end donnemediane;

    -- dans l'architecture
    process
    variable seed1 : positive := 1;
    variable seed2 : positive := 2;
    variable aleatoire : real;
    variable Aint, Bint, Cint : integer := -1;
    begin
        uniform(seed1, seed2, aleatoire); aleatoire := aleatoire - 0.5;
        Aint := integer(floor(aleatoire * real(2 ** W)));
        uniform(seed1, seed2, aleatoire); aleatoire := aleatoire - 0.5;
        Bint := integer(floor(aleatoire * real(2 ** W)));
        uniform(seed1, seed2, aleatoire); aleatoire := aleatoire - 0.5;
        Cint := integer(floor(aleatoire * real(2 ** W)));
        A <= to_signed(Aint, W); B <= to_signed(Bint, W); C <= to_signed(Cint, W);
        wait for 10 ns;
        assert donnemediane(Aint, Bint, Cint) = to_integer(lamediane)
            report "erreur!" severity failure;
    end process;
```

Question 3. (8 points)

Une opération communément utilisée en traitement d'image est le calcul de la somme des différences absolues (SDA) entre deux images A et B. Cette somme est une mesure de la similarité entre les deux images.

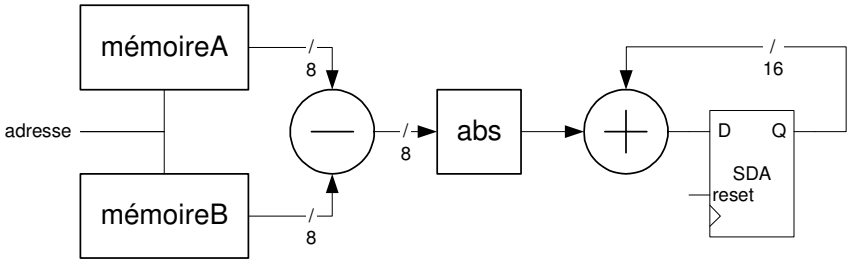
Supposons deux (petites) images de 16 × 16 pixels (256 pixels), entreposées dans deux mémoires distinctes de 256 octets chacune. Chaque octet de chaque mémoire contient l'intensité d'un seul pixel sous la forme d'un nombre non signé entre 0 et 255 inclusivement. Une version en pseudocode d'un algorithme permettant de calculer la SDA des images A et B est donné ici :

```
fini = vrai;
tanque(vrai) {
    tantque (go = 0) {
        ; // attendre
    }
    adresse = 255;
    SDA = 0;
    fini = faux;
    tant que (adresse >= 0) {
        SDA = SDA + abs(A[adresse] - B[adresse]);
        adresse = adresse - 1;
    }
    fini = vrai;
}
```

Dans le pseudocode, les signaux `go` et `fini` sont des signaux de contrôle permettant respectivement de débuter les calculs et d'indiquer que les calculs sont finis. Ils seraient utilisés pour communiquer avec le reste du système. La variable SDA pourrait être lue par un processus concurrent quand `fini` = vrai.

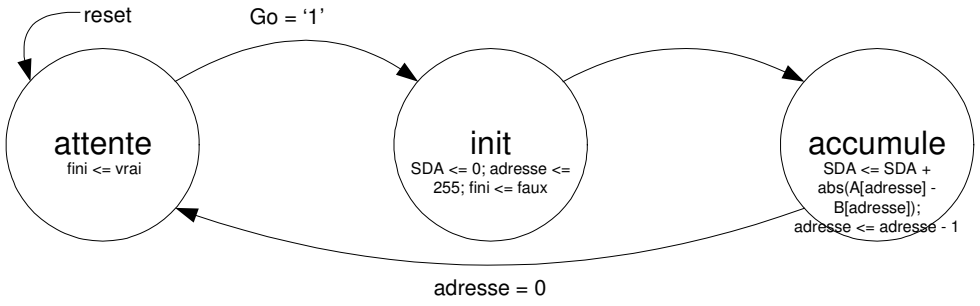
a. Donnez un diagramme montrant le chemin des données du processeur. Indiquez clairement toutes les ressources requises, leur nombre et leur largeur en bits. Indiquez clairement les signaux de contrôle.

Solution :



b. Donnez la machine à états du processeur. Identifiez bien chaque état, les actions à prendre dans chaque état, et les conditions pour les transitions entre les états.

Solution :



c. Donnez une description de votre processeur en VHDL, en composant une architecture pour la déclaration d'entité suivante.

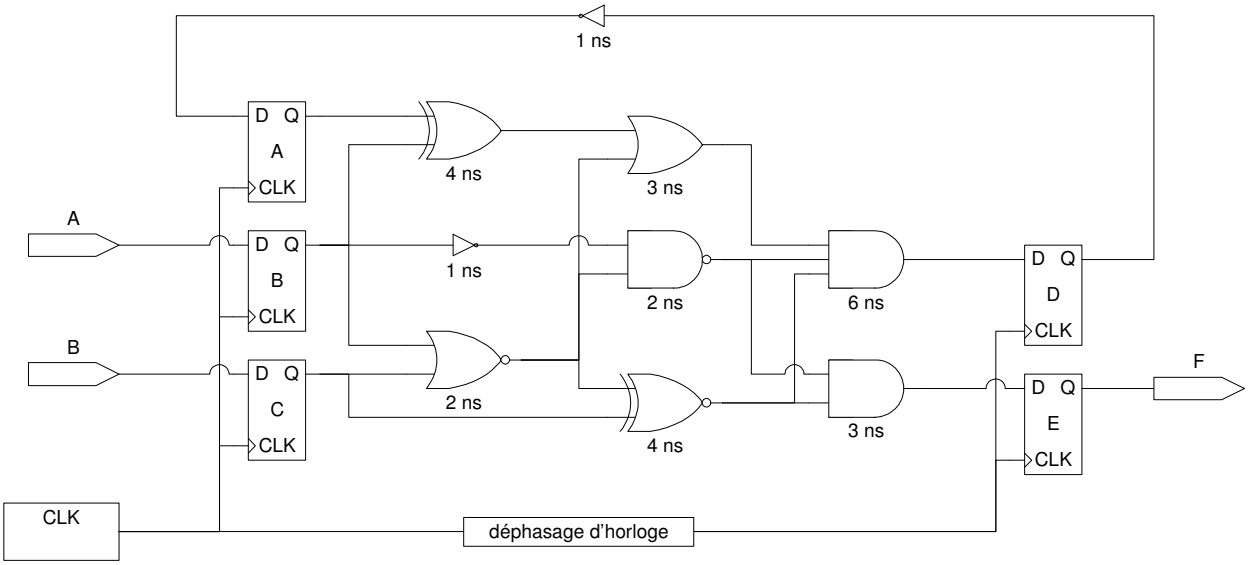
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity SDA is
    generic (
        Npixels : positive := 256 -- nombre de pixels dans chaque image
    );
    port (
        CLK, reset, go : in std_logic;
        A, B : in unsigned(7 downto 0); -- bus des données de chaque mémoire
        adresse : out unsigned(7 downto 0); -- adresse des pixels
        laSDA : out unsigned(15 downto 0);
        fini : out std_logic
    );
end SDA;
```

Solution :

```
architecture arch of SDA is
type type_etat is (attente, init, accumule);
signal etat : type_etat;
begin
    process(CLK, reset) is
    variable adresseInt : integer range 0 to Npixels - 1;
    variable SDAInt : unsigned(15 downto 0);
    begin
        if (rising_edge(CLK)) then
            if (reset = '1') then
               etat <= attente;
            else
               case etat is
                  when attente =>
                     if (go = '1') then
                           etat <= init;
                     end if;
                  when init =>
                     etat <= accumule;
                     SDAInt := (others => '0');
                     adresseInt := Npixels - 1;
                  when accumule =>
                     SDAInt := SDAInt + abs(to_integer(A) - to_integer(B));
                     if (adresseInt = 0) then
                           etat <= attente;
                     else
                           adresseInt := adresseInt - 1;
                     end if;
                  when others =>
                     etat <= attente;
               end case;
            end if;
        end if;
        adresse <= to_unsigned(adresseInt, adresse'length);
        laSDA <= SDAInt;
    end process;
    fini <= '1' when etat = attente else '0';
end arch;
```

Question 4. (8 points)

Considérez le circuit numérique suivant. Supposez que les bascules ont un temps de préparation de 1 ns, un temps de maintien de 0.5 ns et un délai de propagation de 2 ns. Les délais des autres composantes sont indiqués. Négligez les délais des interconnexions. Négligez toute contrainte de synchronisation avec les entrées A et B et la sortie F.



a. (3 points) Identifiez clairement le chemin critique et déterminez la fréquence maximale d'horloge. Supposez qu'il n'y a pas de déphasage d'horloge.

Solution : Le chemin critique va des bascules A ou B à la bascule D en passant par la porte OUX, la porte OU et la porte ET à 3 entrées. La période minimale est de (2 + 4 + 3 + 6 + 1) = 16 ns, donc la fréquence maximale est de 62.5 MHz.

b. (3 points) Ajoutez un seul niveau de pipeline. Indiquez clairement le nombre et la position des bascules ajoutées, et donnez la fréquence maximale d'horloge résultante. Supposez qu'il n'y a pas de déphasage d'horloge.

Solution : Il faut ajouter 3 bascules. Leur meilleur positionnement est immédiatement après la porte OU, la porte NON-ET et la porte ETX. Le nouveau chemin critique va des bascules A ou B à la nouvelle bascule après la porte OU, en passant par la porte OUX, et la porte OU. La période minimale est de (2 + 4 + 3 + 1) = 10 ns, donc la fréquence maximale est de 100 MHz.

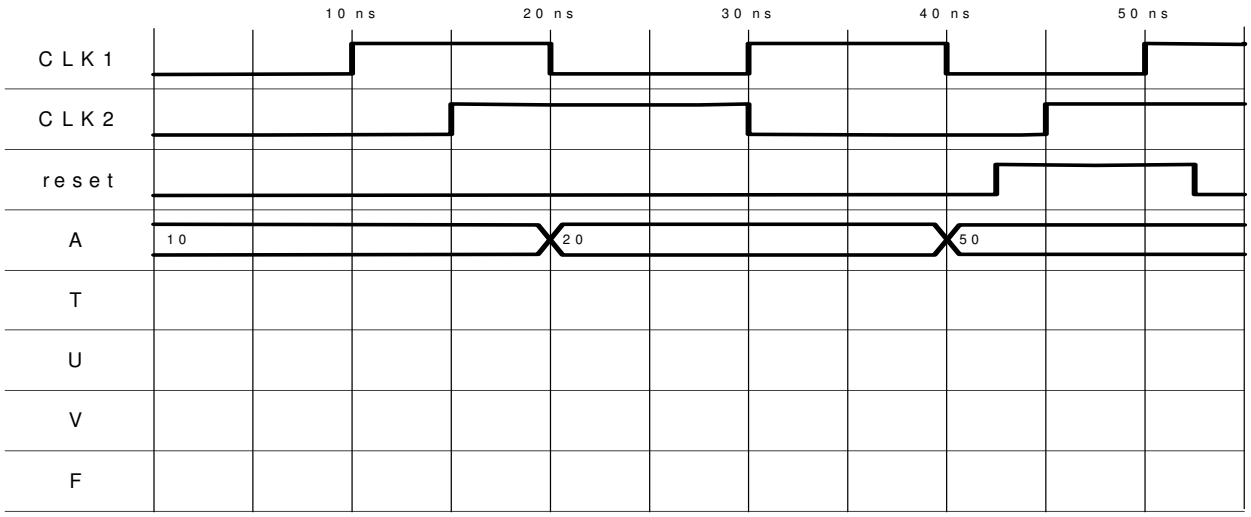
c. (2 points) Quel est l'effet d'avoir un déphasage d'horloge égal à 1 ns sur le circuit sans pipeline? Peut-on diminuer la période d'horloge? Si oui, à quelle valeur? Justifiez votre réponse.

Solution : Un déphasage d'horloge de 1 ns permettrait de réduire la période d'horloge à 15 ns, soit le délai total sur le chemin critique. Pour le chemin allant de la bascule D à la bascule A en passant par l'inverseur, il n'y a pas de problème parce que le délai est de 3 ns, ce qui est supérieur au temps de maintien de la bascule A par 2.5 ns.

Question 5. (4 points)

Considérez l'extrait de code VHDL suivant. Complétez le chronogramme pour T, U, V et F.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity VHDLEstMonAmi is
    port (
        clk1, clk2, reset : in std_logic;
        A: in integer;
        F : out integer
    );
end VHDLEstMonAmi;

architecture jaimeVHDL of VHDLEstMonAmi is
signal T, U : integer := -5;
begin

    process (clk1)
    variable V : integer := 3;
    begin
        if reset = '1' then
            T <= 0;
        elsif (rising_edge(clk1)) then
            T <= T + V;
            V := V + 2;
        end if;
    end process;

    process (clk2)
    begin
        if clk2 = '1' then
            if (reset = '1') then
                U <= 5;
            else
                U <= A + T;
                F <= U;
            end if;
        end if;
    end process;

end jaimeVHDL;
```



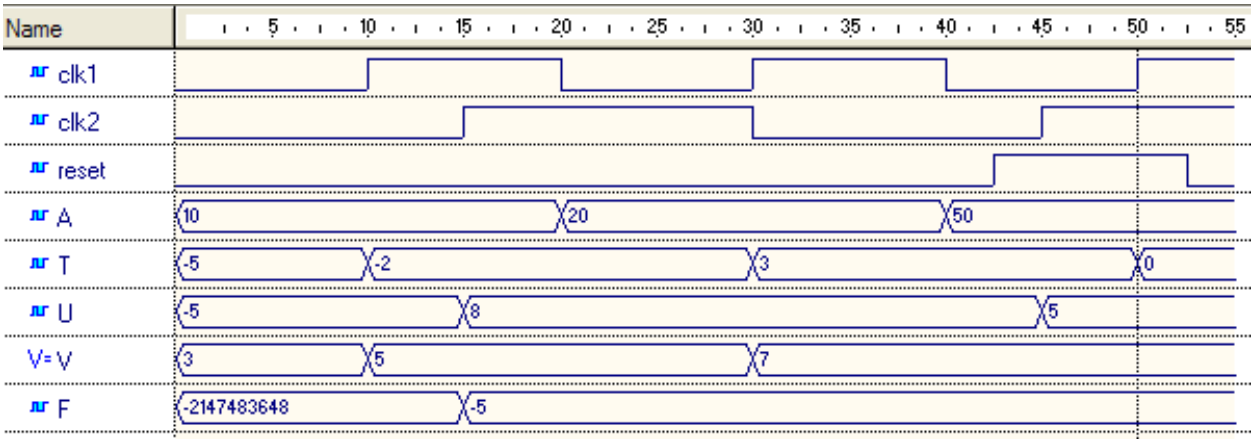
Solution :

La variable V ne dépend d'aucun autre signal. Sa valeur initiale est 3, spécifiée lors de sa déclaration. Elle est mise à jour lors de transitions positives du signal clk1, sauf lors d'un reset où elle conserve sa valeur.

Le signal T dépend de V. Sa valeur initiale est -5, spécifiée lors de sa déclaration. Il est mis à jour lors de transitions positives du signal clk1. La valeur de V utilisée pour le calcul est celle de la dernière itération du processus. Lors d'un reset, il prend la valeur 0.

Le signal U dépend de T et de A. Sa valeur initiale est -5, spécifiée lors de sa déclaration. Il est mis à jour lors de transitions positives du signal clk2, parce que clk2 est dans la liste de sensitivité du processus et qu'on a ajouté la condition (if clk2 = '1'). Attention, le résultat de la synthèse serait différent de celui de la simulation. Lors d'un reset, il prend la valeur 5.

Le signal F dépend uniquement de U. Sa valeur initiale est -2^{31}, parce que son type est integer et qu'on ne spécifie pas de valeur par défaut. Il est mis à jour lors de transitions positives du signal clk2, parce que clk2 est dans la liste de sensitivité du processus et qu'on a ajouté la condition (if clk2 = '1'). Attention, le résultat de la synthèse serait différent de celui de la simulation. La valeur assignée à F est celle de U lors de la dernière exécution du processus.



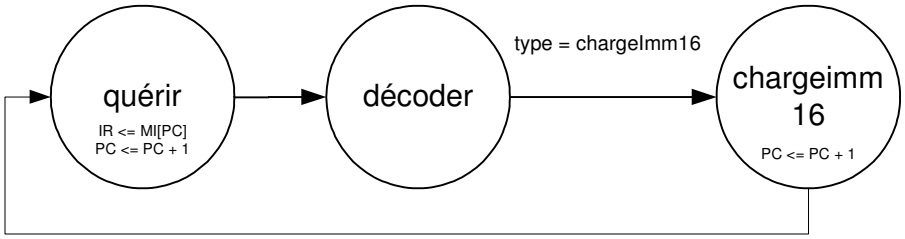
Question 6. (4 points)

Considérez le processeur à usage général décrit dans les notes de cours, sections 8.6 à 8.8 inclusivement. Ajoutez une instruction de la forme Rk ← #WXYZ, permettant de charger le registre *k* avec une valeur immédiate de 16 bits de large (WXYZ représente 4 chiffres hexadécimaux). Cette instruction nécessite deux mots de 16 bits dans la séquence des instructions : le premier représente l'instruction (chargement de valeur immédiate de 16 bits) et le deuxième contient la valeur WXYZ.

a. Donnez le diagramme d'état modifié du processeur (figure 8-17 des notes de cours) pour supporter cette nouvelle instruction. Il n'est pas nécessaire de reproduire le diagramme d'états au complet, il faut seulement indiquer le(s) nouvel(eaux) état(s), sa(leur) relation(s) avec les états existants, et les actions à prendre dans cet(ces) état(s).

Solution :

Un seul état supplémentaire suffit. En arrivant dans cet état, le PC pointe à la valeur de 16 bits qui doit être chargée dans le registre. Il est donc essentiel d'incrémenter le PC à nouveau pour le faire pointer à la prochaine instruction, qui sera chargée dans l'état quérir.



b. Donnez l'encodage de votre nouvelle instruction (tableau 8-5 des notes de cours).

Solution : une possibilité est 1010 | destination | - | -

où destination indique le registre à charger (0 à 15), et les '-' indiquent des valeurs sans importance.

c. Donnez la nouvelle ligne de code pour le signal `choixSource` (exemple 8-14 des notes de cours).

Solution :

```
with etat select choixSource <= 0 when opUAL, 1 when chargeImm16, 3 when others;
```

d. Donnez la nouvelle ligne de code pour le signal `constante` (figure 8-12 et exemple 8-11 des notes de cours)

Solution :

```
constante <= signed(memoireInstructions(PC));
```