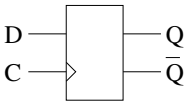
université de BORDEAUX	**ANNÉE UNIVERSITAIRE 2021 – 2022** **SESSION 1 DE PRINTEMPS** **Parcours / Étape : LSTS / L2** **Code UE : 4TIN408U** **Épreuve :** Architecture des ordinateurs **Date :** 05 mai 2022 **Heure :** 11h30 **Durée :** 1h30 Documents : non autorisés Épreuve de M./Mme : F. Pellegrini	Collège Sciences et Technologies

N.B. : - Les réponses aux questions doivent être argumentées et aussi concises que possible.
- Le barème est donné à titre indicatif.
- Inscrivez votre numéro d'anonymat sur la feuille à joindre avant de l'insérer dans votre copie.

Exercice 1 (60 points)

Les bascules D sont des circuits logiques à mémoire disposant de trois fils de contrôle : la sortie `Q` fournit en permanence l'état binaire (0 ou 1) mémorisé par le circuit, l'entrée `D` (pour « *data* ») permet de fournir au circuit la nouvelle valeur à mémoriser, celle-ci n'étant prise en compte que quand l'entrée `C` (pour « *clock* ») passe à l'état haut (front montant). On les représente schématiquement comme ci-contre.



Le but de cet exercice est de réaliser un circuit compteur/décompteur synchrone à quatre bits. Dans toutes les questions suivantes, vous pourrez utiliser les portes logiques classiques `AND`, `OR`, `NOT`, `NAND`, etc. Vous disposez également d'une horloge `H` fournissant un signal rectangulaire régulier de fréquence f.

(1.1) (10 points)

Un demi-additionneur (« HA », pour « *Half Adder* ») à un bit est un circuit qui prend en entrée deux valeurs binaires `A` et `B` sur 1 bit chacune, et renvoie un nombre entier binaire `CS` sur deux bits, où le bit de poids faible `S` représente la somme et le bit de poids fort `C` la retenue.

Donnez les fonctions logiques de `S` et `C` à partir de `A` et `B`. Câblez-les au moyen de portes logiques.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez utiliser ce demi-additionneur `HA` avec les entrées et sorties et le comportement décrits ci-dessus.

(1.2) (10 points)

En vous appuyant sur le circuit précédent, câblez un circuit incrémenteur à 4 bits, prenant en entrée un nombre entier non signé $\mathtt{A} = \mathtt{a}_3\mathtt{a}_2\mathtt{a}_1\mathtt{a}_0$ et fournissant en sortie un nombre entier non signé $\mathtt{X} = \mathtt{x}_3\mathtt{x}_2\mathtt{x}_1\mathtt{x}_0$, tel que $\mathtt{X} = (\mathtt{A} + 1) \mod 16$. Pour cela, donnez :

— la valeur de $\mathtt{x}_0$ et de la retenue correspondante $\mathtt{c}_0$, à partir de $\mathtt{a}_0$;
— la valeur de $\mathtt{x}_1$ et de la retenue correspondante $\mathtt{c}_1$, à partir de $\mathtt{a}_1$ et $\mathtt{c}_0$;
— la valeur de $\mathtt{x}_2$ et de la retenue correspondante $\mathtt{c}_2$, à partir de $\mathtt{a}_2$ et $\mathtt{c}_1$;
— la valeur de $\mathtt{x}_3$, à partir de $\mathtt{a}_3$ et $\mathtt{c}_2$.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez utiliser cet incrémenteur `INC` avec les entrées et sorties et le comportement décrits ci-dessus.

(1.3) (20 points)

Au moyen de bascules D et des circuits précédents, réalisez un compteur incrémenteur muni d'une sortie à quatre fils $\mathtt{X} = \mathtt{x}_3\mathtt{x}_2\mathtt{x}_1\mathtt{x}_0$, et tel que la valeur de `X` est incrémentée à chaque front montant de l'horloge.

N.B. : Mettez en œuvre une solution qui ne crée pas de dérive de l'horloge !

(1.4) (20 points)

Modifiez le circuit précédent pour lui ajouter un fil d'entrée `z`, tel que le compteur soit remis à zéro au prochain front montant, lorsque $\mathtt{z} = 1$.

N.B. : Mettez en œuvre une solution qui ne crée pas de dérive de l'horloge !

Exercice 2 (60 points)

On considère l'algorithme suivant, qui calcule le PGDC (« plus grand diviseur commun ») de deux nombres entiers signés positifs a et b :

```
int  a, b;
a = ...;
b = ...;
while (a != b) {
  if ((a - b) > 0)
    a = (a - b);
  else
    b = (b - a);
}
```

À la fin de l'algorithme, le résultat peut être lu dans `a` ou dans `b`, qui sont égaux.

On veut implanter cet algorithme au sein d'un circuit numérique à mots de 32 bits. Ce circuit sera synchrone, c'est-à-dire piloté par une horloge. Pour le bâtir, vous allez devoir construire et assembler les briques logiques nécessaires. Vous avez à votre disposition :

— des registres sur 32 bits, comprenant une entrée `D` sur 32 bits, une entrée `CK` (« *clock* ») sur un bit activant le registre sur front montant, et une sortie `Q` (ainsi que $\overline{\texttt{Q}}$) sur 32 bits ;

— des multiplexeurs « `MUX1` » 32×1, comprenant deux entrées $\texttt{E}_0$ et $\texttt{E}_1$ sur 32 bits, une entrée `C` sur un bit et une sortie `S` sur 32 bits ;

— des multiplexeurs « `MUX2` » 32×2, comprenant quatre entrées $\texttt{E}_{00}$, $\texttt{E}_{01}$, $\texttt{E}_{10}$ et $\texttt{E}_{11}$ sur 32 bits, deux entrées $\texttt{C}_0$ et $\texttt{C}_1$ sur un bit chacune, et une sortie `S` sur 32 bits ;

— des circuits additionneurs « `ADD` » sur 32 bits, comprenant deux entrées `A` et `B` sur 32 bits, une entrée $\texttt{C}_{\texttt{in}}$ (« *carry in* », c'est-à-dire « retenue d'entrée », dont la valeur sera ajoutée au calcul de l'addition sur les unités) sur un bit, une sortie `R` (résultat) sur 32 bits, et trois sorties `Z` (résultat nul), `S` (résultat négatif) et `C` (retenue générée en sortie), sur un bit chacune.

Chacun des circuits décrits ci-dessus peut être représenté sous la forme d'une boîte disposant des fils indiqués. Vous disposez également d'une horloge `H` fournissant un signal rectangulaire régulier de fréquence f.

(2.1) (10 points)

En notation « complément à deux », comment calcule-t-on l'opposé d'un nombre ?

(2.2) (10 points)

Déduisez-en une façon de construire un circuit soustracteur à partir du circuit additionneur décrit ci-dessus.

À partir de maintenant, et même si vous n'avez pas répondu à la question précédente, vous avez à votre disposition un circuit soustracteur « `SUB` » sur 32 bits, comprenant deux entrées `A` et `B` sur 32 bits, une sortie $\texttt{R} = \texttt{A} - \texttt{B}$ sur 32 bits, et trois sorties `Z` (résultat nul), `S` (résultat négatif) et `C` (retenue générée en sortie), sur un bit chacune.

(2.3) (20 points)

À chaque tour de boucle de l'algorithme, c'est-à-dire à chaque front montant de `CK`, on doit placer dans le registre `a` :

— soit une valeur `A` sur 32 bits fournie en entrée, si un fil de contrôle `L` (« *load* ») est égal à 1 ;

— soit la valeur $(\texttt{a} - \texttt{B})$, où `a` est l'ancienne valeur du registre, si le résultat de cette soustraction est strictement supérieur à zéro (*pour le moment, supposez que `B` est une valeur sur 32 bits fournie en entrée depuis l'extérieur du circuit*) ;

— soit la valeur préexistante du registre `a`, si le résultat de la soustraction précédente est nul (fin de l'algorithme) ou strictement inférieur à zéro (cas du « `else` »).

Construisez un circuit conforme aux spécifications ci-dessus. Explicitez les fonctions logiques que vous mettez en œuvre (table de vérité, câblage) pour le pilotage de la mise à jour de la valeur du registre. Faites preuve d'astuce, pour que ces fonctions soient simples.

N.B. : Mettez en œuvre une solution qui ne crée pas de dérive de l'horloge !

(2.4) (20 points)

En vous inspirant du circuit précédent, construisez un circuit synchrone calculant le PGDC, piloté par un fil d'horloge `CK` et un fil de contrôle `L` (« *load* »), de telle sorte que :

— si `L` = 1, les registres `a` et `b` du circuit sont chargés avec les valeurs `A` et `B` fournies en entrée ;

— si `L` = 0, le circuit modifie les valeurs de `a` et `b` à l'image de l'exécution d'un tour de boucle de l'algorithme présenté en début de problème ;

— lorsque le calcul est terminé, le fil de sortie `T` du circuit passe à 1 et les valeurs ne sont plus modifiées.

N.B. : Mettez en œuvre une solution qui ne crée pas de dérive de l'horloge !

Exercice 3 (40 points)

La feuille jointe au sujet reproduit l'essentiel du code HCL du processeur Y86 séquentiel. Au milieu de ce code se trouve le bloc « `dstE = [ ... ]` ».

(3.1) (10 points)

Quel est le rôle de ce bloc ?

(3.2) (30 points)

Expliquez précisément le sens de chacune des deux lignes contenues entre les « `[ ... ]` ». Pour chacune, dites à quel cas de figure elle correspond, et la valeur qu'elle prendra, pour chacun des différents types d'opérations (*icode*) considérés.

Exercice 4 (40 points)

La feuille jointe au sujet reproduit l'essentiel du code HCL du processeur Y86 séquentiel.

On se propose d'introduire une nouvelle instruction dans le processeur Y86 séquentiel : « `XCHGL` » (« *Exchange register values* »). Cette instruction, prenant comme argument deux noms de registres, intervertit les valeurs contenues dans les deux registres donnés comme argument.

(4.1) (20 points)

Est-il possible de mettre en œuvre cette instruction avec la structure actuelle de la banque de registres ? Si non, comment faudrait-il la modifier en interne ? Justifiez votre réponse.

(4.2) (20 points)

Écrivez directement sur la feuille de code HCL les ajouts et/ou modifications que vous proposez pour traiter cette nouvelle instruction (de `icode XCHGL`).

***NB** : n'oubliez pas d'inscrire votre numéro d'anonymat sur la feuille avant de l'insérer dans votre copie.*

— Feuille à détacher et à rendre avec la copie —

Numéro d'anonymat :

```
################ Fetch Stage     ###################################

# Does fetched instruction require a regid byte?
bool need_regids =
        icode in { RRMOVL, OPL, PUSHL, POPL, IRMOVL, RMMOVL, MRMOVL };

# Does fetched instruction require a constant word?
bool need_valC =
        icode in { IRMOVL, RMMOVL, MRMOVL, JXX, CALL };

# List of all valid instructions
bool instr_valid =
         icode in { NOP, HALT, RRMOVL, IRMOVL, RMMOVL, MRMOVL,
                    OPL, JXX, CALL, RET, PUSHL, POPL };

################ Decode Stage    ###################################

## What register should be used as the A source?
int srcA = [
        icode in { RRMOVL, RMMOVL, OPL, PUSHL } : rA;
        icode in { POPL, RET } : RESP;
        1 : RNONE; # Don't need register
];

## What register should be used as the B source?
int srcB = [
        icode in { RMMOVL, MRMOVL, OPL } : rB;
        icode in { PUSHL, POPL, CALL, RET } : RESP;
        1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
int dstE = [
        icode in { RRMOVL, IRMOVL, OPL } : rB;
        icode in { PUSHL, POPL, CALL, RET } : RESP;
        1 : RNONE;  # Don't need register
];

## What register should be used as the M destination?
int dstM = [
        icode in { MRMOVL, POPL } : rA;
        1 : RNONE;  # Don't need register
];
```

TSVP .../... ⟶

```
################ Execute Stage   ###################################

## Select input A to ALU
int aluA = [
        icode in { RRMOVL, OPL } : valA;
        icode in { IRMOVL, RMMOVL, MRMOVL } : valC;
        icode in { CALL, PUSHL } : -4;
        icode in { RET, POPL } : 4;
        # Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
        icode in { RMMOVL, MRMOVL, OPL, CALL, PUSHL, RET, POPL } : valB;
        icode in { RRMOVL, IRMOVL } : 0;
        # Other instructions don't need ALU
];

## Set the ALU function
int alufun = [
        icode in { OPL } : ifun;
        1 : ALUADD;
];

## Should the condition codes be updated?
bool set_cc = (icode == OPL);

################ Memory Stage    ###################################

## Set read control signal
bool mem_read = icode in { MRMOVL, POPL, RET };

## Set write control signal
bool mem_write = icode in { RMMOVL, PUSHL, CALL };

## Select memory address
int mem_addr = [
        icode in { RMMOVL, MRMOVL, PUSHL, CALL } : valE;
        icode in { POPL, RET } : valA;
        # Other instructions don't need address
];

## Select memory input data
int mem_data = [
        icode in { RMMOVL, PUSHL } : valA;
        (icode == CALL) : valP;
        # Default: Don't write anything
];

################ Program Counter Update ############################

## What address should instruction be fetched at
int new_pc = [
        icode == CALL : valC;
        icode == JXX && Bch : valC;
        icode == RET : valM;
        1 : valP;
];
```