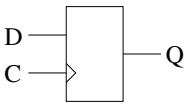
université de BORDEAUX	**ANNÉE UNIVERSITAIRE 2019 – 2020** **SESSION 1 D'AUTOMNE** **Parcours / Étape : LSTS / L2** **Code UE : 4TIN304U** **Épreuve :** Architecture des ordinateurs **Date :** 20 décembre 2019 **Heure :** 11h30 **Durée :** 1h30 Documents : non autorisés Épreuve de M./Mme : F. Pellegrini	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.

Question 1 (120 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.



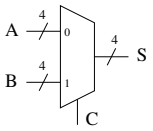
On dispose d'une horloge `H` fournissant un signal rectangulaire régulier de fréquence f.

(1.1) (20 points)

Écrivez l'équation logique d'un multiplexeur « `MUX` » à deux entrées `A` et `B` sur un bit, et un bit de contrôle `C`, qui renvoie sur sa sortie `S` la valeur `A` si `C` = 0 et `B` si `C` = 1. Câblez ce multiplexeur au moyen de portes logiques.

En vous appuyant sur celui-ci, câblez un multiplexeur « `MUX4` » à un bit de contrôle `C`, mais prenant des entrées $\texttt{A} = \texttt{a}_3\texttt{a}_2\texttt{a}_1\texttt{a}_0$ et $\texttt{B} = \texttt{b}_3\texttt{b}_2\texttt{b}_1\texttt{b}_0$ sur 4 bits chacune.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce multiplexeur 4×1 *sous la forme du circuit* `MUX4` *ci-à côté.*

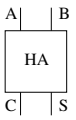


(1.2) (20 points)

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 `C S` sur deux bits, où le bit `S` représente la somme et `C` la retenue.

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

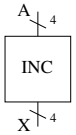
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce demi-additionneur `HA` *sous la forme du circuit ci-à côté.*



(1.3) (20 points)

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

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter cet incrémenteur « `INC` » sous la forme du circuit ci-à côté.



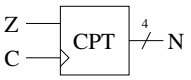
(1.4) (20 points)

En vous appuyant sur les circuits précédents, câblez un circuit compteur « `CPT` » à 4 bits, doté des caractéristiques suivantes :

— une sortie `N` sur 4 bits, fournissant le nombre entier non signé sur 4 bits actuellement mémorisé ;
— une entrée `C` sur un bit, pour l'horloge ;
— une entrée `Z` sur un bit, tel que, si elle est à 1, le compteur est remis à zéro au prochain top d'horloge. Lorsque `Z` est à 0, la valeur `N` est incrémentée à chaque top de l'horloge `C`.

(On ne veut pas de dérive de l'horloge)

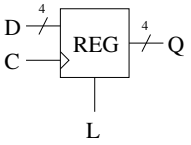
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce compteur « `CPT` » sous la forme du circuit ci-à côté.



(1.5) (10 points)

Câblez un circuit registre « `REG` » qui, au moyen de bascules `D`, permet de mémoriser, lors d'un front montant de son entrée d'horloge `C`, la valeur entière sur 4 bits placée sur son entrée `D` lorsque son entrée `L` sur un bit (« *load* ») est à 1, et conserve sa valeur `Q` sur 4 bits sinon. *(On ne veut pas de dérive de l'horloge)*

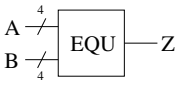
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce registre « `REG` » sous la forme du circuit ci-à côté.



(1.6) (10 points)

Câblez un circuit « `EQU` » de test d'égalité entre deux nombres entiers non signés à 4 bits. Ce circuit dispose de deux entrées `A` et `B` sur 4 bits chacun, et d'une sortie `Z` sur un bit, qui ne vaut 1 que lorsque les nombres $\mathtt{A} = \mathtt{a}_3\mathtt{a}_2\mathtt{a}_1\mathtt{a}_0$ et $\mathtt{B} = \mathtt{b}_3\mathtt{b}_2\mathtt{b}_1\mathtt{b}_0$ sont égaux.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce comparateur « `EQU` » sous la forme du circuit ci-à côté.



(1.7) (20 points)

En vous appuyant sur les circuits précédents, câblez un circuit compteur à 4 bits « `CPTM` », borné par un nombre maximum. Ce circuit est doté des caractéristiques suivantes :

— une entrée `C` sur un bit, pour l'horloge ;
— une entrée `M` sur 4 bits, permettant de fournir un nombre maximum à mémoriser dans un registre ;
— une entrée `L` sur un bit, permettant d'activer la mémorisation du nombre maximum fourni à l'entrée `M` dans ce registre ;
— une sortie `N` sur 4 bits, fournissant le nombre entier non signé sur 4 bits actuellement mémorisé dans le compteur ;
— une entrée `Z` sur un bit, tel que, si elle est à 1, le compteur est remis à zéro au prochain top d'horloge. Lorsque `Z` est à 0, la valeur `N` est incrémentée à chaque top de l'horloge `C`, jusqu'à la valeur maximum stockée dans le registre, et repasse à zéro au top suivant.

(On ne veut pas de dérive de l'horloge)

Question 2 (80 points)

(2.1) (40 points)

L'unité arithmétique et logique (UAL) est le bloc fonctionnel du processeur Y86 qui effectue, à chaque instruction, un calcul arithmétique ou logique entre les deux opérandes qui lui sont fournies sur ses entrées `A` et `B`. Le code HCL de ce circuit est fourni sur la feuille jointe, dans le bloc intitulé "*Execute stage*".

Expliquez le sens de chacune des lignes du multiplexeur `aluA`, vis-à-vis des types d'instruction concernés. Pour appuyer vos explications, vous pouvez vous aider des informations correspondantes contenues dans le paramétrage du multiplexeur `aluB`.

(2.2) (20 points)

On se propose d'introduire une nouvelle instruction dans le processeur Y86 : `cmpl` *`rA,rB`* (« *Compare Long* »). Cette instruction, qui prend deux registres en arguments, effectue l'équivalent de `subl rA,rB`, excepté qu'elle ne modifie pas la valeur de *`rB`*. Le seul effet de cette instruction est donc de modifier les indicateurs des codes de conditions (notamment *Zero Flag* et *Sign Flag*).

Ecrivez directement sur la feuille HCL jointe les ajouts et modifications que vous proposez pour traiter cette nouvelle instruction (de code `CMPL`, que l'on suppose déjà créé). *N.B. : N'oubliez pas d'inscrire votre numéro d'anonymat sur la feuille avant de l'insérer dans votre copie !*

(2.3) (20 points)

On se propose d'introduire deux autres instructions nouvelles dans le processeur Y86 : `incl` *`rA`* (« *Increment Long* ») et `decl` *`rA`* (« *Decrement Long* »). Ces instructions incrémentent et décrémentent le registre qui est leur seul argument. Ecrivez directement sur la feuille HCL jointe les ajouts que vous proposez pour traiter ces nouvelles instructions (de codes `INCL` et `DECL`).

— 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 } ||
        ((icode == OPL) && (rA == RNONE));

# 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 } || ((icode == OPL) && (rA != RNONE)) : valA;
        icode in { IRMOVL, RMMOVL, MRMOVL, OPL } : 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;
];
```