université de BORDEAUX	**ANNÉE UNIVERSITAIRE 2020 – 2021** **SESSION 1 D'AUTOMNE** **Parcours / Étape : LSTS / L2** **Code UE : 4TIN304U** **Épreuve :** Architecture des ordinateurs **Date :** 06 janvier 2020 **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 (110 points)

Le but de cet exercice est de réaliser, pas à pas, un circuit synchrone calculant la puissance $\texttt{N}^{\text{ième}}$ d'un nombre entier `X`, où `N` est un entier positif ou nul. Il vous sera d'abord demandé d'en construire une version non optimisée, puis une version optimisée.

Pour cela, vous avez à votre disposition tous les blocs logiques vus en cours : comparateurs, additionneurs, multiplicateurs, décodeurs, multiplexeurs, bascules D, etc. Un signal d'horloge rectangulaire `H` vous est également fourni.

Le fonctionnement de ce circuit sera le suivant :

— Lorsque le fil `L` (comme « *load* ») est mis à `1`, les valeurs d'entrée `X` et `N`, sur n bits chacune, sont mémorisées par le circuit au prochain front montant d'horloge (qu'on appellera pour simplifier « *top d'horloge* »).

— À chaque top d'horloge, le calcul progresse selon le câblage du circuit que vous allez réaliser. Le résultat intermédiaire `R`, sur `n` bits lui aussi, est toujours visible en sortie. Lorsque le circuit aura terminé son calcul, `R` contiendra le résultat final.

— Lorsque le calcul est terminé, un signal `F` passe à `1`, indiquant que le résultat valide peut être lu sur les fils `R` : $\texttt{R} = \texttt{X}^{\texttt{N}}$.

(1.1) (10 points)

Il est utile, sur les bascules D, de disposer d'une entrée `E` (comme « *enable* »), qui permet d'activer ou non la mémorisation : lors du front montant d'horloge, la mémorisation depuis l'entrée `D` ne se fait que si l'entrée `E` est à `1`. Si elle est à `0`, la valeur mémorisée reste inchangée.

Dessinez le circuit permettant d'obtenir ce comportement à partir d'une bascule D standard.

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

À partir de maintenant, et même si vous n'avez pas répondu à la question, vous pouvez utiliser dans la suite cette bascule « DE » dotée des entrées `D`, `E` et `Ck`, et de la sortie `Q`.

(1.2) (10 points)

À partir du circuit précédent, créez un registre sur n bits, qui fournit en permanence une valeur mémorisée, et mémorise une nouvelle valeur au prochain top d'horloge lorsque son signal `L` est à `1`.

À partir de maintenant, et même si vous n'avez pas répondu à la question, vous pouvez utiliser dans la suite ce registre « Reg » doté des entrées `D`, sur n bits, et `E` et `Ck`, sur un bit chacune, et de la sortie `Q` sur n bits.

(1.3) (10 points)

Créez un circuit qui, sur son fil de sortie `Z`, renvoie `1` si sa valeur d'entrée D sur n bits vaut zéro, et renvoie `0` sinon.

À partir de maintenant, et même si vous n'avez pas répondu à la question, vous pouvez utiliser dans la suite ce circuit « =Z ? » doté d'une entrée `D`, sur n bits et de la sortie `Z` sur 1 bit.

La première version, non optimisée, du circuit que vous allez réaliser met en œuvre l'algorithme suivant :

```
R ← 1
while (N > 0) do
    R ← R × X
    N ← N − 1
end while
```

(1.4) (20 points)

Créez un circuit décompteur doté des propriétés suivantes :

— Un fil de sortie `S`, sur 1 bit, est à `1` lorsque le compteur vaut `0`.

— Lorsque le fil d'entrée `L` est à `1`, le compteur est chargé, au prochain top d'horloge, avec une valeur d'entrée `D` sur n bits.

— À chaque top d'horloge, le contenu du compteur, visible sur sa sortie `Q`, est décrémenté de `1`, jusqu'à ce que le compteur atteigne la valeur `0`.

— Lorsque la valeur `0` est atteinte, le compteur reste bloqué à zéro, à chaque top d'horloge (et `S` reste donc à `1`), jusqu'à ce qu'une nouvelle valeur non nulle soit chargée dans le compteur.

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

À partir de maintenant, et même si vous n'avez pas répondu à la question, vous pouvez utiliser dans la suite ce circuit « CPT » doté des entrées et sorties décrites ci-dessus.

(1.5) (20 points)

En reprenant les éléments pertinents des circuits ci-dessus, réalisez un circuit qui calcule $\mathtt{X}^{\mathtt{N}}$, selon la méthode de l'algorithme ci-dessus.

L'algorithme proposé ci-dessus est très lent, puisqu'il calcule en `N` cycles, soit au plus 2^n cycles ! On propose donc un algorithme optimisé, décrit ci-dessous :

```
R ← 1
while (N > 0) do
    if (N est impair) then
        R ← R × X
        N ← N − 1
    end if
    X ← X²
    N ← N/2
end while
```

(1.6) (5 points)

Comment savoir simplement si une valeur entière est impaire ?

(1.7) (5 points)

Comment peut-on simplement effectuer une division par 2 d'un nombre entier positif ?

(1.8) (30 points)

En reprenant les éléments pertinents des circuits vus précédemment, réalisez un circuit qui calcule $\mathtt{X}^{\mathtt{N}}$, selon la méthode optimisée de l'algorithme ci-dessus.

Exercice 2 (40 points)

La feuille jointe au sujet reproduit l'essentiel du code HCL du processeur Y86 séquentiel. À la toute fin de ce code se trouve le bloc « `new_pc = [ ... ]` ».

(2.1) (10 points)

Quel est le rôle de ce bloc ?

(2.2) (30 points)

Expliquez précisément le sens de chacune des quatre lignes contenues entre les « `[ ... ]` ». Pour chacune, dites à quel cas de figure elle correspond, et le sens de la valeur qu'elle renvoie.

Exercice 3 (50 points)

(3.1) (10 points)

Examinez le programme Y86 ci-contre. Que fait-il ?

```
      .pos  0
      irmovl t,%edi
      mrmovl size,%ecx
      xorl   %eax,%eax
loop: rmmovl %eax,(%edi)
      iaddl  4,%edi
      isubl  1,%ecx
      jne    loop
      halt
      .pos  100
size: .long  2
t:
```

(3.2) (20 points)

On se propose d'introduire une nouvelle instruction dans le processeur Y86 : « `STOSL` » (« *Store String Long* »). Cette instruction (sans argument) effectue l'équivalent de `rmmovl %eax,(%edi)`, tout en ajoutant 4 au registre `%edi`, la valeur de `%edi` utilisée pour l'accès mémoire du `rmmovl` étant celle avant incrémentation.

La feuille jointe au sujet reproduit l'essentiel du code HCL du processeur Y86. Ecrivez directement sur cette feuille les ajouts que vous proposez pour traiter cette nouvelle instruction (de code `STOSL`).

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

(3.3) (20 points)

Pour continuer à réduire le nombre d'instructions au sein de la boucle, on peut s'inspirer des processeurs X86 et envisager d'ajouter une instruction « `loop` *addr* » à notre processeur. Cette instruction décrémente le registre `%ecx`, puis réalise un saut à l'adresse *addr* si la nouvelle valeur d'`%ecx` est différente de zéro.

Comme pour l'instruction `STOSL`, ajoutez le code `LOOP` partout où c'est nécessaire sur la feuille jointe afin de gérer correctement le fonctionnement de cette instruction.

— 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;
];
```