université de BORDEAUX	**ANNÉE UNIVERSITAIRE 2023 – 2024** **SESSION 1 DE PRINTEMPS** **Parcours / Étape : LSTS / L2** **Code UE : 4TIN408U** **Épreuve :** Architecture des ordinateurs **Date :** 29 avril 2024 **Heure :** 09h00 **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 (100 points)

On considère l'algorithme suivant, qui réalise la multiplication de deux nombres entiers non signés a et b :

```
1   unsigned int  a, b, p;
2   a = ...;
3   b = ...;
4   p = 0;
5   while (a != 0) {
6     if ((a % 2) == 1)
7       p += b;
8     a = a >> 1;       /* Décalage de bits à droite une fois */
9     b = b << 1;       /* Décalage de bits à gauche une fois */
10  }
```

(1.1) (5 points)

Sur votre copie, exécutez « à la main » cet algorithme, avec les valeurs initiales `a` $= 5$ et `b` $= 6$. Pour ce faire, représentez dans un tableau l'évolution des valeurs des variables `a`, `b` et `p` à l'entrée dans la boucle et chaque fois que l'une des valeurs est modifiée au cours des tours de boucle, en indiquant les numéros des lignes concernées, jusqu'à la fin de celle-ci.

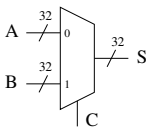
On veut implanter cet algorithme au sein d'un circuit numérique synchrone, c'est-à-dire piloté par une horloge, susceptible d'être implanté au sein d'un processeur à mots de 32 bits. Pour cela, vous allez devoir construire et assembler les briques logiques nécessaires. Vous pourrez utiliser les portes logiques vues en cours, y compris, si besoin, celles à plus de deux entrées.

(1.2) (10 points)

Écrivez l'équation logique d'un multiplexeur 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` $= 1$. Dessinez le schéma de câblage de ce multiplexeur, au moyen de portes logiques.

En réutilisant le circuit précédent, concevez et dessinez le schéma de câblage d'un multiplexeur à un bit de contrôle `C`, mais prenant des entrées `A` et `B` sur 32 bits.

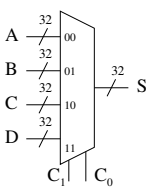
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter un multiplexeur 32×1 *sous la forme du circuit ci-à côté, comprenant deux entrées `A` et `B` sur* 32 *bits, une entrée `C` sur un bit et une sortie `S` sur* 32 *bits.*



(1.3) (5 points)

En vous appuyant sur le circuit précédent, écrivez l'équation logique et dessinez le schéma de câblage d'un circuit multiplexeur à quatre entrées `A`, `B`, `C` et `D` et deux bits de contrôle C_1C_0 (C_1 étant le bit de poids fort).

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter un multiplexeur 32×2 *sous la forme du circuit ci-à côté, comprenant quatre entrées `A`, `B`, `C` et `D` sur* 32 *bits, deux entrées* C_0 *et* C_1 *sur un bit chacune et une sortie `S` sur* 32 *bits.*

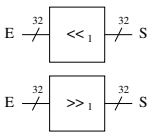


(1.4) (10 points)

Dessinez le câblage d'un circuit décaleur non signé d'un bit vers la droite, qui prend en entrée un mot de 32 bits $\texttt{E} = \texttt{e}_{31}\texttt{e}_{30}\ldots\texttt{e}_1\texttt{e}_0$ et renvoie en sortie un mot de 32 bits $\texttt{S} = \texttt{s}_{31}\texttt{s}_{30}\ldots\texttt{s}_1\texttt{s}_0$, tel que la valeur de `S` soit la valeur de `E` décalée d'un bit vers la droite, un zéro étant inséré à la place du bit de poids fort.

Faites de même pour un circuit décaleur non signé d'un bit vers la gauche.

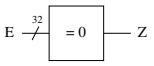
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter des circuits décaleurs à un bit vers la gauche et vers la droite sous la forme des circuits ci-à côté, comprenant une entrée `E` sur 32 bits et une sortie `S` sur 32 bits.



(1.5) (10 points)

Construisez un circuit de test de nullité, prenant une valeur d'entrée $\texttt{E} = \texttt{e}_{31}\texttt{e}_{30}\ldots\texttt{e}_1\texttt{e}_0$ sur 32 bits et muni d'une sortie `Z` sur un bit, telle que $\texttt{Z} = 1$ si `E` vaut 0 et $\texttt{Z} = 0$ si $\texttt{E} \neq 0$.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter le circuit testeur de nullité sous la forme du circuit ci-à côté, comprenant une entrée `E` sur 32 bits et une sortie `Z` sur 1 bit.



(1.6) (20 points)

En vous inspirant des circuits précédents et de registres « REG » sur 32 bits, mis à jour sur les fronts montants de l'horloge, construisez un circuit synchrone, disposant d'une entrée `E` sur 32 bits, d'un fil d'horloge `CK` et d'un fil de contrôle `C` sur un bit chacun, et en sortie d'une valeur `S` sur 32 bits et d'un fil `T` sur un bit, de telle sorte que :

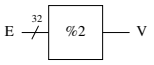
— si $\texttt{C} = 1$, le registre du circuit est chargé avec la valeur sur 32 bits fournie sur l'entrée `E` du circuit ;
— si $\texttt{C} = 0$, la valeur du registre, lisible sur la sortie `S` du circuit, est décalée à droite d'un bit à chaque top d'horloge ;
— lorsque la valeur de sortie `S` vaut 0, le fil de sortie `T` du circuit vaut 1, sinon `T` vaut 0.

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

(1.7) (10 points)

Comment réaliser en pratique, de façon simple, le test « `((a % 2) == 1)` » de l'algorithme, permettant d'ajouter ou non à `p` la valeur de `b` ? Pour ce faire, supposez que la variable `a` est câblée sous la forme d'une valeur d'entrée $\texttt{E} = \texttt{e}_{31}\texttt{e}_{30}\ldots\texttt{e}_1\texttt{e}_0$.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter le circuit testeur de parité sous la forme du circuit ci-à côté, comprenant une entrée `E` sur 32 bits et une sortie `V` sur un bit qui vaut 1 si le résultat du test est vrai.



(1.8) (20 points)

En vous inspirant des circuits précédents, de registres « REG » sur 32 bits, mis à jour sur les fronts montants de l'horloge, ainsi que d'un circuit additionneur sur 32 bits mis à votre disposition appelé « ADD », réalisez un circuit synchrone piloté par un fil d'horloge `CK` et un fil de contrôle C, tel que :

— si $\texttt{C} = 1$, les registres `a` et `b` du circuit sont chargés avec les valeurs d'entrée `A` et `B` sur 32 bits founies en entrée, et le registre `p` est chargé avec la valeur 0 ;
— si $\texttt{C} = 0$, le circuit modifie les valeurs de `a`, `b` et `p` 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 résultat est contenu dans le registre `p` et le fil de sortie `T` du circuit passe à 1.

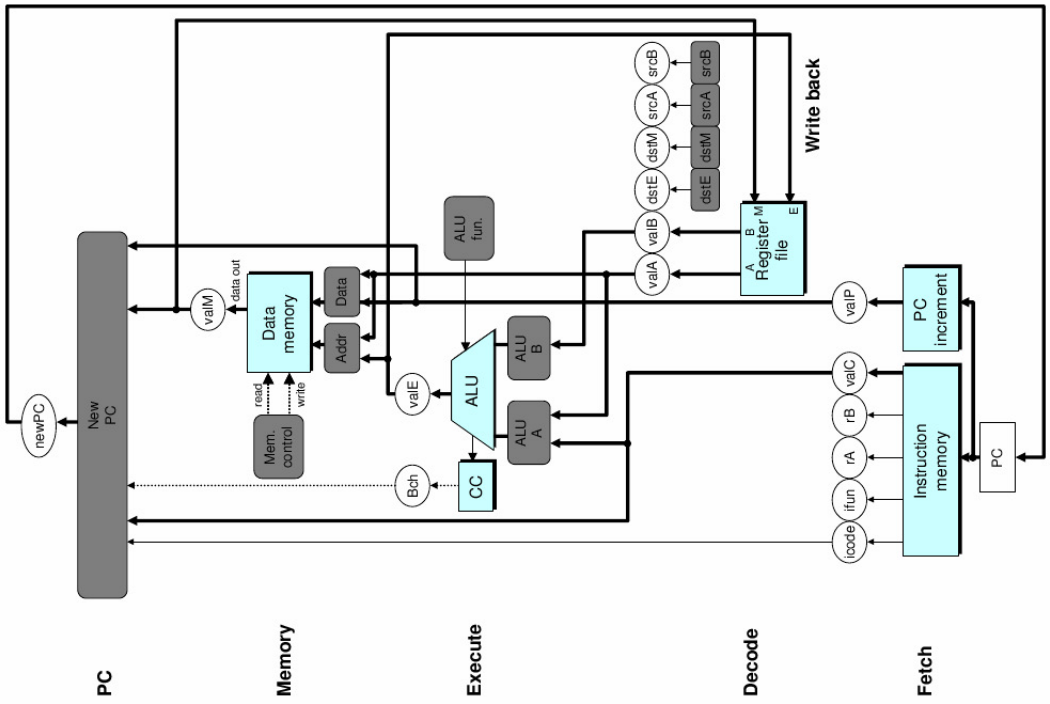
N.B. : Mettez en œuvre une solution qui ne crée pas de dérive de l'horloge !

(1.9) (10 points)

Modifiez le circuit précédent afin que, tant qu'un calcul n'est pas terminé, il ne soit pas possible d'insérer de nouvelles valeurs dans le circuit.

Exercice 2 (100 points)

La figure ci-dessous est le schéma de l'architecture y86 séquentielle.



Le tableau ci-dessous représente le fonctionnement des différents étages de l'architecture Y86 séquentielle lors de l'exécution des deux instructions « `popl rA` » et « `jxx valC` » du jeu d'instructions Y86.

Étage	popl rA	j*xx* D
Fetch	icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	icode:ifun = M_1[PC] valC = M_4[PC+1] valP = PC + 5
Decode	valA = R[%esp] valB = R[%esp]	
Execute	valE = valB + 4	Bch = Cond (CC, ifun)
Memory	valM = M_4[valA]	
Write back	R[%esp] = valE R[rA] = valM	
PC update	PC = valP	PC = (Bch) ? valC : valP

Rappel : « R[x] » indique un accès à la banque de registres à l'adresse (numéro de registre) x, et « M_y[x] » indique un accès à la mémoire centrale (d'instructions et/ou de données) de y octets à l'adresse x. « PC » est le registre compteur ordinal.

(2.1) (20 points)

Expliquez le sens des opérations effectuées à chaque étage pour l'instruction « `popl` ».

N.B. : **on ne veut pas de paraphrase** ; il faudra, pour chaque étape, expliquer *pourquoi* les opérations sont effectuées.

On veut câbler en HCL l'instruction « `maxl rA,rB` », qui place dans le registre `rB` le maximum des valeurs contenues dans les deux registres `rA` et `rB` (le registre `rB` peut donc être modifié, ou pas).

(2.2) (20 points)

Comment peut-on utiliser les circuits déjà présents dans l'architecture Y86 séquentielle actuelle pour déterminer si la valeur de la banque de registres correspondant au registre `rB` doit être changée ou non ? Quelle opération doit-elle être réalisée ?

(2.3) (10 points)

Au niveau de la banque de registres, comment peut-on faire en sorte que la valeur du registre de numéro `rB` soit réécrite avec la bonne valeur, selon qu'il faut la remplacer ou pas ?

(2.4) (10 points)

Quelle valeur d'`ifun` est-il pertinent de donner à l'instruction `maxl` afin que la valeur `Bch` soit à 1 lorsque la valeur contenue dans la banque de registres doit être remplacée ? *(il ne s'agit pas de donner une valeur numérique, mais de dire quelle instruction existante possède la valeur à reprendre.)*

(2.5) (20 points)

Écrivez le tableau représentant le comportement des différents étages de l'architecture Y86 séquentielle pour exécuter l'instruction `maxl`, à l'image des tableaux présentés au début de cet exercice.

(2.6) (20 points)

Ajoutez à la feuille HCL ci-jointe, à détacher et à rendre avec votre copie, les instructions HCL permettant de mettre en œuvre l'instruction `maxl`.

***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;
];
```