université de BORDEAUX	**ANNÉE UNIVERSITAIRE 2022 – 2023** **SESSION 1 DE PRINTEMPS** **Parcours / Étape : LSTS / L2** **Code UE : 4TIN408U** **Épreuve :** Architecture des ordinateurs **Date :** 05 mai 2023 **Heure :** 11h30 **Durée :** 1h30 Documents : non autorisés Épreuve de M./Mme : F. Pellegrini	Collège Sciences et Technologies

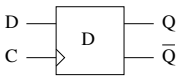
N.B. : - Les réponses aux questions doivent être argumentées et aussi concises que possible.
- Le barème est donné à titre indicatif.
- Inscrivez votre numéro d'anonymat sur la feuille à joindre avant de l'insérer dans votre copie.

Exercice 1 (100 points)

On s'intéresse à des circuits manipulant des nombres entiers signés sur 4 bits, en notation « complément à deux ». Le but est de créer un circuit compteur/décompteur, qui va tendre vers zéro quelle que soit la valeur initiale du compteur.

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.

Vous pourrez également utiliser des bascules D. 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.



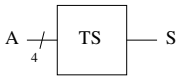
(1.1) (10 points)

Expliquer ce qu'est le « complément à deux » en arithmétique entière.
(10 lignes maximum)

(1.2) (5 points)

Câblez un circuit détecteur de signe qui, prenant en entrée une valeur entière signée $\texttt{A} = \texttt{a}_3\texttt{a}_2\texttt{a}_1\texttt{a}_0$ sur 4 bits, fournit une sortie `S` sur 1 bit telle que $\texttt{S} = 0$ lorsque `A` est positif ou nul, et $\texttt{S} = 1$ lorsque `A` est négatif.

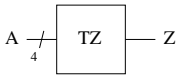
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce testeur de signe `TS` sous la forme du circuit ci-à côté (dessiné verticalement ou horizontalement).



(1.3) (5 points)

Câblez un circuit détecteur de valeur nulle qui, prenant en entrée une valeur entière signée $\texttt{A} = \texttt{a}_3\texttt{a}_2\texttt{a}_1\texttt{a}_0$ sur 4 bits, fournit une sortie `Z` sur 1 bit telle que $\texttt{Z} = 1$ lorsque `A` est égal à zéro.

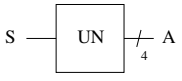
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce testeur de zéro `TZ` sous la forme du circuit ci-à côté (dessiné verticalement ou horizontalement).



(1.4) (10 points)

Donnez le codage binaire des nombres +1 et −1 sur 4 bits, en notation « complément à deux ». Créez un circuit « générateur de uns » qui, prenant en entrée une valeur binaire `S` sur 1 bit, renvoie un nombre signé ($\texttt{A} = \texttt{a}_3\texttt{a}_2\texttt{a}_1\texttt{a}_0$), tel que $\texttt{A} = -1$ si $\texttt{S} = 0$ et $\texttt{A} = +1$ si $\texttt{S} = 1$.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce circuit « `UN` » sous la forme du circuit ci-à côté (dessiné verticalement ou horizontalement).

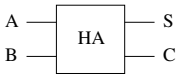


(1.5) (10 points)

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 `S` représente la somme et `C` la retenue.

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

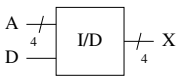
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce demi-additionneur ***HA*** *sous la forme du circuit ci-à côté (dessiné verticalement ou horizontalement).*



(1.6) (10 points)

En vous appuyant sur le circuit précédent, dessinez le schéma de câblage d'un circuit incrémenteur/décrémenteur à 4 bits, prenant en entrée un nombre entier signé $\mathtt{A} = \mathtt{a}_3\mathtt{a}_2\mathtt{a}_1\mathtt{a}_0$ sur 4 bits et une valeur d'entrée de direction `D` sur 1 bit, et fournissant en sortie un nombre entier signé $\mathtt{X} = \mathtt{x}_3\mathtt{x}_2\mathtt{x}_1\mathtt{x}_0$, tel que $\mathtt{X} = (\mathtt{A} + 1)$ si $\mathtt{D} = 0$, et $\mathtt{X} = (\mathtt{A} - 1)$ si $\mathtt{D} = 1$ (il n'y aura pas de gestion des débordements éventuels).

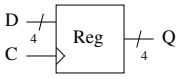
À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter cet incrémenteur/décrémenteur « I/D » sous la forme du circuit ci-à côté (dessiné verticalement ou horizontalement).



(1.7) (5 points)

Au moyen de bascules D, créez un registre sur 4 bits, disposant d'une entrée $\mathtt{D} = \mathtt{d}_3\mathtt{d}_2\mathtt{d}_1\mathtt{d}_0$ sur 4 bits et d'une sortie $\mathtt{Q} = \mathtt{q}_3\mathtt{q}_2\mathtt{q}_1\mathtt{q}_0$ sur 4 bits, et dont la mémorisation est activée par une entrée C.

*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 (dessiné verticalement ou horizontalement).*



(1.8) (10 points)

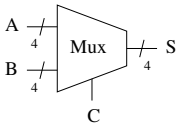
Au moyen des circuits précédents, réalisez un compteur incrémenteur/décrémenteur muni d'une sortie à quatre fils $\mathtt{X} = \mathtt{x}_3\mathtt{x}_2\mathtt{x}_1\mathtt{x}_0$, et tel que la valeur de `X` est incrémentée ou décrémentée à chaque front montant de l'horloge, selon le sens d'une entrée de direction `D` (incrémentation si $\mathtt{D} = 0$ et décrémentation si $\mathtt{D} = 1$).

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

(1.9) (5 points)

Écrivez l'équation logique d'un multiplexeur à deux entrées `A` et `B` sur 1 bit, et un bit de contrôle `C`, qui renvoie sur sa sortie `S` la valeur `A` si $\mathtt{C} = 0$ et `B` si $\mathtt{C} = 1$. Dessinez le schéma de câblage de ce multiplexeur, au moyen de portes logiques.

À partir de maintenant, vous pouvez utiliser des multiplexeurs à 1 bit de contrôle, soit à 1 bit d'entrée et de sortie, soit à 4 bits d'entrée et de sortie tel que sous la forme du circuit ci-à côté (dessiné verticalement ou horizontalement).



(1.10) (10 points)

Modifiez le circuit compteur/décompteur précédent pour que l'on puisse modifier la valeur du compteur pour la positionner à la valeur fournie par une entrée `V` sur 4 bits, lorsqu'une entrée `M`sur 1 bit est à `1`.

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

(1.11) (10 points)

Modifiez le circuit précédent pour que le circuit incrémente la valeur du compteur si celle-ci est négative, et la décrémente si elle est positive ou nulle.

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

(1.12) (10 points)

Que se passe-t-il avec le circuit précédent une fois que le compteur atteint la valeur zéro ? Modifiez le circuit pour qu'il reste à zéro une fois cette valeur atteinte.

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

Exercice 2 (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 « `aluA = [ ... ]` ».

(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 « `[ ... ]` ». Dites à quel cas de figure chaque ligne correspond, et la valeur qu'elle prendra pour chacun des différents types d'opérations (`icode`) considérés dans la ligne.

Exercice 3 (60 points)

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

(3.1) (20 points)

Sur la feuille de code HCL (à rendre), effectuez la factorisation des instructions `nop` et `halt`. À la place des `icode` « `NOP` » et « `HALT` », utilisez l'`icode` « `NOHA` », avec une valeur de `ifun` égale à `0` pour `nop` et à `1` pour `halt`.

On se propose maintenant d'introduire une nouvelle instruction dans le processeur Y86 séquentiel : « `jcxz` » (« *Jump if CX is Zero* »). Cette instruction fonctionne comme un `jmp`, mais n'effectue le branchement que si la valeur du registre `ecx` est nulle. Elle aura son propre `icode`, dont le nom symbolique sera « `JCXZ` ».

(3.2) (20 points)

Expliquez comment vous souhaitez mettre en œuvre cette instruction. Notamment :

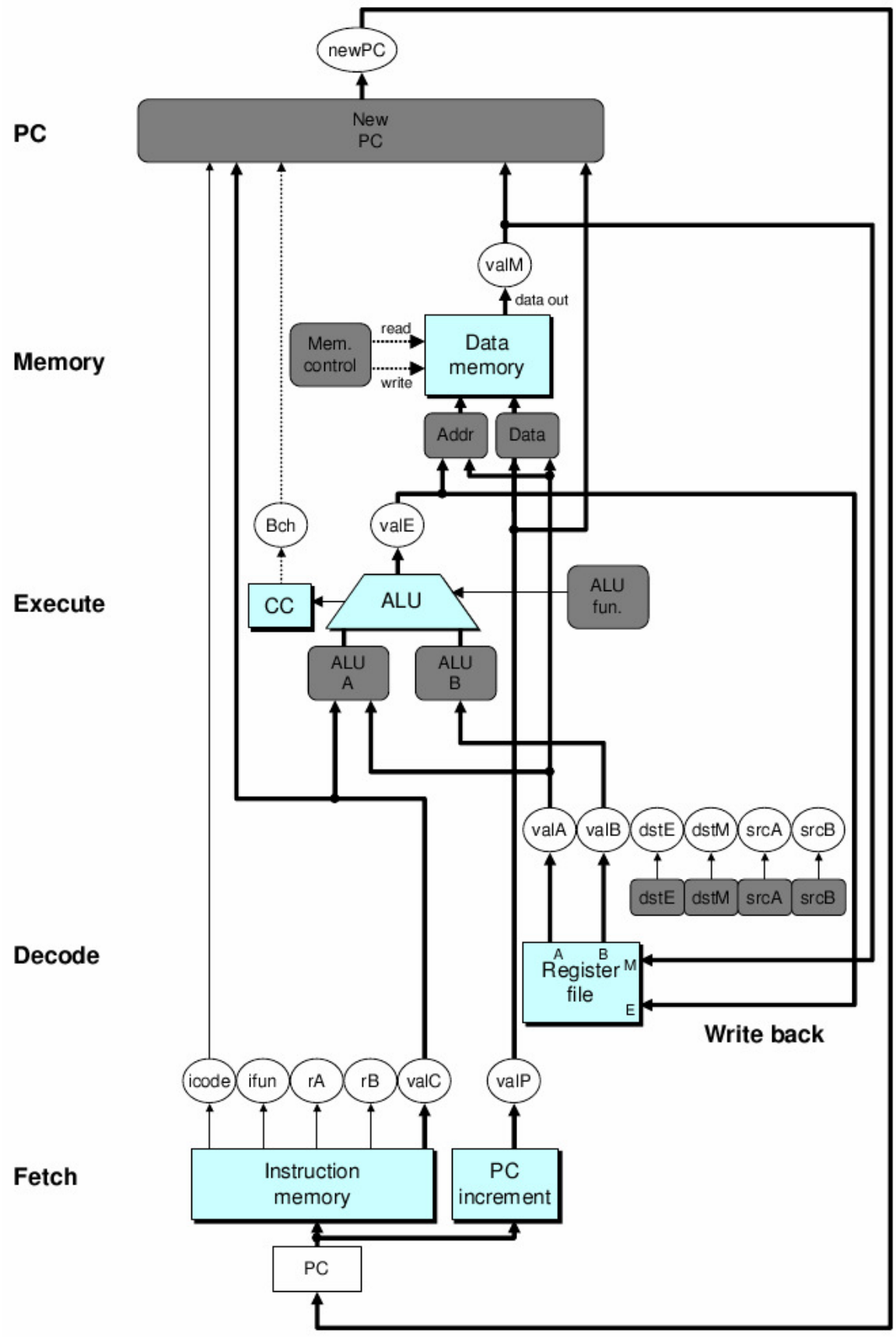
- Comment allez vous configurer les différentes unités fonctionnelles du processeur (banque de registres, unité arithmétique et logique, mémoire, etc.) ?
- Quel `ifun` allez-vous choisir pour votre instruction ?
- Les drapeaux d'état (`Z`, `S` et `O`) seront-ils modifiés par votre solution ?

(3.3) (20 points)

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

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

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



— 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;
];
```