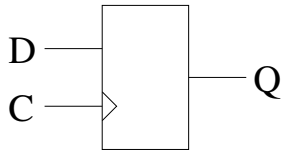
université de BORDEAUX	**ANNÉE UNIVERSITAIRE 2017 – 2018** **SESSION 1 D'AUTOMNE** **Parcours / Étape : LSTS / L2** **Code UE : 4TIN304U** **Épreuve :** Architecture des ordinateurs **Date :** 9 janvier 2018 **Heure :** 11h30 **Durée :** 1h30 Documents : non autorisés Épreuve de M./Mme : F. Pellegrini	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 (80 points)

Les bascules D sont des circuits logiques à mémoire disposant de deux fils d'entrée et d'un fil de sortie : 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 ainsi :



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

(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 3 bits, prenant en entrée un nombre entier non signé $\mathtt{A} = \mathtt{a}_2\mathtt{a}_1\mathtt{a}_0$ et fournissant en sortie un nombre entier non signé $\mathtt{X} = \mathtt{x}_2\mathtt{x}_1\mathtt{x}_0$, tel que $\mathtt{X} = (\mathtt{A} + 1) \mod 8$. 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 $\mathtt{x}_2$, à partir de $\mathtt{a}_2$ et $\mathtt{c}_1$.

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) (10 points)

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

(1.4) (10 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 $\mathtt{z} = 1$.

TSVP .../... ⟶

(1.5) (10 points)

En notation « complément à deux », comment calcule-t-on l'opposé d'un nombre ? Quel est le codage de la valeur « -1» sur 3 bits ?

(1.6) (20 points)

En vous appuyant sur la définition précédente, câblez un circuit décrémenteur à 3 bits, prenant en entrée un nombre entier non signé $\mathtt{A} = \mathtt{a}_2\mathtt{a}_1\mathtt{a}_0$ et fournissant en sortie un nombre entier non signé $\mathtt{X} = \mathtt{x}_2\mathtt{x}_1\mathtt{x}_0$, tel que $\mathtt{X} = (\mathtt{A} - 1) \mod 8$. 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 $\mathtt{x}_2$, à partir de $\mathtt{a}_2$ et $\mathtt{c}_1$.

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

(1.7) (10 points)

En vous inspirant des circuits précédents et de multiplexeurs `MUX` à un bit de contrôle `c`, disposant chacun de deux entrées à un bit `a` et `b` et d'une sortie à un bit `s`, réalisez un compteur incrémenteur/décrémenteur muni d'une sortie à trois fils $\mathtt{X} = \mathtt{x}_2\mathtt{x}_1\mathtt{x}_0$, de deux entrées `z` et `t` sur un bit chacune, et tel que, à chaque front montant de l'horloge, la valeur de `X` est incrémentée si $\mathtt{t} = 0$, décrémentée si $\mathtt{t} = 1$, et remise à zéro si $\mathtt{z} = 1$ (quelle que soit la valeur de `t`).

Question 2 (50 points)

Le tableau ci-dessous représente le fonctionnement des différents étages lors de l'exécution des trois instructions « `irmovl valC,rB` », « `opl rA,rB` », et « `popl rA` ».

Étage	irmovl valC,rB	opl rA,rB	popl rA
Fetch	icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valC = M_4[PC+2] valP = PC + 6	icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2
Decode		valA = R[rA] valB = R[rB]	valA = R[%esp] valB = R[%esp]
Execute	ValE = 0 + valC	valE = valB OP(ifun) valA CC = SetCC ()	valE = valB + 4
Memory			ValM = M_4[valA]
Write back	R[rB] = valE	R[rB] = valE	R[%esp] = valE R[rA] = valM
PC update	PC = valP	PC = valP	PC = 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)

En vous basant sur les informations précédentes, remplissez les tableaux correspondant aux instructions « `rrmovl rA,rB` », « `iopl valC,rB` » et « `ret` ».

(2.2) (20 points)

Complétez le code HCL de la feuille jointe afin de prendre en compte les trois instructions que vous venez de définir.

(2.3) (10 points)

On souhaite ajouter au processeur l'instruction `jreg rA`, qui effectue un branchement inconditionnel à l'adresse contenue dans le registre `rA`. Comme pour les trois instructions précédentes, ajoutez sur la feuille jointe, là où cela est nécessaire, le code permettant de mettre en œuvre l'instruction `JREG`.

TSVP .../... ⟶

Question 3 (70 points)

Il s'agit d'écrire le code Y86 correspondant aux fonctions C ci-contre.

(3.1) (20 points)

Expliquez le passage des paramètres par la pile, ainsi que les conventions «*caller save / callee save* » des registres que vous utilisez. *(15 lignes maximum)*

(3.2) (30 points)

Codez la fonction `main` et les éléments permettant que le programme s'exécute bien au sein de l'environnement Y86.

(3.3) (20 points)

Codez la fonction `Cnp`.

```
int t[10];

Cnp (int n, int p)
{
  if ((p == 0) || (p == n))
    return (1);
  else
    return (Cnp (n - 1, p) +
            Cnp (n - 1, p - 1));
}

main () {
  int    i;

  for (i = 0; i <= 9; i ++)
    t[i] = Cnp (9, i);
}
```

— 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 { OPL, POPL, IRMOVL };

# Does fetched instruction require a constant word?
bool need_valC =
        icode in { IRMOVL };

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

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

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

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

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

TSVP .../... ⟶

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

## Select input A to ALU
int aluA = [
        icode in { OPL } : valA;
        icode in { IRMOVL } : valC;

        icode in { POPL } : 4;
        # Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
        icode in { OPL, POPL } : valB;
        icode in { 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 in { OPL };

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

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

## Set write control signal
bool mem_write = icode in { };

## Select memory address
int mem_addr = [

        icode in { POPL } : valA;
        # Other instructions don't need address
];

## Select memory input data
int mem_data = [

        # Default: Don't write anything
];

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

## What address should instruction be fetched at

int new_pc = [

        1 : valP;
];
```