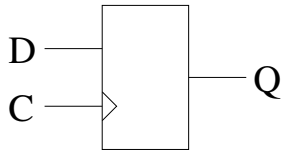
université de BORDEAUX	**ANNÉE UNIVERSITAIRE 2016 – 2017** **SESSION 1 D'AUTOMNE** **Parcours / Étape : LSTS / L2** **Code UE : 4TIN304U** **Épreuve :** Architecture des ordinateurs **Date :** 3 janvier 2017 **Heure :** 09h00 **Durée :** 1h30 Documents : non autorisés Épreuve de M./Mme : F. Pellegrini	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.

Question 1 (60 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 ainsi :



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

(1.1) (15 points)

Comment câbler les fils `D` et `Q` d'une bascule D, au moyen de portes logiques, pour qu'à chaque front montant de `C` la sortie `Q` de la bascule change d'état ? Dessinez le schéma correspondant. *(On ne veut pas de dérive de l'horloge)*

(1.2) (15 points)

Tracer un chronogramme représentant l'évolution des états de `H` et de `Q` au cours du temps sur quatre cycles d'horloge ; on supposera que l'état initial de `Q` est 0. Au vu de ce schéma, que pouvez-vous dire du signal de `Q` par rapport à celui de `H` ?

(1.3) (15 points)

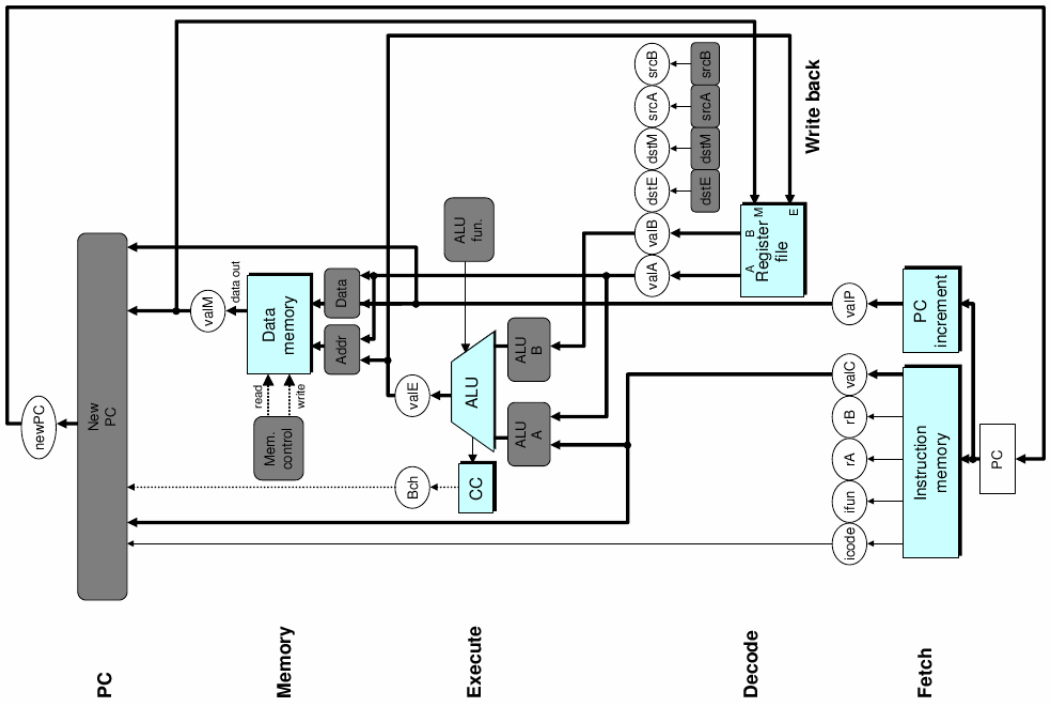
On veut câbler deux bascules D, appelées $\mathtt{D}_0$ et $\mathtt{D}_1$, au moyen de portes logiques, pour que leurs états de sortie $\mathtt{Q}_0$ et $\mathtt{Q}_1$ parcourent les quatre états 00, 01, 10, 11, dans cet ordre, en changeant d'état à chaque front montant de `H`. Le signal $\mathtt{Q}_0$ correspondra au bit de poids faible et $\mathtt{Q}_1$ au bit de poids fort. Dessinez le chronogramme correspondant de $\mathtt{Q}_0$ et $\mathtt{Q}_1$ pour cinq cycles d'horloge. Déduisez-en les tables des fonctions logiques $\mathtt{D}_0 = f_0(\mathtt{Q}_0, \mathtt{Q}_1)$ et $\mathtt{D}_1 = f_1(\mathtt{Q}_0, \mathtt{Q}_1)$, puis dessinez le schéma de câblage correspondant. *(On ne veut pas de dérive de l'horloge)*

(1.4) (15 points)

Au moyen de portes logiques et de bascules D, Réalisez le schéma d'un circuit permettant de commander trois signaux de sortie afin de réaliser un « chenillard », prenant successivement les six états suivants : 000, 001, 011, 111, 110, 100, et ainsi de suite, et changeant d'état à chaque front montant de H. On étiquettera de $\mathtt{S}_0$ à $\mathtt{S}_2$ les sorties du chenillard, $\mathtt{S}_0$ correspondant au bit de poids le plus faible des six représentations binaires ci-dessus. *(On ne veut pas de dérive de l'horloge)*

Question 2 (70 points)

La figure ci-dessous est le schéma de l'architecture Y86 séquentielle.



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

Étage	irmovl valC,rB	popl rA	call valC
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] valC = M_4[PC+1] valP = PC + 5
Decode		valA = R[%esp] valB = R[%esp]	valB = R[%esp]
Execute	ValE = 0 + valC	valE = valB + 4	valE = valB + (-4)
Memory		ValM = M_4[valA]	M_4[valE] = valP
Write back	R[rB] = valE	R[%esp] = valE R[rA] = valM	R[%esp] = valE
PC update	PC = valP	PC = valP	PC = valC

(2.1) (20 points)

En vous basant sur les informations précédentes, remplissez les tableaux correspondant aux instructions « `rrmovl rA,rB` », « `pushl rA` » 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 `xchgl rA,rB`, qui échange le contenu des deux registres arguments de l'instruction. Combien d'accès à la banque de registres sont-ils nécessaires pour mettre en œuvre cette instruction ?

(2.4) (10 points)

Pourquoi cette instruction ne peut-elle être mise en œuvre avec l'architecture présentée dans le schéma vu plus haut ? Justifiez votre réponse. De fait, quelles modifications proposez-vous au chemin de données du processeur pour supporter cette instruction ?

(2.5) (10 points)

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 `XCHGL`. On ne demande pas d'écrire le code correspondant à la modification du chemin de données.

TSVP .../... ⟶

Question 3 (70 points)

Il s'agit d'écrire le code Y86 correspondant aux fonctions C ci-contre. Pour cela, vous disposez d'une fonction `mul` qui prend en paramètres deux valeurs entières, et renvoie le résultat de leur multiplication.

(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) (20 points)

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

(3.3) (30 points)

Codez les fonctions `fact2` et `fact`.

```
int fact2 (int n)
{
  if (n <= 1)
    return (1);
  else
    return (n * fact2 (n - 1));
}
int fact (int n) {
  if (n < 0)
    return (-1);   /* Erreur */
  else
    return (fact2 (n));
}
main ()
{
  a = fact (5);
}
int   a;
```

— 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, IOPL, POPL, IRMOVL, RMMOVL, MRMOVL };

# Does fetched instruction require a constant word?
bool need_valC =
icode in { IRMOVL, RMMOVL, MRMOVL, JXX, CALL, IOPL };

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

## What register should be used as the A source?
int srcA = [
icode in { RMMOVL, 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, IOPL, RMMOVL, MRMOVL } : rB;
icode in { POPL, CALL } : RESP;
1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
int dstE = [
icode in { IRMOVL, OPL, IOPL } : rB;
icode in { POPL, CALL } : 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
];
```

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

## Select input A to ALU
int aluA = [
icode in { OPL } : valA;
icode in { IRMOVL, RMMOVL, MRMOVL, IOPL } : valC;
icode in { CALL } : -4;
icode in { POPL } : 4;
# Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
icode in { RMMOVL, MRMOVL, OPL, IOPL, CALL, POPL } : valB;
icode in { IRMOVL } : 0;
# Other instructions don't need ALU
];

## Set the ALU function
int alufun = [
icode in { OPL, IOPL } : ifun;
1 : ALUADD;
];

## Should the condition codes be updated?
bool set_cc = icode in { OPL, IOPL };

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

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

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

## Select memory address
int mem_addr = [
icode in { RMMOVL, CALL, MRMOVL } : valE;
icode in { POPL } : valA;
# Other instructions don't need address
];

## Select memory input data
int mem_data = [
icode in { RMMOVL } : 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;
1 : valP;
];
```