Université de Bordeaux — U.F.R. Mathématiques et Informatique
Licence Informatique 2ème année — Année 2013 – 2014

ARCHITECTURE DES ORDINATEURS

Devoir Surveillé 2

1 heure 20 minutes
sans documents

N.B. : - Travaillez calmement. Ne bâclez pas vos réponses : il vaut mieux traiter correctement moins de questions que tout faire de travers.
- Les réponses aux questions doivent être argumentées et aussi concises que possible.
- Le barème est donné à titre indicatif.

Question 1 (10 points)

On veut réaliser un circuit calculant la puissance $\mathtt{N}^{\text{ième}}$ d'un nombre entier `X`, avec `N` entier positif ou nul. Pour ce faire, vous disposez de tous les blocs logiques vus en cours : comparateurs, additionneurs, multiplicateurs, décodeurs, multiplexeurs, bascules D, etc. Un signal d'horloge `Clk` vous est également fourni.

Le fonctionnement de ce circuit est 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, car lorsque le circuit aura terminé son calcul, `R` contiendra le résultat final.

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

Le but de cet exercice est de construire, pas à pas, un tel circuit. On construire d'abord une version non optimisée, puis une version optimisée.

(1.1) (1 point)

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 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`. C'est un nouveau bloc que vous pourrez utiliser dans la construction du circuit.

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

$\mathtt{R} \leftarrow 1$
while $(\mathtt{N} > 0)$ **do**
  $\mathtt{R} \leftarrow \mathtt{R} \times \mathtt{X}$
  $\mathtt{N} \leftarrow \mathtt{N} - 1$
end while

(1.2) (3 points)

Créez un circuit décompteur doté des propriétés suivantes :

— Un fil de sortie `S` 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 sur n bits.

— À chaque top d'horloge, le contenu du compteur 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 à 1), jusqu'à ce qu'une nouvelle valeur soit chargée dans le compteur.

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

(1.3) (2 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 :

$\mathtt{R} \leftarrow 1$
while ($\mathtt{N} > 0$) **do**
  if (`N` est impair) **then**
    $\mathtt{R} \leftarrow \mathtt{R} \times \mathtt{X}$
    $\mathtt{N} \leftarrow \mathtt{N} - 1$
  end if
  $\mathtt{X} \leftarrow \mathtt{X}^2$
  $\mathtt{N} \leftarrow \mathtt{N}/2$
end while

(1.4) (1 point)

Comment savoir simplement si une valeur entière est impaire ? Comment peut-on simplement effectuer une division par 2 d'un nombre entier positif ?

(1.5) (3 points)

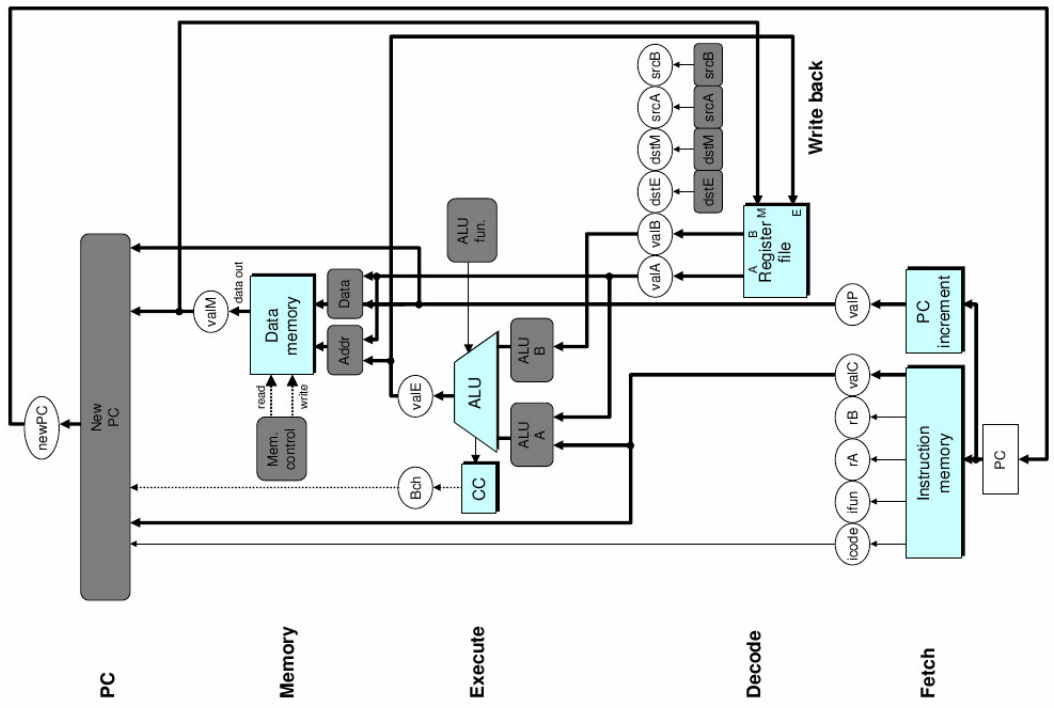
En reprenant les éléments pertinents des circuits vus précédemment, réalisez un circuit qui calcule $\mathtt{X}^{\mathtt{N}}$, selon la méthode optimisée de l'algorithme ci-dessus.

Question 2 (3 points)

Expliquez ce qu'est le « *data forwarding* » dans le contexte des architectures pipe-linées.
(20 lignes maximum)

Question 3 (7 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 « `rrmovl rA,rB` », « `pushl rA` » et « `ret` ».

Étage	rrmovl rA,rB	pushl rA	ret
Fetch	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	icode:ifun = M_1[PC] valP = PC + 1
Decode	valA = R[rA]	valA = R[rA] valB = R[%esp]	valA = R[%esp] valB = R[%esp]
Execute	ValE = valA + 0	valE = (-4) + valB	valE = 4 + valB
Memory		M_4[valE] = valA	valM = M_4[valA]
Write back	R[rB] = valE	R[%esp] = valE	R[%esp] = valE
PC update	PC = valP	PC = valP	PC = valM

(3.1) (4 points)

En vous basant sur les informations précédentes, remplissez le tableau correspondant aux instructions « `mrmovl D(rB),rA` », « `popl rA` » et « `call val` ».

Le code ci-dessous est un fragment incomplet de code HCL correspondant à l'architecture Y86 séquentielle.

```
## What registers should be read from and written to register bank
int srcA = [
                icode in { RRMOVL, RMMOVL, OPL, PUSHL } : rA;
                icode in { RET } : RESP;
                1 : RNONE; # Don't need register
];

int srcB = [
                icode in { OPL, IOPL, RMMOVL } : rB;
                icode in { PUSHL, RET } : RESP;
                1 : RNONE;  # Don't need register
];

int dstE = [
                icode in { RRMOVL, IRMOVL, OPL, IOPL } : rB;
                icode in { PUSHL, RET } : RESP;
                1 : RNONE;  # Don't need register
];

## Select input A to ALU
int aluA = [
            icode in { RRMOVL, OPL } : valA;
            icode in { IRMOVL, RMMOVL, IOPL } : valC;
            icode in { PUSHL } : -4;
            icode in { RET } : 4;
];

## Select input B to ALU
int aluB = [
            icode in { RMMOVL, OPL, IOPL, PUSHL, RET } : valB;
            icode in { RRMOVL, IRMOVL } : 0;
];

## Select memory address
int mem_addr = [
                icode in { RMMOVL, PUSHL } : valE;
                icode in { RET } : valA;
];
```

(3.2) (3 points)

Complétez ce code afin de prendre en compte les trois instructions que vous venez de définir.