DEVUIP
Service Scolarité

ANNÉE UNIVERSITAIRE 2011 / 2012
SESSION 1 DE PRINTEMPS

PARCOURS / ÉTAPE : LSTS / L2 **CODE UE : J1IN4001**
Épreuve : Architecture des ordinateurs (INF 155)
Date : 15 mai 2012 **Heure :** 14h **Durée :** 3h
Documents : non autorisés
Épreuve de M./Mme : F. Pellegrini

N.B. : - Les réponses aux questions doivent être argumentées et aussi concises que possible.
- Le barème est donné à titre indicatif.

Question 1 (20 points)

(1.1) (10 points)

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

(1.2) (10 points)

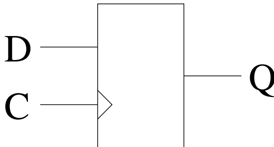
En utilisant des portes ET, OU et NON, construisez et dessinez un dispositif ayant les propriétés suivantes :

– le dispositif possèdera 4 lignes d'entrée et 2 lignes de sortie ;
– les 4 lignes d'entrée représentent l'écriture, au moyen de la notation en complément à deux, de nombres x en arithmétique entière signée ;
– les 2 lignes de sortie doivent fournir les valeurs suivantes :

Condition	S_0	S_1
$x = 0$	0	0
$x < 0$	0	1
$x > 1$	1	0
$x = 1$	1	1

Question 2 (40 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.

(2.1) (10 points)

Comment câbler les fils D et Q d'une bascule pour qu'à chaque front montant de C la sortie Q de la bascule change d'état ? Dessinez le schéma correspondant.

(2.2) (10 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 ?

(2.3) (10 points)

Comment câbler deux bascules D, appelées D_0 et D_1, pour que leurs états de sortie Q_0 et Q_1 parcourent les quatre états 00, 01, 10, 11, dans cet ordre, en changeant d'état à chaque front montant de H ? Dessinez le schéma correspondant. Le signal Q_0 correspondra au bit de poids faible, et Q_1 au bit de poids fort.

(2.4) (10 points)

Réalisez le schema 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 S_0 à S_2 les sorties du chenillard, S_0 correspondant au bit de poids le plus faible des six représentations binaires ci-dessus.

Question 3 (40 points)

Soit un processeur pipe-liné à quatre étages exécutant des instructions en virgule flottante du type $R_i \leftarrow R_j$ *op* R_k. Le pipe-line est constitué d'un étage F réalisant la recherche des instructions (« *instruction fetch* »), d'un étage D pour leur décodage et la lecture des opérandes à partir de la banque de registres, d'un étage E d'exécution, et d'un étage W de réécriture du résultat dans la banque des registres. Cette banque autorise plusieurs lectures mais en revanche une seule écriture par cycle. Les étages F, D et W prennent chacun un cycle. L'étage E, lui-même pipe-liné et cadencé à la même fréquence que le reste du processeur, peut exécuter soit une addition flottante en deux cycles, soit une multiplication flottante en quatre cycles. Le programme exécuté par le processeur est le suivant :

S1 :	R2	←	R1 + R4	S4 :	R4	←	R1 × R2
S2 :	R3	←	R2 × R4	S5 :	R1	←	R2 + R3
S3 :	R2	←	R4 + R5	S6 :	R4	←	R3 × R4

(3.1) (10 points)

Supposez que le processeur ne possède qu'une seule unité d'addition et une seule unité de multiplication, pipe-linées comme décrit ci-dessus, et ne met pas en œuvre de *data forwarding*. Pour cette question, on considèrera également que, parce que l'unité d'addition est nécessaire au calcul d'une multiplication, on ne peut utiliser simultanément les pipe-lines d'addition et de multiplication.

Mettez en évidence toutes les dépendances de données et d'exécution se produisant lors de l'exécution de ce fragment de code. Indiquez à chaque fois la nature de la dépendance. Pour ce faire, complétez et remplissez un schéma analogue à celui entamé ci-dessous et matérialisez les zones de bulles en expliquant leur provenance.

I	1	2	3	4	5	6	7	8	...
S_1	F	D	$+_{(1)}$	$+_{(2)}$	W				
S_2									
S_3									
⋮									

(3.2) (10 points)

Supposez maintenant que l'additionneur et le multiplicateur puissent opérer totalement en parallèle l'un de l'autre. Une instruction ne pourra cependant jamais en « doubler » une autre dans le pipe-line global (exécution dite « *in-order issue* ») : en cas de conflit de sortie de l'étage E, c'est la plus ancienne qui atteindra l'étage W.

Calculez le temps d'exécution du fragment de code ci-dessus en fonction des nouvelles caractéristiques du processeur.

(3.3) (10 points)

Supposez que le processeur, en plus des caractéristiques de la question précédente, dispose maintenant d'un mécanisme de *data forwarding*, permettant de fournir directement à l'entrée de l'étage d'exécution le contenu qui vient d'en sortir, sans avoir à passer par les étages W et D, lorsque l'instruction suivante prend comme opérande source l'opérande destination de l'instruction en cours de terminaison. Ce mécanisme permet à l'étage D de terminer même si les données ne sont pas disponibles, l'étage E restant bloqué en entrée jusqu'à la sortie du pipe-line de ce même étage du résultat produit par l'instruction devant fournir la donnée.

Calculez le temps d'exécution du fragment de code ci-dessus en fonction des nouvelles caractéristiques du processeur.

(3.4) (10 points)

Réordonnez les instructions du fragment ci-dessus, éventuellement en utilisant des registres supplémentaires, de façon à minimiser son temps d'exécution sur le processeur disposant du mécanisme de *data forwarding*. Les valeurs des registres à la fin de l'exécution du fragment réordonné doivent être les mêmes que pour le fragment original. Donnez le temps d'exécution du fragment ainsi réordonné.

Question 4 (30 points)

On souhaite pipe-liner un circuit combinatoire. Pour cela, on a décomposé ce circuit en cinq blocs A à E de durées respectives 20, 90, 50, 40 et 60 ps. Ces blocs doivent être exécutés l'un après l'autre dans cet ordre, après quoi on charge un registre au prochain front d'horloge. La durée de chargement d'un tel registre est de 20 ps.

(4.1) (10 points)

Insérer un seul registre intermédiaire fournit un pipeline de profondeur 2. Où faut-il insérer ce registre pour obtenir un débit maximal ? Calculez la durée minimale du cycle d'horloge auquel peut être cadencé le pile-line, son débit et sa latence.

(4.2) (10 points)

Même question que ci-dessus en insérant deux registres intermédiaires au lieu d'un seul, pour obtenir un pipe-line de profondeur 3.

(4.3) (10 points)

Vous disposez maintenant d'autant de registres que vous en avez besoin. Quel est le pipeline de profondeur optimale ? Pourquoi ? Combien de registres utilise-t-il ? Comme précédemment, calculez la durée minimale de son cycle d'horloge, son débit et sa latence.

Question 5 (70 points)

On s'intéresse à la version pipe-linée du processeur Y86, disposant du mécanisme de *data forwarding*.

(5.1) (15 points)

Examinez le programme ci-contre. Que fait-il ? Chronogramme à l'appui, indiquez combien s'écoulent de cycles entre l'instant où l'instruction `rmmovl %eax,(%edi)` entre dans le processeur pour la première fois et l'instant où l'instruction `halt` entre à son tour dans le processeur. Attention à la prédiction de branchement ! Celle-ci opère de façon habituelle pour ce processeur, conformément à ce qui est indiqué dans le code HCL de la feuille jointe.

```
.pos  0
      irmovl t,    %edi
      mrmovl size, %ecx
      xorl   %eax, %eax
      irmovl 1,    %ebx
      irmovl 4,    %edx
boucle:
      rmmovl %eax, (%edi)
      addl   %edx, %edi
      subl   %ebx, %ecx
      jne    boucle
      halt
.pos  100
size: .long  2
t:
```

(5.2) (15 points)

On se propose d'introduire une nouvelle instruction dans le processeur Y86 : `STOSL` (« *Store String Long* »). Cette instruction (sans argument) effectue l'équivalent de `rmmovl %eax,(%edi)`, tout en ajoutant 4 au registre `%edi`, la valeur de `%edi` utilisée par `rmmovl` étant celle avant incrémentation.

La feuille jointe au sujet reproduit l'essentiel du fichier pipe-std.hcl, qui décrit la version pipelinée du processeur Y86. Ecrivez directement sur cette feuille les ajouts que vous proposez pour traiter cette nouvelle instruction (de code `STOSL`).

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

(5.3) (10 points)

Quel est le principal intérêt d'utiliser une telle instruction ? Dans le programme ci-contre, si l'on remplace les deux premières instructions de la boucle par `stosl`, gagne-t-on des cycles ?

(5.4) (15 points)

Pour continuer à réduire le nombre d'instructions au sein de la boucle, on peut s'inspirer des processeurs x86 et envisager d'ajouter une instruction `loop <label>` à notre processeur. Cette instruction décrémente le registre `%ecx`, puis réalise un saut à l'adresse `label` si la nouvelle valeur d'`%ecx` est différente de zéro.

Comme pour l'instruction `STOSL`, ajoutez le code `LOOP` partout où c'est nécessaire sur la feuille jointe afin de gérer correctement le fonctionnement de cette instruction. Laissez de côté les aspects relatifs à la prédiction de branchement pour cette question...

(5.5) (15 points)

La gestion des mauvaises prédictions de branchement pose problème avec l'instruction `LOOP`. Contrairement aux instructions de saut conditionnel (`JXX`), l'instruction `LOOP` appuie sa décision non pas sur des codes de conditions positionnés par des opérations antérieures, mais par le résultat sortant de l'ALU au moment où l'instruction `LOOP` se trouve à l'étage *Execute*. Expliquez précisément d'où vient le problème et comment on pourrait le corriger (on ne demande pas de le faire sur la feuille annexe).

— Feuille à détacher et à rendre avec la copie —

Numéro d'anonymat :

```
################ Fetch Stage     ###################################

## What address should instruction be fetched at
int f_pc = [
# Mispredicted branch.  Fetch at incremented PC
M_icode == JXX && !M_Bch : M_valA;
# Completion of RET instruction.
W_icode == RET : W_valM;
# Default: Use predicted value of PC
1 : F_predPC;
];

# Predict next value of PC
int new_F_predPC = [
f_icode in { JXX, CALL } : f_valC;
1 : f_valP;
];

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

## What register should be used as the A source?
int new_E_srcA = [
D_icode in { RRMOVL, RMMOVL, OPL, PUSHL  } : D_rA;
D_icode in { POPL, RET } : RESP;
1 : RNONE; # Don't need register
];

## What register should be used as the B source?
int new_E_srcB = [
D_icode in { OPL, RMMOVL, MRMOVL  } : D_rB;
D_icode in { PUSHL, POPL, CALL, RET } : RESP;
1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
int new_E_dstE = [
D_icode in { RRMOVL, IRMOVL, OPL} : D_rB;
D_icode in { PUSHL, POPL, CALL, RET } : RESP;
1 : RNONE;  # Don't need register
];

## What register should be used as the M destination?
int new_E_dstM = [
D_icode in { MRMOVL, POPL } : D_rA;
1 : RNONE;  # Don't need register
];

## What should be the A value?
## Forward into decode stage for valA
int new_E_valA = [
D_icode in { CALL, JXX } : D_valP; # Use incremented PC
d_srcA == E_dstE : e_valE;    # Forward valE from execute
d_srcA == M_dstM : m_valM;    # Forward valM from memory
d_srcA == M_dstE : M_valE;    # Forward valE from memory
d_srcA == W_dstM : W_valM;    # Forward valM from write back
```

```
d_srcA == W_dstE : W_valE;    # Forward valE from write back
1 : d_rvalA;  # Use value read from register file
];

int new_E_valB = [
d_srcB == E_dstE : e_valE;    # Forward valE from execute
d_srcB == M_dstM : m_valM;    # Forward valM from memory
d_srcB == M_dstE : M_valE;    # Forward valE from memory
d_srcB == W_dstM : W_valM;    # Forward valM from write back
d_srcB == W_dstE : W_valE;    # Forward valE from write back
1 : d_rvalB;  # Use value read from register file
];

################ Execute Stage ###################################

## Select input A to ALU
int aluA = [
E_icode in { RRMOVL, OPL } : E_valA;
E_icode in { IRMOVL, RMMOVL, MRMOVL } : E_valC;
E_icode in { CALL, PUSHL } : -4;
E_icode in { RET, POPL } : 4;
# Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
E_icode in { RMMOVL, MRMOVL, OPL, CALL,
      PUSHL, RET, POPL } : E_valB;
E_icode in { RRMOVL, IRMOVL } : 0;
# Other instructions don't need ALU
];

## Set the ALU function
int alufun = [
E_icode == OPL : E_ifun;
1 : ALUADD;
];

## Should the condition codes be updated?
bool set_cc = E_icode == OPL;

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

## Select memory address
int mem_addr = [
M_icode in { RMMOVL, PUSHL, CALL, MRMOVL } : M_valE;
M_icode in { POPL, RET } : M_valA;
# Other instructions don't need address
];

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

## Set write control signal
bool mem_write = M_icode in { RMMOVL, PUSHL, CALL };
```