DEVUIP
Service Scolarité

ANNÉE UNIVERSITAIRE 2012 / 2013
SESSION 1 DE PRINTEMPS

PARCOURS / ÉTAPE : LSTS / L2 **CODE UE : J1IN4001**
Épreuve : Architecture des ordinateurs (INF 155)
Date : 13 mai 2013 **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)

On considère le circuit donné par la table suivante :

E_0	E_1	E_2	E_3	S_1	S_0
1	0	0	0	0	0
x	1	0	0	0	1
x	x	1	0	1	0
x	x	x	1	1	1

où E_0, E_1, E_2 et E_3 sont les entrées, et S_0 et S_1 les sorties. La valeur x signifie "sans importance", c'est-à-dire que les valeurs des sorties sont les mêmes, que x vale 0 ou 1.

(1.1) (10 points)

Pour cette question, le comportement du circuit lorsque les quatre entrées sont simultanément nulles est sans importance. Au moyen de tables de Karnaugh, simplifiez l'écriture des fonctions S_0 et S_1.

(1.2) (5 points)

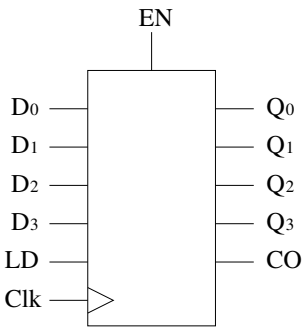
On ajoute une sortie Z qui vaut 1 si les quatre entrées sont nulles, et 0 sinon. Donnez une écriture simple de cette fonction.

(1.3) (5 points)

En utilisant des portes ET, OU et/ou NON, dessinez le schéma du circuit permettant de calculer S_0, S_1 et Z à partir de E_0, E_1, E_2 et E_3. *N.B. : Ne perdez pas de temps à construire un circuit horriblement compliqué qui ne vous rapporterait aucun point.*

Question 2 (40 points)

Un compteur sur 4 bits possède 4 sorties Q_3, Q_2, Q_1, Q_0 qui sont interprétées comme les chiffres binaires codant un entier naturel n. Le circuit compteur dessiné ci-contre fonctionne de la façon suivante :



- à chaque top d'horloge, n est incrémenté. Par exemple, si $Q_3Q_2Q_1Q_0 = 1011$ ($n = 11$ en décimal), au top d'horloge suivant, on aura $Q_3Q_2Q_1Q_0 = 1100$ ($n = 12$) ;
- l'incrémentation est circulaire, c'est-à-dire « modulo 16 ». Lorsque la valeur 15 est atteinte, la valeur suivante est 0 ;
- le compteur possède un signal d'entrée LD (pour « *Load* »). Lorsque $LD = 0$, le compteur fonctionne normalement (n sera incrémenté au prochain top d'horloge). Lorsque $LD = 1$, le compteur fonctionne comme un registre : quatre signaux d'entrée D_3, D_2, D_1 et D_0 définissent la nouvelle valeur de n au prochain top d'horloge ;
- un signal de sortie CO (pour « *Counter Out* ») indique si le compteur a atteint sa valeur maximale, 15. Dans ce cas, et dans ce cas seulement, $CO = 1$.
- un signal d'entrée EN (pour « *Enable* ») sert à activer ($EN = 1$) ou désactiver ($EN = 0$) le compteur. Lorsque EN est désactivé, la valeur de n reste constante à chaque top d'horloge.

(2.1) (10 points)

On relie la sortie CO à l'entrée LD et on maintient en permanence $D_3D_2D_1D_0 = 0110$. Expliquez quelles sont les valeurs successives de n (la valeur initiale importe peu).

(2.2) (10 points)

On construit un circuit combinatoire calculant la fonction $X = \bar{Q}_3Q_2Q_1\bar{Q}_0$. On relie la sortie X de ce circuit à l'entrée LD et on maintient en permanence $D_3D_2D_1D_0 = 0000$. Expliquez quelles sont les valeurs successives de n (la valeur initiale importe peu).

(2.3) (10 points)

Expliquez comment assembler deux compteurs à 4 bits pour construire un compteur à 8 bits (qui compte donc circulairement de 0 à 255), et tracez le schéma correspondant. *N.B. : On ne veut pas avoir de dérive de l'horloge.*

(2.4) (10 points)

Vous disposez d'une vieille version du circuit compteur ci-dessus, qui ne possède pas de signal d'entrée EN. Montrer comment, à partir d'un tel circuit et d'autres blocs et/ou portes logiques, créer un circuit amélioré mettant en œuvre ce signal EN, et tracez le schéma correspondant. *N.B. : On ne veut pas avoir de dérive de l'horloge.*

Question 3 (20 points)

Pour les fonctions demandées ci-dessous, vous utiliserez le passage de paramètres standard sur la pile, mais sans utiliser `ebp`, sauf si cela est absolument nécessaire. On rappelle que le processeur Y86 ne dispose que des opérations `addl`, `subl`, `andl` et `xorl`.

(3.1) (10 points)

Écrivez une fonction `not` qui inverse chaque bit de son unique paramètre.

(3.2) (10 points)

Écrivez une fonction `neg`, qui transforme son unique paramètre x en $-x$ **sans utiliser l'instruction** `subl`.

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, 70, 30, 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 (90 points)

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

(5.1) (10 points)

Examinez le programme ci-contre. Que fait-il ? Que vaut la variable `res` une fois l'exécution terminée ? *N.B. : Détaillez bien votre réponse.*

```
.pos 0
      irmovl 200,%esp
      mrmovl size,%ecx
      xorl   %eax,%eax
      rrmovl %eax,%esi
loop: pushl  %eax
      mrmovl t(%esi),%edx
      subl   %edx,%eax
      popl   %eax
      jge    next
      rrmovl %edx,%eax
next: iaddl  4,%esi
      isubl  1,%ecx
      jne    loop
      rmmovl %eax,res
      halt
.pos 100
t:    .long 12
      .long 5
      .long 20
      .long 8
size: .long 4
res:  .long 0
```

(5.2) (10 points)

Montrez, à l'aide d'un chronogramme, la progression des instructions dans les différents étages du pipe-line, à partir de l'instruction `pushl` de l'étiquette `loop` jusqu'à l'instruction `jge next` incluse. On suppose que le comportement du pipe-line est donné par le fichier HCL joint en fin de sujet. Justifiez votre réponse.

(5.3) (10 points)

Comment peut-on optimiser l'exécution du fragment de code étudié dans la question précédente ? Proposez une nouvelle écriture de ce fragment. Donnez le chronogramme associé.

(5.4) (15 points)

On se propose d'introduire une nouvelle instruction dans le processeur Y86 : `cmpl` ***ra,rb*** (« *Compare Long* »). Cette instruction, qui prend deux registres en arguments, effectue l'équivalent de `subl ra,rb`, excepté qu'elle ne modifie pas la valeur de ***rb***. Le seul effet de cette instruction est donc de modifier les indicateurs des codes de conditions (notamment *Zero Flag* et *Sign Flag*).

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 `CMPL`).

N.B. : N'oubliez pas d'inscrire votre numéro d'anonymat sur la feuille avant de l'insérer dans votre copie !

(5.5) (10 points)

On se propose d'introduire une autre instruction nouvelle dans le processeur Y86 : `negl` ***ra*** (« *Negate Long* »). Cette instruction renvoie l'opposé de la valeur du registre qui est son seul argument.

Ecrivez directement sur la feuille HCL jointe les ajouts que vous proposez pour traiter cette nouvelle instruction (de code `NEGL`).

(5.6) (15 points)

On se propose d'introduire deux autres instructions nouvelles dans le processeur Y86 : `incl` ***ra*** (« *Increment Long* ») et `decl` ***ra*** (« *Decrement Long* »). Ces instructions incrémentent et décrémentent le registre qui est leur seul argument.

Ecrivez directement sur la feuille HCL jointe les ajouts que vous proposez pour traiter ces nouvelles instructions (de codes `INCL` et `DECL`).

(5.7) (10 points)

Quelle(s) optimization(s) pouvez-vous réaliser afin de consommer le moins d'opcodes nouveaux possibles pour ces deux nouvelles instructions `INCL` et `DECL` ?

(5.8) (10 points)

Réécrivez de façon optimisée le programme ci-contre grâce aux nouvelles instructions ajoutées. De combien d'octets la taille de la boucle principale a-t-elle été réduite ? Justifiez votre réponse.

— 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 };
```