université de BORDEAUX DEVUIP Service Scolarité	**ANNÉE UNIVERSITAIRE 2013 – 2014** **SESSION 1 DE PRINTEMPS** **PARCOURS / ÉTAPE : LSTS / L2** **CODE UE : J1IN4001** **Épreuve :** Architecture des ordinateurs (INF 155) **Date :** 13 mai 2014 **Heure :** 8h30 **Durée :** 3h Documents : non autorisés Épreuve de M./Mme : F. Pellegrini	Département DL Licence

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 (30 points)

(1.1) (10 points)

Expliquer et montrer en quoi les portes NAND sont « complètes » vis-à-vis de l'algèbre booléenne. *(15 lignes maximum, place des schémas éventuels comprise)*

On considère un dispositif permettant de comparer deux nombres à deux bits signés $X = x_1x_0$ et $Y = y_1y_0$ codés en notation « complément à deux ». Le bit 1 est le bit de poids fort, et le bit 0 est le bit de poids faible. Le comparateur renvoie 1 si $X \geq Y$, et 0 sinon.

(1.2) (10 points)

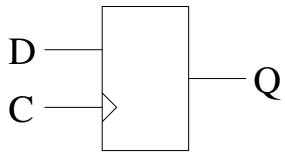
À quelles valeurs entières signées les valeurs binaires 00, 01, 10 et 11 correspondent-elles ?

(1.3) (10 points)

Écrivez la table de Karnaugh de la fonction comparateur décrite ci-dessus ou bien, à votre convenance, donnez le vecteur de vérité à 16 bits décrivant cette fonction logique. Déduisez-en une formule simplifiée de cette fonction.

Question 2 (20 points)

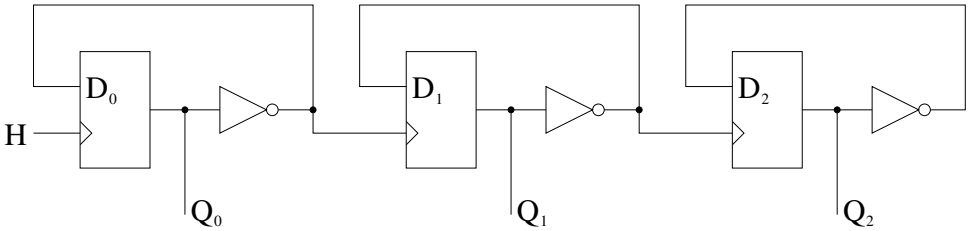
Les bascules D sont des circuits logiques à mémoire disposant de trois fils de contrôle : la sortie Q fournit en permanence l'état binaire (0 ou 1) mémorisé par le circuit, l'entrée D (pour « *data* ») permet de fournir au circuit la nouvelle valeur à mémoriser, celle-ci n'étant prise en compte que quand 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)

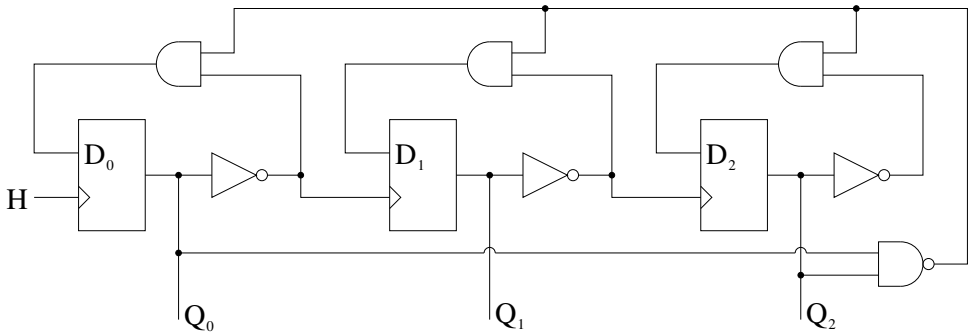
Soit le circuit suivant :



Tracer un chronogramme représentant l'évolution des états de H, Q_0, Q_1 et Q_2 au cours du temps sur onze cycles d'horloge. On supposera que l'état initial des bascules est 0, et on négligera la dérive de l'horloge. Soit Q le mot binaire à trois bits constitué des bits $Q_2Q_1Q_0$ (le bit 0 est toujours le bit de poids le plus faible). Au vu du chronogramme, quelle fonction ce circuit remplit-il ?

(2.2) (10 points)

On modifie le circuit précédent afin d'obtenir le circuit suivant :



Tracer un chronogramme représentant l'évolution des états de H, Q_0, Q_1 et Q_2 au cours du temps sur onze cycles d'horloge. On supposera que l'état initial des bascules est 0, et on négligera la dérive de l'horloge. Soit Q le mot binaire à trois bits constitué des bits $Q_2Q_1Q_0$ (le bit 0 est toujours le bit de poids le plus faible). Au vu du chronogramme, quelle fonction ce circuit remplit-il ?

Question 3 (50 points)

On veut construire un distributeur de bonbons. Celui-ci distribue un bonbon lorsque l'on a introduit 15 Brouzoufs (Bz) dans le distributeur. Le distributeur possède deux détecteurs de pièces, l'un pour les pièces de 5 Bz (fil d'entrée P5) et l'autre pour les pièces de 10 Bz (fil d'entrée P10). Le circuit de commande possède également deux sorties : la sortie D active la distribution d'un bonbon, et la sortie R active la restitution d'une pièce de 5 Bz. On dispose d'un signal d'horloge H qui servira à cadencer le dispositif.

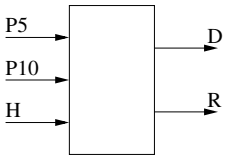
Le cahier des charges du distributeur est le suivant :

— on ne peut introduire que des pièces de 5 et 10 Bz ;

— deux pièces ne peuvent être introduites (et donc détectées) au même moment ;

— une pièce n'est détectée que pendant un seul cycle d'horloge. Entre deux insertions de pièces, les fils du détecteur de pièces retournent à l'état 0 pendant au moins un cycle.

Le circuit à construire possède donc les entrées et sorties suivantes :



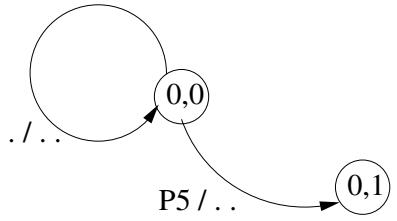
(3.1) (20 points)

En reproduisant et complétant le schéma ci-dessous sur votre feuille, dessinez le comportement attendu de la machine sous la forme d'un graphe (appelé automate) dont les sommets (dessinés sous forme de cercles) représentent les états, et les arcs (sous forme de flèches) représentent les transitions entre états.

Chaque sommet portera deux nombres, qui sont les quantités de pièces de 10 et 5 Bz déjà insérées. Chaque arc sera étiqueté par le fil d'entrée reçu et par les valeurs de sortie à produire sur les fils D et R, séparés par un « / ». Par exemple, « P10 / D R » veut dire que, sur activation du fil P10, on devra activer les fils D et R tout en allant vers l'état destination. On ne met rien après le « / » si ni D ni R ne doivent être activés. D'après le cahier des charges, les fils P5 et P10 ne peuvent être actifs en même temps.

N'oubliez pas les boucles lorsque l'utilisateur ne fait rien. Il vous est possible, au besoin, d'insérer des états intermédiaires pour envoyer certains signaux avant de passer à un autre état.

Bien évidemment, une fois que la machine a distribué le bonbon et rendu la monnaie éventuelle, elle doit être prête à fonctionner à nouveau.



(3.2) (10 points)

Numérotez les états de votre automate, en partant de 0. Votre numérotation devra respecter l'ordre croissant des couples de valeurs (P10,P5) contenues dans chaque cercle. Par exemple, le numéro choisi pour (1,0) devra être plus grand que celui choisi pour (0,1). Indiquez sur votre figure les numéros choisis pour chaque état, à côté de chaque cercle.

De combien de bits de registre devez-vous disposer pour mémoriser l'état courant de la machine à un moment donné ? On appellera E ce nombre, codé par les bits e_0, e_1, etc. (le bit 0 étant le bit de poids faible).

En vous appuyant sur l'automate précédent, faites une table qui, en fonction du codage binaire du numéro de l'état courant et de la valeur d'entrée des fils P5 et P10, donne le numéro de l'état suivant et la valeur des fils D et R.

(3.3) (10 points)

Proposez le câblage d'un circuit, à base de bascules D et de multiplexeurs, qui permette de construire le dispositif ci-dessus. De combien de multiplexeurs aurez-vous besoin ? On ne demande pas de représenter le câblage de toutes les entrées individuelles de chacun des multiplexeurs, mais d'en donner l'idée générale.

(3.4) (10 points)

Refaites le dessin de l'automate, dans le cas où le distributeur peut également accepter des pièces de 20 Bz, détéctées au moyen d'un fil d'entrée supplémentaire P20. Le retour de la monnaie se fera encore uniquement au moyen de pièces de 5 Bz.

Question 4 (20 points)

Écrivez le code Y86 correspondant aux fonctions C ci-contre. Vous disposez d'une fonction `mul` qui prend comme arguments deux valeurs, et renvoie le résultat de leur multiplication. Expliquez comment vous mettez en œuvre le passage de paramètres par la pile, ainsi que les conventions «*caller save* / *callee save* » des registres.

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

Question 5 (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 90, 60, 20, 50 et 40 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.

(5.1) (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.

(5.2) (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.

(5.3) (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 6 (70 points)

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

(6.1) (10 points)

Qu'est-ce qu'un « *load-use hazard* » ? Dans quel(s) cas se produisent-ils ?

(6.2) (10 points)

Examinez le programme ci-contre. Que fait-il ? Chronogramme à l'appui, indiquez combien s'écoulent de cycles entre deux itérations de boucle ; vous pouvez prendre comme référence le moment où l'instruction `mrmovl (%esi),%eax` entre dans l'étage F du pipe-line. 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, %esi
      mrmovl s, %ecx
      xorl   %edx, %edx
      irmovl 4, %ebx
boucle:
      mrmovl (%esi), %eax
      addl   %eax, %edx
      addl   %ebx, %esi
      isubl  1, %ecx
      jne    boucle
      halt
.pos  100
s:    .long  5
t:    .long  2
      .long  3
      .long  5
      .long  7
      .long  11
```

(6.3) (10 points)

En vertu des dépendances entre instructions, est-il possible, en modifiant le code Y86, de consommer moins de cycles par tour de boucle ? Justifiez votre réponse.

(6.4) (10 points)

On se propose d'introduire une nouvelle instruction dans le processeur Y86 : `LODSL` (« *Load String Long* »). Cette instruction (sans argument) effectue l'équivalent de `mrmovl (%esi),%eax`, tout en ajoutant 4 au registre `%esi`, la valeur de `%esi` utilisée par `mrmovl` 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 `LODSL`).

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

(6.5) (10 points)

On souhaite ajouter à notre 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 ? Est-ce compatible avec l'architecture actuelle ? Justifiez votre réponse.

Quelles modifications éventuelles proposez-vous au chemin de données du processeur pour supporter cette instruction ?

(6.6) (10 points)

Comme pour l'instruction `LODSL`, 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 éventuelle du chemin de données.

— 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 };
```