université de BORDEAUX	**ANNÉE UNIVERSITAIRE 2015 – 2016** **SESSION 1 DE PRINTEMPS** **Parcours / Étape : LSTS / L2** **Code UE : J1IN4001** **Épreuve :** Architecture des ordinateurs (INF 155) **Date :** 2 mai 2016 **Heure :** 14h00 **Durée :** 3h 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 (50 points)

On veut réaliser un circuit câblé effectuant la multiplication de deux nombres à 8 bits non signés $\mathtt{A} = \mathtt{a}_7\mathtt{a}_6 \ldots \mathtt{a}_1\mathtt{a}_0$ et $\mathtt{B} = \mathtt{b}_7\mathtt{b}_6 \ldots \mathtt{b}_1\mathtt{b}_0$, et fournissant un résultat sur x bits $\mathtt{P} = \mathtt{p}_{x-1}\mathtt{p}_{x-2} \cdots \mathtt{p}_1\mathtt{p}_0$.

(1.1) (5 points)

Donnez, sous la forme d'une formule utilisant les puissances de deux, la plus grande valeur possible de `A` et de `B`. Déduisez-en la plus grande valeur possible de `P`, sous la forme d'une formule utilisant les puissances de deux. Sur combien de bits x doit donc être câblé le résultat ?

(1.2) (5 points)

Donnez la table de vérité de la multiplication à un bit entre deux bits `a` et `b`. À quelle fonction logique correspond-elle ?

(1.3) (5 points)

Une multiplication en binaire s'effectue de la même façon qu'en base dix : pour chaque chiffre $\mathtt{b}_i$, du multiplicateur, on écrit un produit partiel correspondant à la multiplication de `A` par $\mathtt{b}_i$, décalé de i chiffres par rapport à l'unité.

Pour bien comprendre le principe, recopiez et effectuez sur votre feuille la multiplication ci-dessous. Certains chiffres ont été laissés afin de vous guider.

	A	=						1	0	1	1	0	0	0	1
×	B	=						1	1	0	1	0	0	1	1
	P_0	=						.	.	.	.	.	.	0	1
+	P_1	=					.	.	.	.	.	.	.	.	
+	P_2	=				.	.	.	.	.	.	.	0		
+	P_3	=			.	.	.	.	.	.	.	.			
+	P_4	=		.	.	.	.	.	.	.	.				
+	P_5	=	.	.	.	.	.	.	.	.					
+	P_6	=	.	.	.	.	.	.	.						
+	P_7	=	.	.	.	.	.	.	.						
	P	=	.	.	.	.	.	.	.	1	1	0	0	0	1	1

(1.4) (10 points)

Donnez le schéma de câblage du circuit calculant le produit partiel $\mathtt{P}_0$ sur 8 bits, en fontion de `A` et $\mathtt{b}_0$.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce circuit sous la forme d'une boîte étiquetée « P_0 », comprenant une entrée `A` sur 8 bits et une entrée $\mathtt{b}_0$ sur 1 bit, et une sortie P_0 sur 8 bits.

(1.5) (5 points)

En vous appuyant sur le circuit précédent, donnez le schéma de câblage du produit partiel $\mathtt{P}_i$ sur $(8+i)$ bits, incluant les i bits issus du décalage du produit partiel.

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce circuit sous la forme d'une boîte étiquetée « P_i », avec i compris entre 0 et 7, comprenant une entrée `A` sur 8 bits et une entrée $\mathtt{b}_i$ sur 1 bit, et une sortie P_i sur $(8+i)$ bits.

Comme on le voit, pour connaître le résultat de la multiplication, il faut effectuer de nombreuses additions entre les produits partiels $\texttt{P}_i$. Si chacune de ces additions devait nécessiter la propagation d'une retenue, le temps de calcul de la multiplication serait très long, du fait de la longueur du chemin critique. Il faut donc pouvoir effectuer de nombreuses additions sans avoir besoin de propager une retenue.

(1.6) (5 points)

Donnez la table de vérité d'un additionneur à trois bits d'entrée `x`, `y` et `z`, et deux bits de sortie `s` (somme) et `c` (retenue sortante).

En fait, lors d'une addition, on peut décider de ne pas reporter les retenues. L'additionneur « à conservation de retenue » (« *Carry Save Adder* », ou CSA) permet d'effectuer l'addition de trois nombres binaires, en conservant les retenues de chaque bit dans un mot dédié `C` de 8 bits plus 1 bit de décalage, soit 9 bits. Ainsi, si `X`, `Y` et `Z` sont trois nombres codés sur 8 bits, on aura :

$$\begin{array}{rcccccccccc} & X & = & & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ + & Y & = & & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ + & Z & = & & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline & S & = & & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ & C & = & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & \end{array}$$

On a ainsi transformé le problème de l'addition de trois nombres en l'addition de deux nombres, sans avoir eu besoin de propager une retenue !

(1.7) (5 points)

Si l'on utilise un additionneur CSA pour additionner trois produits partiels de f, g et h bits respectivement, par quelles valeurs minimales et maximales pouvez-vous borner les tailles s et c des mots `S` et `C` ?

À partir de maintenant, et même si vous n'avez pas répondu aux questions précédentes, vous pouvez représenter ce circuit sous la forme d'une boîte étiquetée « `CSA` », comprenant trois entrées `X`, `Y` et `Z` de tailles variables, et deux sorties `S` et `C` de tailles variables également.

(1.8) (10 points)

En vous appuyant sur des circuits CSA, ainsi que sur un seul additionneur « classique » à propagation de retenue que vous représenterez sous la forme d'une boîte « + » munie de deux entrées d'une sortie sur x bits chacune, construisez un circuit permettant d'obtenir la valeur de `P` à partir des huit valeurs partielles $\texttt{P}_0$ à $\texttt{P}_7$, et minimisant la longueur du chemin critique.

Question 2 (40 points)

On considère l'algorithme suivant, qui calcule le PGCD de deux nombres entiers signés positifs a et b :

```
int  a, b;
a = ...;
b = ...;
while (a != b) {
  if ((a - b) > 0)
    a = (a - b);
  else
    b = (b - a);
}
```

À la fin de l'algorithme, le résultat peut être lu dans `a` ou dans `b`, qui sont égaux.

On veut implanter cet algorithme au sein d'un circuit numérique synchrone, c'est-à-dire piloté par une horloge, à mots de 32 bits. Pour cela, vous allez devoir construire et assembler les briques logiques nécessaires. Vous avez à votre disposition :

— des registres sur 32 bits, comprenant une entrée `D` sur 32 bits, une entrée `CK` (« *clock* ») sur un bit activant le registre sur front montant, et une sortie `Q` sur 32 bits ;

— des multiplexeurs 32×1, comprenant deux entrées $\texttt{E}_0$ et $\texttt{E}_1$ sur 32 bits, une entrée `C` sur un bit et une sortie `S` sur 32 bits ;

— des multiplexeurs 32×2, comprenant quatre entrées $\mathtt{E}_{00}$, $\mathtt{E}_{01}$, $\mathtt{E}_{10}$ et $\mathtt{E}_{11}$ sur 32 bits, deux entrées $\mathtt{C}_0$ et $\mathtt{C}_1$ sur un bit chacune, et une sortie `S` sur 32 bits ;

— des circuits additionneurs sur 32 bits, comprenant deux entrées `A` et `B` sur 32 bits, une entrée $\mathtt{C}_{\mathtt{in}}$ (« *carry in* », c'est-à-dire « retenue d'entrée », dont la valeur sera ajoutée au calcul de l'addition sur les unités) sur un bit, une sortie `R` (résultat) sur 32 bits, et trois sorties `Z` (résultat nul), `S` (résultat négatif) et `C` (retenue générée en sortie), sur un bit chacune.

Chacun peut être représenté sous la forme d'une boîte disposant des fils indiqués.

(2.1) (10 points)

En notation « complément à deux », comment calcule-t-on l'opposé d'un nombre ? Déduisez-en une façon de construire un circuit soustracteur à partir du circuit additionneur décrit ci-dessus.

À partir de maintenant, et même si vous n'avez pas répondu à la question précédente, vous avez à votre disposition un circuit soustracteur sur 32 *bits, comprenant deux entrées* `A` *et* `B` *sur* 32 *bits, une sortie* $\mathtt{R} = \mathtt{A} - \mathtt{B}$ *sur* 32 *bits, et trois sorties* `Z` *(résultat nul),* `S` *(résultat négatif) et* `C` *(retenue générée en sortie), sur un bit chacune.*

(2.2) (15 points)

À chaque tour de boucle de l'algorithme, c'est-à-dire à chaque front montant de `CK`, on doit placer dans le registre `a` :

— une valeur `A` sur 32 bits fournie en entrée, si un fil de contrôle `L` (« *load* ») est égal à 1 ;

— la valeur $(\mathtt{a} - \mathtt{B})$, où `a` est l'ancienne valeur du registre, si le résultat de cette soustraction est strictement supérieur à zéro (*pour le moment, supposez que* `B` *est une valeur sur* 32 *bits fournie en entrée depuis l'extérieur du circuit*) ;

— la valeur préexistante du registre `a`, si le résultat de la soustraction précédente est nul (fin de l'algorithme) ou strictement inférieur à zéro (cas du « `else` »).

Construisez un circuit conforme aux spécifications ci-dessus. Explicitez les fonctions logiques que vous mettez en œuvre (table de vérité, câblage) pour le pilotage de la mise à jour de la valeur du registre. Faites preuve d'astuce, pour que ces fonctions soient simples.

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

(2.3) (15 points)

En vous inspirant du circuit précédent, construisez un circuit synchrone calculant le PGCD, piloté par un fil d'horloge `CK` et un fil de contrôle `L` (« *load* »), de telle sorte que :

— si $\mathtt{L} = 1$, les registres `a` et `b` du circuit sont chargés avec les valeurs d'entrée `A` et `B` fournies en entrée ;

— si $\mathtt{L} = 0$, le circuit modifie les valeurs de `a` et `b` à l'image de l'exécution d'un tour de boucle de l'algorithme présenté en début de problème ;

— lorsque le calcul est terminé, le fil de sortie `T` du circuit passe à 1 et les valeurs ne sont plus modifiées.

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

Question 3 (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 60, 40, 30, 70 et 20 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.

(3.1) (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.

(3.2) (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.

(3.3) (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 4 (80 points)

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

(4.1) (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   %esi,%esi
      mrmovl t(%esi),%eax
loop: iaddl  4,%esi
      isubl  1,%ecx
      je     endl
      pushl  %eax
      mrmovl t(%esi),%edx
      subl   %edx,%eax
      popl   %eax
      jle    loop
      rrmovl %edx,%eax
      jmp    loop
endl: rmmovl %eax,res
      halt
.pos 100
t:    .long 8
      .long 12
      .long 5
      .long 20
size: .long 4
res:  .long 0
```

(4.2) (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 `jmp loop` incluse. On suppose que le comportement du pipe-line est donné par le fichier HCL joint en fin de sujet. Justifiez votre réponse.

(4.3) (10 points)

Peut-on optimiser l'exécution du fragment de code étudié dans la question précédente ? Si oui, proposez une nouvelle écriture de ce fragment et donnez le chronogramme associé. Si non, justifiez.

(4.4) (10 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 !

(4.5) (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`).

(4.6) (10 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`).

(4.7) (10 points)

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

(4.8) (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 };
```