NOM, Prénom, numéro d'étudiant, groupe :

CC-TP - sujet A - LIF6 - vendredi 20/5/16 - noté sur 10 - durée : 1h50

Aucun document autorisé, sauf l'aide de Pennsim et celle de Logisim. Vous devez travailler seul : aucune communication n'est autorisée. Les fichiers de travail sont à l'adresse `http://perso.ens-lyon.fr/nicolas.louvet/tp/` Rendez le sujet avec vos réponses écrites au dos. Pour rendre vos fichiers :

- connectez-vous dans TOMMUS (`http://tomuss.univ-lyon1.fr`),
- dans la partie « UE-INF2003L Lif6... » cliquez sur le lien « déposer » de la case « exo**i** » pour déposer votre fichier de réponse à l'exercice **i** (**i** $\in \{1, 2\}$).
- vérifiez que vos fichiers ont bien été déposés en les téléchargeant...

Exercice 1 (Programmation dans la langage du LC3 avec Pennsim). Téléchargez le fichier `dedoub.asm`, et éditez-le. Dans ce fichier, vous devez compléter la routine `dedoub` (voir ci-dessous), et modifier le programme principal de façon à ce que l'appel à `strcpy` soit remplacé par l'appel à `dedoub` ; vous adapterez aussi le commentaire de cet appel en conséquence. Vous n'avez rien d'autre à modifier dans le fichier !

Pour compléter la routine `dedoub`, vous suivrez les commentaires et le pseudo-code donné dans le fichier. Notez que les commentaires vous permettent de savoir quel est le comportement attendu de votre routine.

Notez que dans le fichier que vous avez téléchargé, le programme principal vous permet déjà de tester votre routine. Toutes les chaînes de caractères doivent se terminer par le caractère de code ASCII 0 (lire « zéro »). Voici le tableau des instructions du LC3 dont vous pouvez avoir besoin :

syntaxe	**action**	**NZP**
NOT DR,SR	DR ← not SR	*
ADD DR,SR1,SR2	DR ← SR1 + SR2	*
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*
AND DR,SR1,SR2	DR ← SR1 and SR2	*
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*
LEA DR,label	DR ← PC + SEXT(PCoffset9)	*
LD DR,label	DR ← mem[PC + SEXT(PCoffset9)]	*
ST SR,label	mem[PC + SEXT(PCoffset9)] ← SR	
LDR DR,BaseR,Offset6	DR ← mem[BaseR + SEXT(Offset6)]	*
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] ← SR	
BR[n][z][p] label	Si (cond) PC ← PC + SEXT(PCoffset9)	
RET	PC ← R7	
JSR label	R7 ← PC ; PC ← PC + SEXT(PCoffset11)	

Exercice 2 (cicuits avec Logisim - rotation à gauche). Vous travaillerez dans le fichier `rotg.circ`. On veut ajouter une instruction `ROTG` de rotation à gauche dans le jeu d'instructions du LC3, et on veut un circuit combinatoire pour cela. On rappelle que les registres sont sur 16 bits, les bits étant numérotés de 0 à 15 (le bit de poids fort est le bit numéro 15).

Une rotation à gauche permute circulairement les bits d'une position vers la gauche. Le bit 15 prend la valeur du bit 14, le bit 14 celle du bit 13, ..., le bit 1 la valeur du bit 0 et le bit 0 la valeur du bit 15 avant la rotation. Une rotation à gauche de k positions consiste à effectuer k rotations à gauche.

1. Si `R1` = (0110001111111110), quel est le résultat d'une rotation à gauche de `R1` ? De trois ?
2. Complétez le circuit `rotg` pour qu'il effectue une rotation à gauche de son entrée `in` sur sa sortie `out`. Vous utiliserez les composants `splitter` de logisim (surtout pas `shifter`) en remarquant qu'une rotation à gauche est juste une copie de bits.
3. Complétez le circuit `rotgsel`, de façon à ce qu'il effectue une rotation à gauche uniquement si son entrée `sel` est à 1. Si `sel` est à 0, le circuit se contente de recopier son entrée `in` sur sa sortie `out`.
4. On vous fournit le circuit combinatoire `cumdecod` : en essayant différentes valeurs pour l'entrée `k` du circuit, étudiez son fonctionnement. Quelles sorties de `cumdecod` sont activées pour chaque valeur de `k`, en regardant `k` comme un entier naturel sur 3 bits ?
5. Complétez le circuit `krotg` ; ce circuit prendra une entrée `in` sur 16 bits, une entrée `k` sur 3 bits, et produira une sortie `out` sur 16 bits : `out` sera le résultat de `k` rotations à gauche sur `in`. Vous utiliserez `cumdecod`, ainsi que 7 fois `rotgsel`.

6. En plus des opérations habituelles (`ADD`, `AND`, `NOT`) on veut que l'ALU puisse effectuer une rotation de k bits à gauche de `in1` (k est fourni soit par les 3 bits de poids faibles `in2`, soit par ceux de `cste`). Complétez le circuit `alu` de façon à ce qu'il produise `out` comme indiqué ci-dessous.

	`UseCste`	
e2	0	1
00	`out` ← `AND` `in1`, `in2`	`out` ← `AND` `in1`, `cste`
01	`out` ← `ROTG` `in1`, `in2`	`out` ← `ROTG` `in1`, `cste`
10	`out` ← `ADD` `in1`, `in2`	`out` ← `ADD` `in1`, `cste`
11	`out` ← `NOT` `in1`	`out` ← `NOT` `in1`

NOM, Prénom, numéro d'étudiant, groupe :

CC-TP - sujet B - LIF6 - vendredi 20/5/16 - noté sur 10 - durée : 1h50

Aucun document autorisé, sauf l'aide de Pennsim et celle de Logisim. Vous devez travailler seul : aucune communication n'est autorisée. Les fichiers de travail sont à l'adresse `http://perso.ens-lyon.fr/nicolas.louvet/tp/`
Rendez le sujet avec vos réponses écrites au dos. Pour rendre vos fichiers :

- connectez-vous dans TOMMUS (`http://tomuss.univ-lyon1.fr`),
- dans la partie « UE-INF2003L Lif6… » cliquez sur le lien « déposer » de la case « exo**i** » pour déposer votre fichier de réponse à l'exercice **i** (**i** $\in \{1, 2\}$).
- vérifiez que vos fichiers ont bien été déposés en les téléchargeant…

Exercice 1 (Programmation dans la langage du LC3 avec Pennsim). Téléchargez le fichier `suppdbl.asm`, et éditez-le. Dans ce fichier, vous devez compléter la routine `suppdbl` (voir ci-dessous), et modifier le programme principal de façon à ce que l'appel à `strcpy` soit remplacé par l'appel à `suppdbl` ; vous adapterez aussi le commentaire de cet appel en conséquence. Vous n'avez rien d'autre à modifier dans le fichier !

Pour compléter la routine `suppdbl`, vous suivrez les commentaires et le pseudo-code donné dans le fichier. Notez que les commentaires vous permettent de savoir quel est le comportement attendu de votre routine.

Notez que dans le fichier que vous avez téléchargé, le programme principal vous permet déjà de tester votre routine. Toutes les chaînes de caractères doivent se terminer par le caractère de code ASCII 0 (lire « zéro »). Voici le tableau des instructions du LC3 dont vous pouvez avoir besoin :

syntaxe	**action**	**NZP**
NOT DR,SR	DR ← not SR	*
ADD DR,SR1,SR2	DR ← SR1 + SR2	*
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*
AND DR,SR1,SR2	DR ← SR1 and SR2	*
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*
LEA DR,label	DR ← PC + SEXT(PCoffset9)	*
LD DR,label	DR ← mem[PC + SEXT(PCoffset9)]	*
ST SR,label	mem[PC + SEXT(PCoffset9)] ← SR	
LDR DR,BaseR,Offset6	DR ← mem[BaseR + SEXT(Offset6)]	*
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] ← SR	
BR[n][z][p] label	Si (cond) PC ← PC + SEXT(PCoffset9)	
RET	PC ← R7	
JSR label	R7 ← PC ; PC ← PC + SEXT(PCoffset11)	

Exercice 2 (cicuits avec Logisim - décalage à droite). Vous travaillerez dans le fichier `decd.circ`. On veut ajouter une instruction `DECD` de décalage à droite dans le jeu d'instructions du LC3, et on veut un circuit combinatoire pour cela. On rappelle que les registres sont sur 16 bits, les bits étant numérotés de 0 à 15 (le bit de poids fort est le bit numéro 15).

Un décalage à droite décale les bits d'une position vers la droite. Le bit 15 prend la valeur 0, le bit 14 celle du bit 15, …, le bit 1 la valeur du bit 2 et le bit 0 la valeur du bit 1 avant le décalage. Un décalage à droite de k positions consiste à effectuer k décalages à droite.

1. Si `R1` = (0110101010101010), quel est le résultat d'un décalage à droite de `R1` ? De trois ?
2. Complétez le circuit `decd` pour qu'il effectue un décalage à droite de son entrée `in` sur sa sortie `out`. Vous utiliserez les composants `splitter` de logisim (surtout pas `shifter`) en remarquant qu'un décalage à droite est juste une copie de bits.
3. Complétez le circuit `decdsel`, de façon à ce qu'il effectue un décalage à droite uniquement si son entrée `sel` est à 1. Si `sel` est à 0, le circuit se contente de recopier son entrée `in` sur sa sortie `out`.
4. On vous fournit le circuit combinatoire `cumdecod` : en essayant différentes valeurs pour l'entrée `k` du circuit, étudiez son fonctionnement. Quelles sorties de `cumdecod` sont activées pour chaque valeur de `k`, en regardant `k` comme un entier naturel sur 3 bits ?
5. Complétez le circuit `kdecd` ; ce circuit prendra une entrée `in` sur 16 bits, une entrée `k` sur 3 bits, et produira une sortie `out` sur 16 bits : `out` sera le résultat de `k` décalages à droite sur `in`. Vous utiliserez `cumdecod`, ainsi que 7 fois `decdsel`.

6. En plus des opérations habituelles (`ADD`, `AND`, `NOT`) on veut que l'ALU puisse effectuer un décalage de k bits à droite de `in1` (k est fourni soit par les 3 bits de poids faibles `in2`, soit par ceux de `cste`). Complétez le circuit `alu` de façon à ce qu'il produise `out` comme indiqué ci-dessous.

	`UseCste`	
e2	0	1
00	`out` ← `AND` `in1`, `in2`	`out` ← `AND` `in1`, `cste`
01	`out` ← `ADD` `in1`, `in2`	`out` ← `ADD` `in1`, `cste`
10	`out` ← `DECD` `in1`, `in2`	`out` ← `DECD` `in1`, `cste`
11	`out` ← `NOT` `in1`	`out` ← `NOT` `in1`

NOM, Prénom, numéro d'étudiant, groupe :

CC-TP - sujet C - LIF6 - vendredi 20/5/16 - noté sur 10 - durée : 1h50

Aucun document autorisé, sauf l'aide de Pennsim et celle de Logisim. Vous devez travailler seul : aucune communication n'est autorisée. Les fichiers de travail sont à l'adresse `http://perso.ens-lyon.fr/nicolas.louvet/tp/` Rendez le sujet avec vos réponses écrites au dos. Pour rendre vos fichiers :

- connectez-vous dans TOMMUS (`http://tomuss.univ-lyon1.fr`),
- dans la partie « UE-INF2003L Lif6… » cliquez sur le lien « déposer » de la case « exo**i** » pour déposer votre fichier de réponse à l'exercice **i** (**i** $\in \{1, 2\}$).
- vérifiez que vos fichiers ont bien été déposés en les téléchargeant…

Exercice 1 (Programmation dans la langage du LC3 avec Pennsim). Téléchargez le fichier `subs.asm`, et éditez-le. Dans ce fichier, vous devez compléter la routine `subs` (voir ci-dessous), et modifier le programme principal de façon à ce que l'appel à `strcpy` soit remplacé par l'appel à `subs` ; vous adapterez aussi le commentaire de cet appel en conséquence. Vous n'avez rien d'autre à modifier dans le fichier !

Pour compléter la routine `subs`, vous suivrez les commentaires et le pseudo-code donné dans le fichier. Notez que les commentaires vous permettent de savoir quel est le comportement attendu de votre routine.

Notez que dans le fichier que vous avez téléchargé, le programme principal vous permet déjà de tester votre routine. Toutes les chaînes de caractères doivent se terminer par le caractère de code ASCII 0 (lire « zéro »). Voici le tableau des instructions du LC3 dont vous pouvez avoir besoin :

syntaxe	**action**	**NZP**
NOT DR,SR	DR ← not SR	*
ADD DR,SR1,SR2	DR ← SR1 + SR2	*
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*
AND DR,SR1,SR2	DR ← SR1 and SR2	*
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*
LEA DR,label	DR ← PC + SEXT(PCoffset9)	*
LD DR,label	DR ← mem[PC + SEXT(PCoffset9)]	*
ST SR,label	mem[PC + SEXT(PCoffset9)] ← SR	
LDR DR,BaseR,Offset6	DR ← mem[BaseR + SEXT(Offset6)]	*
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] ← SR	
BR[n][z][p] label	Si (cond) PC ← PC + SEXT(PCoffset9)	
RET	PC ← R7	
JSR label	R7 ← PC ; PC ← PC + SEXT(PCoffset11)	

Exercice 2 (cicuits avec Logisim - décalage à gauche). Vous travaillerez dans le fichier `decg.circ`. On veut ajouter une instruction `DECG` de décalage à gauche dans le jeu d'instructions du LC3, et on veut un circuit combinatoire pour cela. On rappelle que les registres sont sur 16 bits, les bits étant numérotés de 0 à 15 (le bit de poids fort est le bit numéro 15).

Un décalage à gauche décale les bits d'une position vers la gauche. Le bit 15 prend la valeur du bit 14, le bit 14 celle du bit 13, …, le bit 1 la valeur du bit 0 et le bit 0 la valeur 0. Un décalage à gauche de k positions consiste à effectuer k décalages à gauche.

1. Si `R1` = (0110101011111010), quel est le résultat d'un décalage à gauche de `R1` ? De trois ?
2. Complétez le circuit `decg` pour qu'il effectue un décalage à gauche de son entrée `in` sur sa sortie `out`. Vous utiliserez les composants `splitter` de logisim (surtout pas `shifter`) en remarquant qu'un décalage à gauche est juste une copie de bits.
3. Complétez le circuit `decgsel`, de façon à ce qu'il effectue un décalage à gauche uniquement si son entrée `sel` est à 1. Si `sel` est à 0, le circuit se contente de recopier son entrée `in` sur sa sortie `out`.
4. On vous fournit le circuit combinatoire `cumdecod` : en essayant différentes valeurs pour l'entrée `k` du circuit, étudiez son fonctionnement. Quelles sorties de `cumdecod` sont activées pour chaque valeur de `k`, en regardant `k` comme un entier naturel sur 3 bits ?
5. Complétez le circuit `kdecg` ; ce circuit prendra une entrée `in` sur 16 bits, une entrée `k` sur 3 bits, et produira une sortie `out` sur 16 bits : `out` sera le résultat de `k` décalages à gauche sur `in`. Vous utiliserez `cumdecod`, ainsi que 7 fois `decgsel`.

6. En plus des opérations habituelles (`ADD`, `AND`, `NOT`) on veut que l'ALU puisse effectuer un décalage de k bits à gauche de `in1` (k est fourni soit par les 3 bits de poids faibles `in2`, soit par ceux de `cste`). Complétez le circuit `alu` de façon à ce qu'il produise `out` comme indiqué ci-dessous.

	`UseCste`	
e2	0	1
00	`out` ← `AND` `in1`, `in2`	`out` ← `AND` `in1`, `cste`
01	`out` ← `ADD` `in1`, `in2`	`out` ← `ADD` `in1`, `cste`
10	`out` ← `NOT` `in1`	`out` ← `NOT` `in1`
11	`out` ← `DECG` `in1`, `in2`	`out` ← `DECG` `in1`, `cste`

NOM, Prénom, numéro d'étudiant, groupe :

CC-TP - sujet D - LIF6 - vendredi 20/5/16 - noté sur 10 - durée : 1h50

Aucun document autorisé, sauf l'aide de Pennsim et celle de Logisim. Vous devez travailler seul : aucune communication n'est autorisée. Les fichiers de travail sont à l'adresse `http://perso.ens-lyon.fr/nicolas.louvet/tp/` Rendez le sujet avec vos réponses écrites au dos. Pour rendre vos fichiers :

- connectez-vous dans TOMMUS (`http://tomuss.univ-lyon1.fr`),
- dans la partie « UE-INF2003L Lif6… » cliquez sur le lien « déposer » de la case « exo**i** » pour déposer votre fichier de réponse à l'exercice **i** (**i** $\in \{1, 2\}$).
- vérifiez que vos fichiers ont bien été déposés en les téléchargeant…

Exercice 1 (Programmation dans la langage du LC3 avec Pennsim). Téléchargez le fichier `rmv.asm`, et éditez-le. Dans ce fichier, vous devez compléter la routine `rmv` (voir ci-dessous), et modifier le programme principal de façon à ce que l'appel à `strcpy` soit remplacé par l'appel à `rmv` ; vous adapterez aussi le commentaire de cet appel en conséquence. Vous n'avez rien d'autre à modifier dans le fichier !

Pour compléter la routine `rmv`, vous suivrez les commentaires et le pseudo-code donné dans le fichier. Notez que les commentaires vous permettent de savoir quel est le comportement attendu de votre routine.

Notez que dans le fichier que vous avez téléchargé, le programme principal vous permet déjà de tester votre routine. Toutes les chaînes de caractères doivent se terminer par le caractère de code ASCII 0 (lire « zéro »). Voici le tableau des instructions du LC3 dont vous pouvez avoir besoin :

syntaxe	**action**	**NZP**
NOT DR,SR	DR ← not SR	*
ADD DR,SR1,SR2	DR ← SR1 + SR2	*
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*
AND DR,SR1,SR2	DR ← SR1 and SR2	*
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*
LEA DR,label	DR ← PC + SEXT(PCoffset9)	*
LD DR,label	DR ← mem[PC + SEXT(PCoffset9)]	*
ST SR,label	mem[PC + SEXT(PCoffset9)] ← SR	
LDR DR,BaseR,Offset6	DR ← mem[BaseR + SEXT(Offset6)]	*
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] ← SR	
BR[n][z][p] label	Si (cond) PC ← PC + SEXT(PCoffset9)	
RET	PC ← R7	
JSR label	R7 ← PC ; PC ← PC + SEXT(PCoffset11)	

Exercice 2 (cicuits avec Logisim - rotation à droite). Vous travaillerez dans le fichier `rotd.circ`. On veut ajouter une instruction `ROTD` de rotation à droite dans le jeu d'instructions du LC3, et on veut un circuit combinatoire pour cela. On rappelle que les registres sont sur 16 bits, les bits étant numérotés de 0 à 15 (le bit de poids fort est le bit numéro 15).

Une rotation à droite permute circulairement les bits d'une position vers la droite. Le bit 0 prend la valeur du bit 1, le bit 1 celle du bit 2, …, le bit 14 la valeur du bit 15 et le bit 15 la valeur du bit 0 avant la rotation. Une rotation à droite de k positions consiste à effectuer k rotations à droite.

1. Si `R1` = (0110001111111110), quel est le résultat d'une rotation à droite de `R1` ? De trois ?
2. Complétez le circuit `rotd` pour qu'il effectue une rotation à droite de son entrée `in` sur sa sortie `out`. Vous utiliserez les composants `splitter` de logisim (surtout pas `shifter`) en remarquant qu'une rotation à droite est juste une copie de bits.
3. Complétez le circuit `rotdsel`, de façon à ce qu'il effectue une rotation à droite uniquement si son entrée `sel` est à 1. Si `sel` est à 0, le circuit se contente de recopier son entrée `in` sur sa sortie `out`.
4. On vous fournit le circuit combinatoire `cumdecod` : en essayant différentes valeurs pour l'entrée `k` du circuit, étudiez son fonctionnement. Quelles sorties de `cumdecod` sont activées pour chaque valeur de `k`, en regardant `k` comme un entier naturel sur 3 bits ?
5. Complétez le circuit `krotd` ; ce circuit prendra une entrée `in` sur 16 bits, une entrée `k` sur 3 bits, et produira une sortie `out` sur 16 bits : `out` sera le résultat de `k` rotations à droite sur `in`. Vous utiliserez `cumdecod`, ainsi que 7 fois `rotdsel`.

6. En plus des opérations habituelles (`ADD`, `AND`, `NOT`) on veut que l'ALU puisse effectuer une rotation de k bits à droite de `in1` (k est fourni soit par les 3 bits de poids faibles `in2`, soit par ceux de `cste`). Complétez le circuit `alu` de façon à ce qu'il produise `out` comme indiqué ci-dessous.

	`UseCste`	
e2	0	1
00	`out` ← `ROTD` `in1`, `in2`	`out` ← `ROTD` `in1`, `cste`
01	`out` ← `ADD` `in1`, `in2`	`out` ← `ADD` `in1`, `cste`
10	`out` ← `NOT` `in1`	`out` ← `NOT` `in1`
11	`out` ← `AND` `in1`, `in2`	`out` ← `AND` `in1`, `cste`