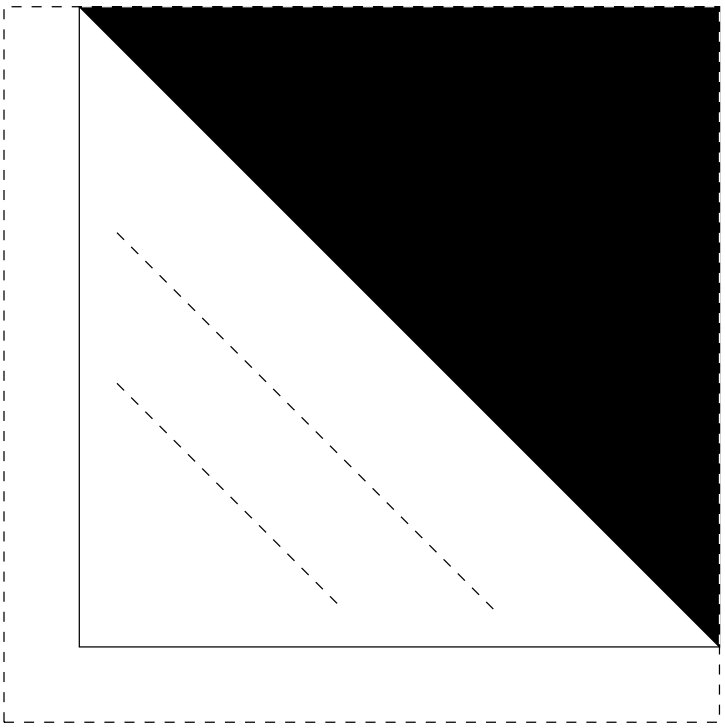
CC1 - LIF6 Printemps 2014
Durée prévue : 1h30

Aucun document autorisé. Calculatrices, téléphones et ordinateurs portables interdits.

N'oubliez pas de noter votre nom, votre prénom et votre numéro d'étudiant sous le cache rabattable.

Justifiez brièvement et précisément vos réponses, tout en respectant les emplacements prévus.

Le barème est donné à titre indicatif, et pourra être modifié.



Ne rien écrire dans ce cadre.
Partie 1 :
Partie 2 :
Partie 3 :
Partie 4 :

1 Calculs avec les nombres flottants binaires (4,5 pts)

Dans cet exercice, on note $x_1 = (3,14)_{10}$, $x_2 = (1.0111011)_2$ et $x_3 = (1.0101110)_2$. Détaillez soigneusement vos calculs.

On travaille sur une machine où les flottants binaires sont codés sur 12 bits. On considère le code :

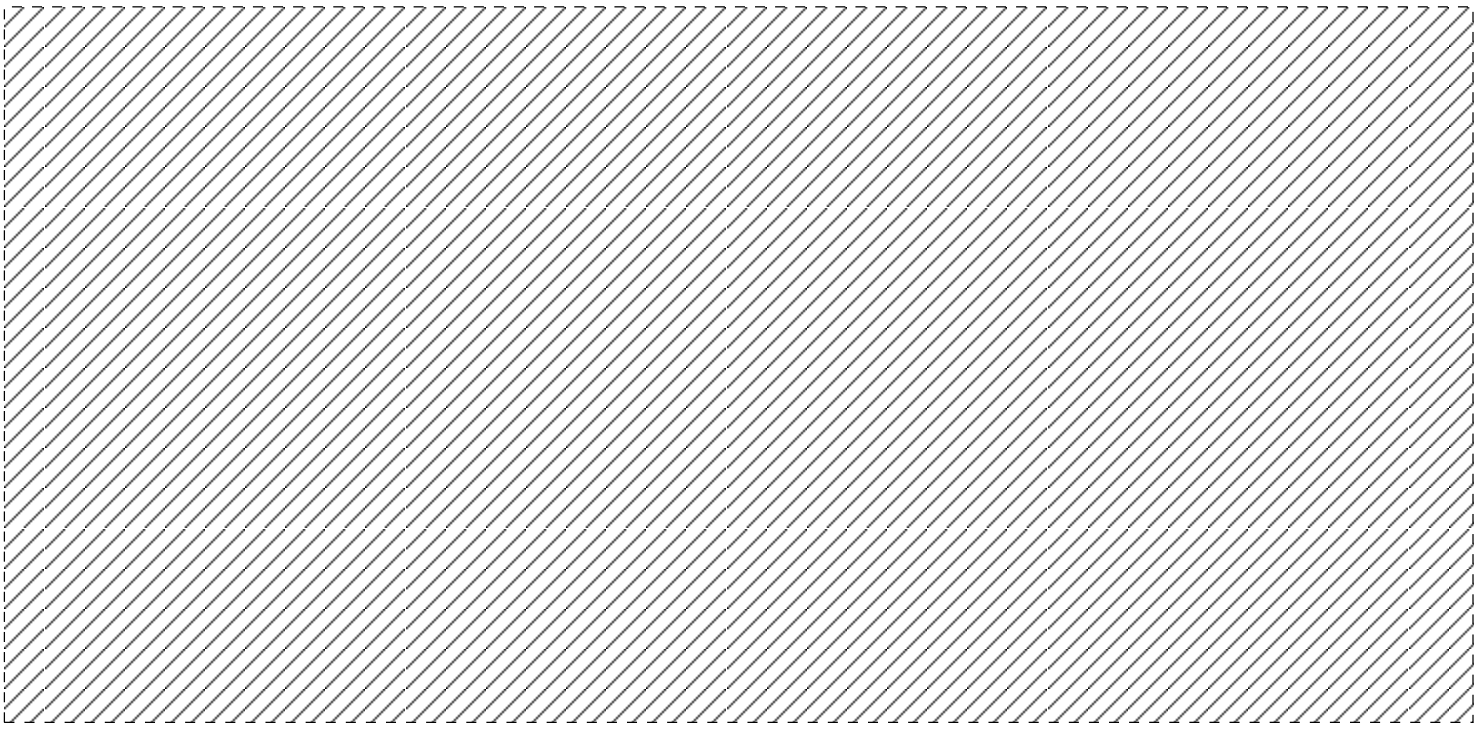
$c =$

11	10	9	8	7	6	5	4	3	2	1	0
s	b				m						

Le nombre représenté est $f(c) = (-1)^s \times (1,m)_2 \times 2^e$, avec $e = (b)_2 - 2^3$ si $1 \leq (b)_2 \leq 14$, (on ne se préoccupe pas du codage des sous-normaux ni des valeurs exeptionnelles ici).

1. (1 point) Convertissez exactement le nombre x_1 en notation positionnelle à virgule en binaire.

2. (1.5 points) Comment peut-on représenter x_1 dans le format flottant binaire décrit ci-dessus ? Vous effectuerez un arrondi au plus proche, et exprimerez votre résultat sous la forme d'un code binaire, puis en *hexadécimal*.



3. (0.5 points) Posez la soustraction $x_2 - x_3$. On note x_4 le résultat de l'opération.

4. (1.5 points) Donnez la représentation de x_4 sous la forme dans le format flottant binaire décrit ci-dessus. Vous effectuerez un arrondi au plus proche, et exprimerez votre résultat sous la forme d'un code binaire, puis en *octal*.

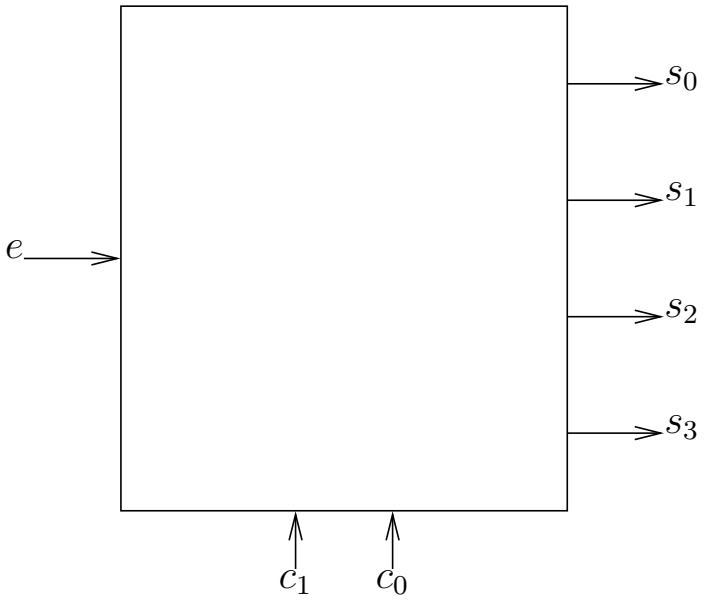
2 Multiplexeurs et démultiplexeurs (4,5 pts)

Pour un certain entier $n > 1$, la définition d'un multiplexeur 2^n vers 1 à été vue en cours. On donne celle d'un démultiplexeur 1 vers 2^n : il s'agit d'un circuit combinatoire qui présente n entrées de commande $c_{n-1}, \ldots, c_0$, une entrée e à « démultiplexer », et de 2^n sorties, telles que $s_{(c_{n-1}\ldots c_0)_2} = e$, et $s_i = 0$ si $i \neq (c_{n-1} \ldots c_0)_2$.

1. (1 point) Complétez la table suivante dans le but de réaliser un démultiplexeur 1 vers 4.

c_1	c_0	s_0	s_1	s_2	s_3
0	0				
0	1				
1	0				
1	1				

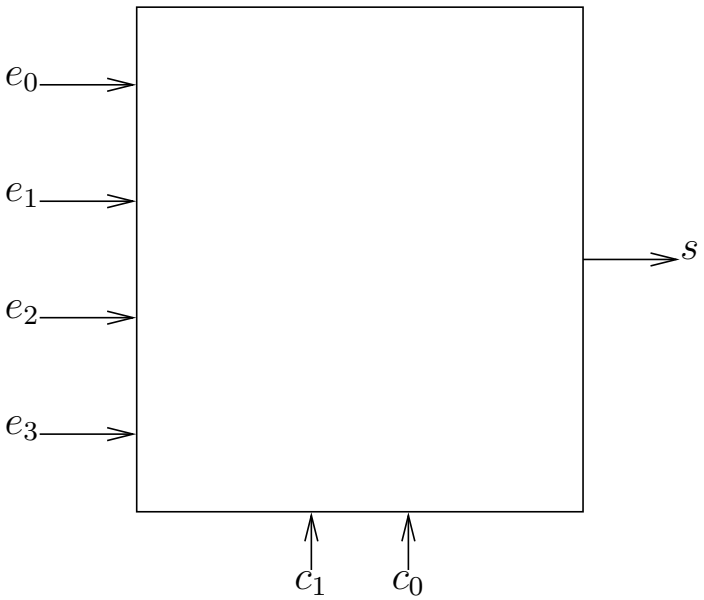
2. (1 point) Donnez un circuit logique combinatoire pour un démultiplexeur 1 vers 4, en utilisant 4 portes AND à 3 entrées, et des inverseurs. Veillez à la lisibilité et à la propreté de votre schéma.



3. (0.5 points) Complétez la table suivante dans le but de réaliser un multiplexeur 4 vers 1 dont les entrées à multiplexer sont $e_0, \ldots, e_3$, la commande est donnée par $(c_1c_0)_2$, et la sortie est s, telles que $s = e_{(c_1c_0)_2}$.

c_1	c_0	e
0	0	
0	1	
1	0	
1	1	

4. (2 points) Donnez un circuit logique combinatoire pour un multiplexeur 4 vers 1, en utilisant 3 multiplexeurs 2 vers 1, et en respectant la définition donnée dans la question précédente.



3 Mémoire cache (5,5 pts)

Un ordinateur dispose d'une mémoire centrale de 64 kio : les adresses sont codées sur 16 bits, et la taille de chaque case mémoire est de 1 o. Entre le processeur et la mémoire centrale est placée une petite mémoire cache directe, composée de 8 entrées pouvant chacune contenir 32 o. On note ℓ_m la ligne de cache d'indice m, et e_n l'entrée de la mémoire cache d'indice n.

On considère le programme ci-dessous, dans lequel la taille d'un `int` est de 32 bits. Les variables dont les déclarations sont contiguës dans le programme le sont également en mémoire. On suppose pour simplifier que la matrice `A` est stockée à partir de l'adresse 0, et que la variable `i` est maintenue dans un registre.

```
#define N 8
int A[N][N], B[N][N], i;
for(i=0; i<N; i++)
  for(j=0; j<N; j++) A[i][j] = B[j][i];
```

1. (0.5 points) Combien d'entiers de type `int` sont contenus dans une ligne de cache ? Justifiez.

   .....

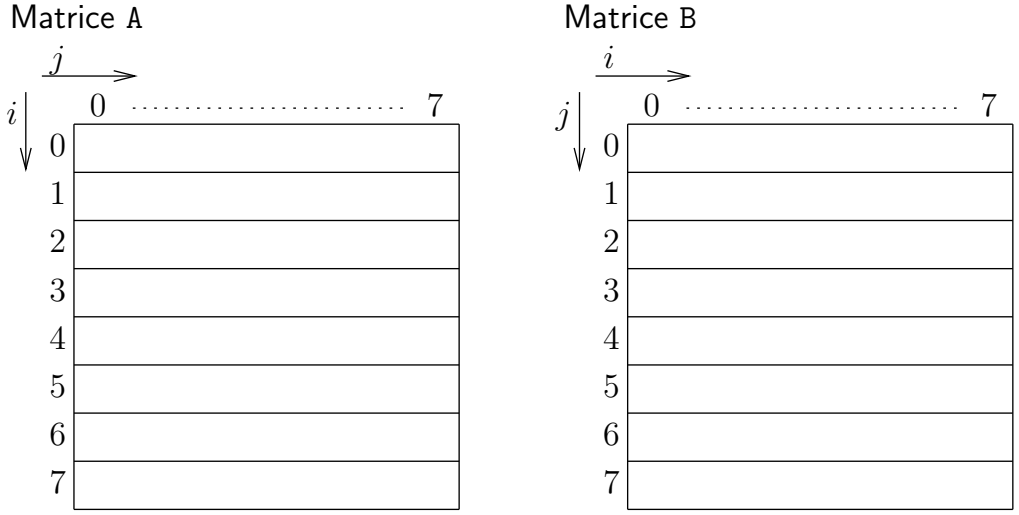
2. (1 point) Exprimez, en fonction de m, l'indice n de l'entrée e_n dans laquelle doit être rangée ℓ_m. Justifiez.

   .....

   .....

   .....

3. (1 point) Complétez le schéma ci-dessous, en indiquant pour chaque partie des matrices `A` et `B`, la ligne à laquelle elle appartient, et l'entrée du cache dans laquelle elle devra être rangée.



4. (1 point) On fixe `i` $= 0$: complétez le tableau ci-dessous, en indiquant la ligne contenue dans chaque entrée du cache à la fin de chaque itération de la boucle « `for(j=0; j<N; j++) A[i][j] = B[j][i];` » (utilisez des pointillés). Indiquez le nombre de défauts de cache qui ont eu lieu au cours de chaque itération.

$j \rightarrow$	0	1	2	3	4	5	6	7
e_0								
e_1								
e_2								
e_3								
e_4								
e_5								
e_6								
e_7								
défauts								

5. (1 point) Même question que la précédente pour `i` $= 1$.

$j \rightarrow$	0	1	2	3	4	5	6	7
e_0								
e_1								
e_2								
e_3								
e_4								
e_5								
e_6								
e_7								
défauts								

6. (1 point) Pour l'exécution du programme complet, quel est le nombre total de défauts de cache ? Justifiez.

.....

.....

.....

.....

.....

.....

.....

4 Instructions de branchement et boucles (5,5 pts)

Un processeur 8 bits est doté d'un espace mémoire de 64 Kio, et chaque case de la mémoire contient 1 octet. Le processeur dispose de 4 registres de travail, indicés de 0 à 4 et notés `R0`...`R3`, ainsi que d'un *program status register* (le PSR), qui comporte un drapeau nommé `z` : ce drapeau est à 1 si la dernière valeur écrite dans l'un des registres était nulle, 0 sinon. Les instructions sont codées sur 1 ou 2 octets, comme indiqué dans le tableau suivant.

		premier octet								second octet
assembleur	action	7	6	5	4	3	2	1	0	7...0
`SET RD,imm8`	`RD <- imm8`	0	0	0	RD		0	0	0	imm8
`ADD RD,RS`	`RD <- RD + RS`	0	0	1	RD		RS		0	—
`SUB RD,RS`	`RD <- RD - RS`	0	1	0	RD		RS		0	—
`ST RS,adr`	`Mémoire[adr] <- RS`	1	0	0	SR		0	0	0	adr
`LD RD,adr`	`RD <- Mémoire[adr]`	1	0	1	RD		0	0	0	adr
`BR dec5`	`PC <- adins + 1 + dec5`	1	1	0	dec5					—
`BRZ dec5`	Si `z` = 1 alors `PC <- adins + 1 + dec5`	1	1	1	dec5					—

Dans ce tableau :
- `imm8` est un entier immédiat codé en complément à 2 sur 8 bits ;
- `RD` désigne un registre de destination et `RS` un registre source (`R0`...`R3`) ;
- `PC` désigne le compteur de programme (adresse de la prochaine instruction) ;
- `adr` et `adins` désignent des adresses en mémoire, codées sur 1 octet ;
- `Mémoire[adr]` désigne la case mémoire d'adresse `adr`.

Les instructions de branchement `BR` et `BRZ` permettent de faire des sauts dans le programme, en modifiant la valeur de `PC`. On note `adins` l'adresse de l'instruction de branchement dans le programme, et `dec5` est un entier (décalage) codé en complément à 2 sur 5 bits par rapport à cette adresse :
- Dans le cas de `BR dec5`, lorsque l'instruction est exécutée, alors l'instruction qui sera exécutée ensuite est celle à l'adresse `adins + 1 + dec5` (branchement inconditionnel).
- Dans le cas de `BRZ dec5` : si `z` = 1 lorsque l'instruction est executée, alors la prochaine instruction exécutée est celle à l'adresse `adins + 1 + dec5` (branchement pris), sinon la prochaine instruction sera celle à l'adresse `adins + 1` (branchement non-pris).

```
n:       76            // case mémoire contenant l'entier 76
r:       0             // case mémoire ou sera stocké le résultat
begin:   SET R0, 0     // ........................................
         SET R1, 1     // ........................................
         LD R2, n      // ........................................
loop:    BRZ endloop   // ........................................
         ADD R0, R2    // ........................................
         SUB R2, R1    // ........................................
         BR loop       // ........................................
endloop: ST R0, r      // ........................................
```

1. (1 point) Expliquez, en justifiant, ce que calcule ce programme.

.....

.....

.....

.....

.....

.....

.....

.....

.....

2. (1.5 points) On suppose que la première ligne du programme sera rangée en mémoire à l'adresse $(002B)_H$. Complétez le tableau ci-dessous, en indiquant pour chaque donnée son adresse, et pour chaque instruction l'adresse de son premier octet.

adresse	instruction	
$(002B)_H$	`n:`	`76`
	`r:`	`0`
	`begin:`	`SET R0,0`
		`SET R1,1`
		`LD R2,n`
	`loop:`	`BRZ endloop`
		`ADD R0,R2`
		`SUB R2,R1`
		`BR loop`
	`endloop:`	`ST R0,r`

3. (1.5 points) Calculez la valeur de décalage `dec5` pour l'instruction `BRZ endloop`, puis pour l'instruction `BR loop` : détaillez bien vos calculs. Donnez votre résultat en décimal puis en binaire.

.....

.....

.....

.....

.....

.....

4. (1.5 points) Donnez le codage en binaire du programme (rayez les octets inutiles) :

instruction		premier octet								second octet							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
`n:`	`76`																
`r:`	`0`																
`begin:`	`SET R0,0`																
	`SET R1,1`																
	`LD R2,n`																
`loop:`	`BRZ endloop`																
	`ADD R0,R2`																
	`SUB R2,R1`																
	`BR loop`																
`endloop:`	`ST R0,r`																