CC2 - LIF6 Printemps 2015
Durée prévue : 1h15

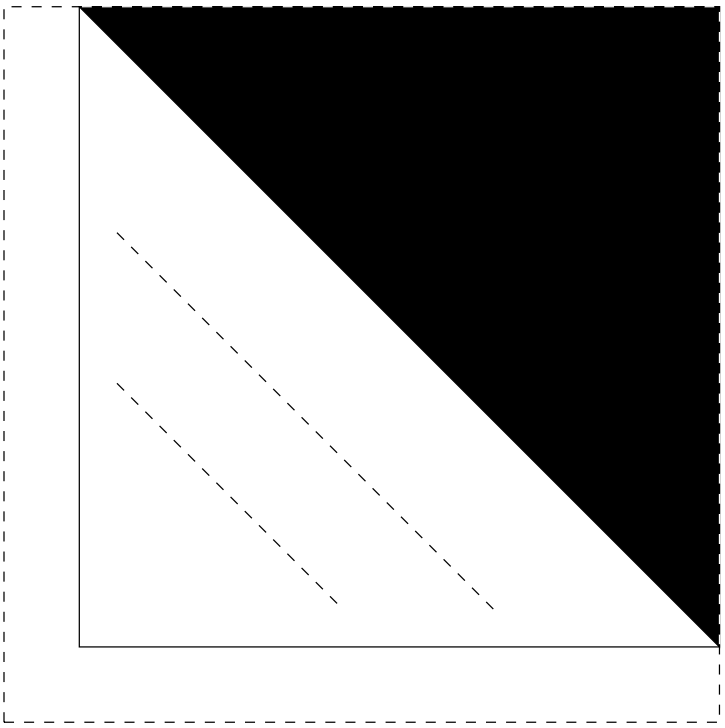
Aucun document autorisé. Calculatrices, téléphones et ordinateurs portables interdits.

N'oubliez pas de noter votre nom, votre prénom et votre numéro d'étudiant sous le cache rabattable. Débrouillez vous pour que le cache soit rabattu et collé !

Ne soyez ni trop prolixe ni trop concis : justifiez précisément vos réponses dans les emplacements prévus.

Le barème est donné à titre indicatif.

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



1 Mémoire cache et transposition d'une matrice (7 points)

On considère la double boucle du programme C ci-dessous dans laquelle la matrice `M` est transposée. L'ordinateur sur lequel on exécute ce programme n'utilise qu'une petite mémoire cache de 4 entrées de 8 octets chacune. On note $\ell_0, \ell_1, \ldots$ les lignes de cache, $e_0, e_1, \ldots$ les entrées du cache, et pour $0 \leq \texttt{i} \leq 7$, on note `M[i][:]` la ligne i de `M`. La matrice `M` est stockée par ligne, et on suppose pour simplifier que son premier élément a l'adresse 0 en mémoire. Les variables `t`, `i`, `j` sont stockées dans des registres, et n'interfèrent donc pas avec le cache.

```
int main(void) {
  unsigned char M[8][8], t;
  int i, j;

  /* Il se passe des choses ici... */
  for(i=0; i<8; i++) {
    for(j=i+1; j<8; j++) {
      t = M[i][j];          /* lecture 1 */
      M[i][j] = M[j][i];    /* lecture 2, puis écriture 1 */
      M[j][i] = t;          /* écriture 2 */
    }
  }
  /* Il se passe d'autres choses là... */

  return(0);
}
```

1. (1 point) Pour $0 \leq \texttt{i} \leq 8$, la ligne `M[i][:]` occupe précisément ℓ_i : pourquoi ?

   .....

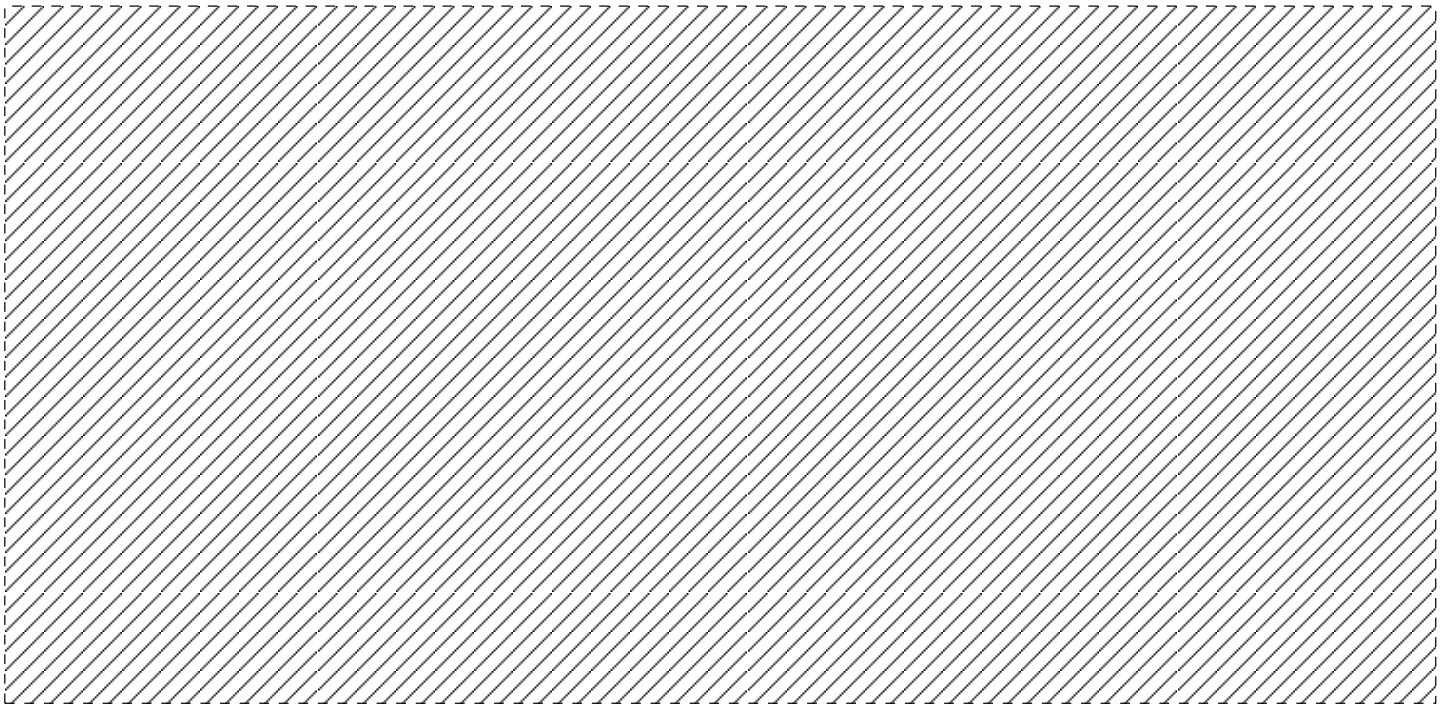
   .....

   .....

2. (1 point) Complétez le tableau suivant en indiquant à quelle ligne de cache appartient chaque ligne de la matrice, et dans quelle entrée elle doit être rangée.

ligne de la `M`	`M[0][:]`	`M[1][:]`	`M[2][:]`	`M[3][:]`	`M[4][:]`	`M[5][:]`	`M[6][:]`	`M[7][:]`
ligne de cache								
entrée du cache								

3. (2 points) On fixe dans cette question `i` $= 0$, et on s'intéresse à l'exécution de la boucle interne sur `j`. Pour chaque itération de la boucle portant sur j, indiquez dans le tableau ci-dessous :
   — quelles sont les lignes de cache accédées pour chaque opération de lecture ou d'écriture du corps de boucle ;
   — quelle ligne est présente dans chaque ligne de cache à la fin de l'exécution du corps de boucle ;



— le nombre de défauts de cache qui se sont produits à la fin de l'exécution du corps de boucle.
La numérotation des opérations de lecture et d'écriture est précisée en commentaire dans le programme.

j	lecture 1	lecture 2	écriture 1	écriture 2	e_0	e_1	e_2	e_3	défauts
1									
2									
3									
4									
5									
6									
7									

4. (2 points) Même chose qu'à la question précédente, mais cette fois-ci pour `i` = 1.

j	lecture 1	lecture 2	écriture 1	écriture 2	e_0	e_1	e_2	e_3	défauts
2									
3									
4									
5									
6									
7									

5. (1 point) Pour les deux premières itérations de la boucle sur `i`, combien d'accès à la mémoire sont effectués ? Quel est le nombre total de défauts de cache ? Le taux de défaut de cache est-il plus grand que 50% ?

.....

.....

2 Pipeline (5 points)

On considère dans cette partie un pipeline à 4 étages, comme celui du cours. Les étages du pipeline sont FE (*fetch*, chargement), DE (*decode*, décodage), EX (*execute*, exécution) et SR (*store register*, stockage du résultat en registre). On rappelle que la phase EX d'une instruction ne peut être effectuée avant que ses opérandes n'aient été effectivement calculées et stockées (phase SR terminée) ; si ce n'est pas le cas on parlera ici de *défaut de pipeline*.

1. (0.5 points) En l'absence de défaut, si une instruction entre dans le pipeline au cycle i, à quel cycle se finit son exécution ?

.....

2. (1.5 points) En ignorant les défauts de pipeline, représentez le traitement des instructions dans le pipeline en remplissant avec FE, DE, EX ou SR dans les cases du tableau suivant. Vous ajouterez entre parenthèses les registres lus dans la phase EX et ceux écrits dans la phase SR :

Instruction	Cycle 1	2	3	4	5	6	7	8	9	10	11
`ADD R4,R1,R2`											
`NOT R4,R4`											
`ADD R3,R3,R4`											
`ADD R4,R5,R1`											
`ADD R5,R5,R2`											

3. (1 point) Entourez dans le tableau précédent les défauts de pipeline et expliquez-les rapidement.

.....

.....

4. (2 points) Résoudre les défauts de cache en rajoutant des attentes de pipeline (un *stall*, que vous noterez S). On considère qu'un stall bloque une instruction à un étage précis du pipeline, ainsi que toutes les instructions qui la suivent dans le pipeline.

Instruction	Cycle 1	2	3	4	5	6	7	8	9	10	11
`ADD R4,R1,R2`											
`NOT R4,R4`											
`ADD R3,R3,R4`											
`ADD R4,R5,R1`											
`ADD R5,R5,R2`											

5. (1 point) Proposez (en expliquant) un ré-ordonnancement des instructions minimisant le temps total d'exécution. Combien de cycle(s) avez-vous gagné ?

.....

.....

3 Renversement d'une chaîne de caractères (8 points)

Le but est d'écrire dans le langage d'assemblage du LC3 une routine `revit` permettant le renversement d'une chaîne de caractères se terminant par un `'\0'` (dont le code ASCII est l'entier 0). Cette routine utlisera `R0` comme paramètre d'entrée : avant l'appel, ce registre devra contenir l'adresse du premier caractère de la chaîne. Après, l'appel, la chaîne aura été transformée en son renversé. Rappelons que le renversé de la chaîne `fins` est `snif`.

1. (1 point) Commencez par compléter le programme ci-dessous de façon à ce qu'il affiche successivement `fins`, puis `snif`. La macro `PUTS` affichage la chaîne dont l'adresse du premier caractère se trouve dans `R0`.

```
; Programme principal
          LEA R6,spinit  ; On initialise le pointeur de pile
          ..............  ; Chargement dans R0 de l'adresse de la chaîne
          PUTS           ; Premier affichage de la chaîne
          ..............  ; Appel à la routine revit pour renverser la chaîne
          ..............  ; Chargement dans R0 de l'adresse de la chaîne
          PUTS           ; Deuxième affichage de la chaîne
          HALT
; Données
chaine:   .STRINGZ "fins"
          .BLKW #8
spinit:   .BLKW #1       ; adresse du fond de la pile
```

2. (3 points) On veut écrire une routine auxiliaire `last` pour obtenir l'adresse du dernier caractère d'une chaîne (le caractère précédent le `'\0'` final). Cette routine prend en entrée l'adresse du premier caractère de la chaîne dans `R0`, et calcule l'adresse du dernier caractère dans `R1`. Le contenu de `R0` doit être préservé par l'appel. Donnez un code pour la routine `last` en langage d'assemblage du LC3.

3. (3 points) Traduisez le pseudo-code suivant pour la routine `revit` en langage d'assemblage du LC3. Veillez à sauvegarder l'adresse de retour sur la pile à l'entrée dans la routine, et à restaurer cette valeur avant la sortie. La ligne `Mem[R0] <-> Mem[R1]` indique l'échange du contenu des cases mémoires pointées par `R0` et `R1`.

```
routine revit(R0) {
  // R0 : entrée, adresse du début de la chaîne à renverser.
  // Attention, R0, R1, R2 et R3 sont modifiés par l'appel.
  push(R7);
  call last(R1, R0);
  while(R0 < R1) {
    Mem[R0] <-> Mem[R1];
    R0++;
    R1--;
  }
  pop(R7);
  return;
}
```

4. (1 point) Pourquoi est-il important de sauvegarder l'adresse de retour au début de la routine `revit`, puis de restaurer cette adresse de retour à la fin de la routine ?