NOM, Prénom, numéro d'étudiant, groupe : VITRIOL, Bleu, 01, V

## CC-TD-3A - LIF6 - vendredi 13/5/16 - noté sur 10 - durée : 30 minutes

Aucun document autorisé; calculatrices, téléphones et ordinateurs interdits. Le barème pourra être modifié.

**Exercice 1.** Vous devez écrire une routine nzb dans le langage d'assemblage du LC3 pour compter le nombre de bits non-nuls dans un entier naturel (par exemple, le nombre de bits non-nuls dans $(211)_{10} = (11010011)_2$ est 5). Vous traduirez pour cela le pseudo-code suivant :

```
routine nzb(R0, R1) {
  // Entrée : R0 (entier naturel, sera modifié dans la routine)
  // Sortie : R1 (nombre de bits non-nuls dans R0)
  // Le registre R2 est écrasé par l'appel.
  R1 <- 0
  while(R0 != 0) {
    R2 <- R0 - 1;
    R0 <- R0 & R2; // & désigne le AND du LC3
    R1 <- R1 + 1;
  }
  return;
}
```

**Question 1** (1 pt). Complétez le tableau ci-dessous, en indiquant les valeurs de R2 et R0 **à la fin de chaque itération** de la boucle de la routine nzb. Expliquez rapidement le principe de l'algorithme mis en œuvre :

| itération | R2 | R0 |
|---|---|---|
| initialement | — | 1101 0011 |
| 1 | 1101 0010 | 1101 0010 |
| 2 | 1101 0001 | 1101 0000 |
| 3 | 1100 1111 | 1100 0000 |
| 4 | 1011 1111 | 1000 0000 |
| 5 | 0111 1111 | 0000 0000 |

L'algo supprime le bit non-nul de poids le plus faible dans R0 à chaque itération

**Question 2** (4 pts). Complétez le programme principal de façon à charger le contenu de la case mémoire de label n dans R0, appeler la routine nzb, puis ranger le contenu de R1 dans la case mémoire de label r. Complétez aussi le code de la routine nzb **conformément au pseudo-code** donné ci-dessus.

```
        .ORIG x3000
; Programme principal
        LD R0,n     ; R0 ← mem[n]
        JSR nzb     ; appel à la routine
        ST R1,r     ; mem[r] ← R1
        HALT        ; rend la main à l'OS
; Données
n:      .FILL 211
r:      .BLKW 1

; Routine nzb
; Entrée : R0 (entier naturel, sera modifié dans la routine)
; Sortie : R1 (nombre de bits non-nuls dans R0)
nzb:    AND R1,R1,0 ; R1 ← 0
loop:   ADD R0,R0,0 ; mise à jour de NZP d'après R0
        BRz endl    ; condition de sortie de boucle : R0 = 0
        ADD R2,R0,-1; R2 ← R0 - 1
        AND R0,R0,R2; R0 ← R0 & R2
        ADD R1,R1,1 ; R1 ← R1 + 1
        BR loop     ; saut à loop
endl:   RET         ; .....
        .....       ; .....
        .....       ; .....
        .END
```

attention, il faut que les drapeaux NZP soient mis à jour avant le test de sortie de boucle! C'est pas bien! — 6 !

Voici le tableau des instructions du LC3 que vous pouvez utiliser :

| 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.** 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 16 entrées pouvant chacune contenir 32 o. On décompose les adresses en trois champs :

| INDICATEUR | | | | | | | ENTREE | | | | OCTET | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

On rappelle que : la concaténation des champs INDICATEUR et ENTREE donne en binaire l'indice d'une ligne de cache ; le champ ENTREE donne en binaire l'indice d'une entrée dans la mémoire cache ; le champ OCTET donne l'indice d'un octet dans une ligne de cache (ou une entrée dans la mémoire cache).

**Question 3** (1 pt). À chaque fois, donnez votre calcul :

1. Dans quelle entrée est stockée la ligne de cache $\ell_{17}$ ? $17 = 1 \times 16 + 1 \Rightarrow e_1$
   Ou encore : $(17)_{10} = (0\cdots01\,0001)_2 \Rightarrow e_1$
   ENTREE = 1

2. Dans quelle entrée est stockée la ligne de cache $\ell_{56}$ ? $56 = 3 \times 16 + 8 \Rightarrow e_8$
   Ou encore : $(56)_{10} = 32 + 16 + 8 = (0\cdots0\,111\,000)_2$
   $= 7 \times 8$ ENTREE = 8

**Question 4** (2 pts). Complétez le tableau suivant, en écrivant les adresses en binaire, puis en donnant leur indice de ligne et d'entrée dans le cache.

| adresse (hexa.) | adresse (binaire) | ligne (décimal) | entrée (décimal) | |
|---|---|---|---|---|
| $0B47_H$ | 0000 1011 0100 0111 | 90 | 10 | $5 \times 16 + 10$ |
| $0D58_H$ | 0000 1101 0101 1000 | 106 | 10 | $6 \times 16 + 10$ |
| $0B52_H$ | 0000 1011 0101 0010 | 90 | 10 | |
| $0AB3_H$ | 0000 1010 1011 0011 | 85 | 5 | $5 \times 16 + 5$ |
| $0AA7_H$ | 0000 1010 1010 0111 | 85 | 5 | |

**Question 5** (2 pts). Le processeur accède successivement aux adresses $0B47_H$, $0D58_H$, $0B52_H$, $0AB3_H$, $0AA7_H$ (à celles-là seulement). Le cache est initialement vide : indiquez quelle ligne est contenue dans chaque entrée après chaque accès (laissez vides les cases où vous ne pouvez pas savoir), et signalez chaque défaut de cache.

| adresse | entrées | | | | | | | | | | | | | | | | défaut |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $e_0$ | $e_1$ | $e_2$ | $e_3$ | $e_4$ | $e_5$ | $e_6$ | $e_7$ | $e_8$ | $e_9$ | $e_{10}$ | $e_{11}$ | $e_{12}$ | $e_{13}$ | $e_{14}$ | $e_{15}$ | |
| $0B47_H$ | | | | | | | | | | | 90 | | | | | | X |
| $0D58_H$ | | | | | | | | | | | 106 | | | | | | X |
| $0B52_H$ | | | | | | | | | | | 90 | | | | | | X |
| $0AB3_H$ | | | | | | 85 | | | | | 90 | | | | | | X |
| $0AA7_H$ | | | | | | 85 | | | | | 90 | | | | | | |

NOM, Prénom, numéro d'étudiant, groupe : D'ŒUF, John, 02, U

**CC-TD-3B - LIF6 - vendredi 13/5/16 - noté sur 10 - durée : 30 minutes**

Aucun document autorisé; calculatrices, téléphones et ordinateurs interdits. Le barème pourra être modifié.

**Exercice 1.** Vous devez écrire une routine nzb dans le langage d'assemblage du LC3 pour compter le nombre de bits non-nuls dans un entier naturel (par exemple, le nombre de bits non-nuls dans $(227)_{10} = (11100011)_2$ est 5). Vous traduirez pour cela le pseudo-code suivant :

```
routine nzb(R0, R1) {
  // Entrée : R1 (entier naturel, sera modifié dans la routine)
  // Sortie : R0 (nombre de bits non-nuls dans R1)
  // Le registre R2 est écrasé par l'appel.
  R0 <- 0
  while(R1 != 0) {
    R0 <- R0 + 1;
    R2 <- R1 - 1;
    R1 <- R1 & R2; // & désigne le AND du LC3
  }
  return;
}
```

**Question 1** (1 pt). Complétez le tableau ci-dessous, en indiquant les valeurs de R2 et R1 **à la fin de chaque itération** de la boucle de la routine nzb. Expliquez rapidement le principe de l'algorithme mis en œuvre :

| itération | R2 | R1 |
|---|---|---|
| initialement | — | 1110 0011 |
| 1 | 1110 0010 | 1110 0010 |
| 2 | 1110 0001 | 1110 0000 |
| 3 | 1101 1111 | 1100 0000 |
| 4 | 1011 1111 | 1000 0000 |
| 5 | 0111 1111 | 0000 0000 |

L'algo supprime le bit non-nul de poids le plus faible dans R1 à chaque itération.

**Question 2** (4 pts). Complétez le programme principal de façon à charger le contenu de la case mémoire de label n dans R1, appeler la routine nzb, puis ranger le contenu de R0 dans la case mémoire de label r. Complétez aussi le code de la routine nzb **conformément au pseudo-code** donné ci-dessus.

```
        .ORIG x3000
; Programme principal
        LD R1,n     ; R1 ← mem[n]
        JSR nzb     ; appel à la routine
        ST R0,r     ; mem[r] ← R0
        HALT        ; rend la main à l'OS
; Données
n:      .FILL 227
r:      .BLKW 1

; Routine nzb
; Entrée : R1 (entier naturel, sera modifié dans la routine)
; Sortie : R0 (nombre de bits non-nuls dans R1)
nzb:    AND R0,R0,0 ; R0 ← 0
        ADD R1,R1,0 ; mise à jour de NZP d'après R1
loop:   BRz endl    ; condition de sortie de boucle : R1 = 0
        ADD R0,R0,1 ; R0 ← R0+1
        ADD R2,R1,-1; R2 ← R1 - 1
        AND R1,R1,R2; R1 ← R1 & R2
        BR loop     ; saut à loop
endl:   RET         ; return
        ...         ; ...
        ...         ; ...
        .END
```

NB : les drapeaux NZP sont bien à jour (d'après R1) lors du test de sortie de boucle... C'est bien !

Voici le tableau des instructions du LC3 que vous pouvez utiliser :

| 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.** 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 mémoire cache directe, composée de 16 entrées pouvant chacune contenir 64 o. On décompose les adresses en trois champs :

| INDICATEUR | | | | | | ENTREE | | | | OCTET | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

On rappelle que : la concaténation des champs INDICATEUR et ENTREE donne en binaire l'indice d'une ligne de cache ; le champ ENTREE donne en binaire l'indice d'une entrée dans la mémoire cache ; le champ OCTET donne l'indice d'un octet dans une ligne de cache (ou une entrée dans la mémoire cache).

**Question 3** (1 pt). À chaque fois, donnez votre calcul :

1. Dans quelle entrée est stockée la ligne de cache $\ell_{19}$ ? $19 = 1 \times 16 + 3 \Rightarrow e_3$
   Ou encore : $(19)_{10} = (0\ldots0\,1\,0\,0\,1\,1)_2 \Rightarrow e_3$
   ENTREE = 3

2. Dans quelle entrée est stockée la ligne de cache $\ell_{52}$ ? $52 = 3 \times 16 + 4 \Rightarrow e_4$
   Ou bien : $(52)_{10} = 3 \times 16 + 4 = (0\ldots0\,1\,1\,0\,1\,0\,0)_2 \Rightarrow e_4$
   ENTREE = 4

**Question 4** (2 pts). Complétez le tableau suivant, en écrivant les adresses en binaire, puis en donnant leur indice de ligne et d'entrée dans le cache.

| adresse (hexa.) | adresse (binaire) | ligne (décimal) | entrée (décimal) | |
|---|---|---|---|---|
| $0EB3_H$ | 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 1 | 58 | 10 | $3 \times 16 + 10$ |
| $0EA7_H$ | 0 0 0 0 1 1 1 0 1 0 1 0 0 1 1 1 | 58 | 10 | |
| $1A47_H$ | 0 0 0 1 1 0 1 0 0 1 0 0 0 1 1 1 | 105 | 9 | $3 \times 32 + 9$ |
| $1658_H$ | 0 0 0 1 0 1 1 0 0 1 0 1 1 0 0 0 | 89 | 9 | $5 \times 16 + 9$ |
| $1A52_H$ | 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 | 105 | 9 | |

**Question 5** (2 pts). Le processeur accède successivement aux adresses $0EB3_H$, $0EA7_H$, $1A47_H$,$1658_H$, $1A52_H$ (à celles-là seulement). Le cache est initialement vide : indiquez quelle ligne est contenue dans chaque entrée après chaque accès (laissez vides les cases où vous ne pouvez pas savoir), et signalez chaque défaut de cache.

| adresse | entrées | | | | | | | | | | | | | | | | défaut |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $e_0$ | $e_1$ | $e_2$ | $e_3$ | $e_4$ | $e_5$ | $e_6$ | $e_7$ | $e_8$ | $e_9$ | $e_{10}$ | $e_{11}$ | $e_{12}$ | $e_{13}$ | $e_{14}$ | $e_{15}$ | |
| $0EB3_H$ | | | | | | | | | | | 58 | | | | | | X |
| $0EA7_H$ | | | | | | | | | | | 58 | | | | | | |
| $1A47_H$ | | | | | | | | | | 105 | 58 | | | | | | X |
| $1658_H$ | | | | | | | | | | 89 | 58 | | | | | | X |
| $1A52_H$ | | | | | | | | | | 105 | 58 | | | | | | X |