Architecture des ordinateurs
L. Gonnord & N. Louvet
http://laure.gonnord.org/pro/teaching/archiL2.html

Contrôle continu Final - LIF6 Archi - 2h – Éléments de correction

Tous documents papier autorisés. Ce contrôle est à rédiger sur une copie anonyme. Merci de bien numéroter les copies, de préciser le numéro d'anonymat sur chacune d'elles, et de numéroter vos réponses (2.1, 2.2, ...). Le sujet est long, le barême en tiendra compte. Les questions avec $\star$ rapporteront le plus de points. La feuille réponse sera jointe à votre copie.

Version 1 des éléments de correction, 1er février 2015. Sans garantie.

1 Représentation des nombres

Question 1:
Donner la représentation en complément à deux sur 7 bits de $(15)_{10}$ et $(-31)_{10}$. Effectuer l'opération en complément à deux correspondant à $(-31)_{10} + (15)_{10}$. Vérifier.

Solution: Sans difficulté, on obtient $(0001111)_{\bar{2}}$ et $(1100001)_{\bar{2}}$, ce qui en additionnant donne (1110000), ie -16.

Question 2:
Quel est le nombre réel dont la représentation en binaire est donnée par $(0, 111)_2$? L'écrire, en base 10, en fraction irréductible ainsi qu'en nombre à virgule.

Solution: Il s'agit de $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} = \frac{7}{8} = (0.875)_{10}$.

2 Mémoire cache

On considère, dans cet exercice [1], que les mots mémoire sont de taille un octet. Soit un cache direct de 4ko (= 2^{12} octets) dont les lignes font 128 (= 2^7) octets. Soit l'adresse hexadécimale $a = (A23847EF)_{16}$ d'une donnée.

Question 1:
Sur combien de bits les adresses sont-elles codées ? Donnez le nombre de bits codant les champs INDICATEUR, ENTREE, et OFFSET (selon la terminologie du cours sur les caches directs).

Solution: (Le correcteur a la flemme de faire un dessin, mais c'est ce qu'il faudrait faire) Comme les lignes font 2^7 octets, l'indice d'une donnée dans une ligne (OFFSET) est donné sur 7 bits (les bits de poids faibles). Le reste des 32 bits (8 chiffres hexa) des adresses fournit INDICATEUR+ENTREE sur 25 bits, donc. Le cache possède $2^{12}/2^7 = 2^5$ entrées, donc une entrée est codée sur 5 bits. On en déduit que INDICATEUR est sur 20 bits.

Question 2:
Pour chacune de ces adresses, donnez (en hexadécimal) le numéro de la ligne du cache où on peut

1. Source : http://www.liafa.jussieu.fr/~amicheli/Ens/Archi/td11.pdf

la trouver ainsi que l'étiquette (INDICATEUR) de cette ligne et l'offset (OCTET) de la donnée correspondante dans le cache. *Certains calculs fastidieux peuvent être évités...*

Solution: Une fois qu'on a reproduit le schéma du cours, tout cela est plus simple, on découpe l'adresse en petits bouts

$$(A23847EF)_{16} = (10100010001110000100|01111|1101111)_2$$

— Les 7 bits de poids faibles forment l'offset : $(1101111)_2 = (6F)_{16}$
— L'indicateur est formé des 20 bits de poids fort, et comme $20 = 4 * 5$ il est facile de récupérer l'hexadécimal correspondant : $(A2384)_{16}$.
— Le numéro de la ligne de cache est formé par les 25 bits de poids fort, et sauf erreur de calcul on trouve $(144708F)_{16}$.

Question 3:
Si la donnée correspondant à l'adresse $(A23847EF)_{16}$ considérée est effectivement dans le cache, donnez (en hexadécimal) les adresses qui seront stockées dans la même ligne du cache que celle-ci. *On pourra se contenter d'une réponse où les calculs ne sont pas effectués jusqu'au bout.*

Solution: Il s'agit des adresses entre $(144708F)_{16} \times 2^7 + 0$ (octet = 0000000) à $(144708F)_{16} \times (2^8 - 1)$ (octet = 1111111).

3 Programmation LC3

L'objet de l'exercice est d'écrire un programme LC3 [2] qui permet de déterminer si une chaîne donnée est un palindrome, *i.e.* son retournement est égal à lui-même. L'algorithme utilisé sera le suivant :

2. adapté de http://home.wlu.edu/~lambertk/classes/210/exercises/hw7.htm

```
bool isPalindrome(char *string){
    int left,right ;
    left = 0 ;
    right = len(string) - 1 ;
    while (left < right){
        if (string[left] != string[right])
            return(false);
        left += 1;
        right -= 1;
    }
    return(true)
}
```

On suppose écrite la routine suivante :

`len` : l'adresse du premier caractère d'une chaîne étant stockée dans le registre R_1, cette routine calcule et place la taille de la chaîne dans le registre R_2.

Question 1:

Expliquer sur des exemples/dessins comment fonctionne cet algorithme.

Solution: On peut judicieusement regarder les chaînes $anna$, et $aibia$, ainsi qu'un mot qui n'est pas un palindrôme. L'algorithme consiste à lire le mot de gauche à droite (variable `left`) et de droite à gauche (variable `left`) et à quitter la procédure avec `false` dès que les lettres de droite et de gauche sont différentes.

La routine considérera que l'adresse de base de la chaîne est stockée dans R_1, que sa longueur est stockée dans R_2, et stockera le résultat dans le registre R_0 (1 pour vrai, 0 pour faux).

Question 2:

Remplir les trous du pseudo-code-LC3 suivant :

```
R2 <- R1 + R2 - 1;
 while(R2 > R1) {
   if(................) {
     R0 <-  ..................
     ............
   }
   R1++;
   R2--;
 }
  R0 <-  ..........
  return;
}
```

et commentez-le (dire en particulier ce que stockent les différents registres).

Question 3: $\star$

Écrire la routine en assembleur LC-3.

Solution: Et le code LC3 correspondant :

```
ispal:
        ADD R2, R1, R2
        ADD R2, R2, -1
loopispal:
```

```
        NOT R5, R1
        ADD R5, R5, 1       ; R5 <- -R1
        ADD R5, R2, R5      ; R5 <- R2 - R1
        BRnz endloopispal   ; condition de sortie de boucle : R2 - R1 <= 0
        LDR R3, R1, #0      ; R3 <- mem[R1]
        LDR R4, R2, #0      ; R4 <- mem[R2]
        NOT R5, R4
        ADD R5, R5, 1       ; R5 <- -R4
        ADD R5, R3, R5      ; R5 <- R3 - R4 = mem[R1] - mem[R2]
        BRz endifispal      ; si mem[R1] - mem[R2] == 0 saut
        AND R0, R0, 0       ; R0 <- 0
        RET
endifispal:
        ADD R1, R1, 1       ; R1++
        ADD R2, R2, -1      ; R2--
        BR loopispal
endloopispal:
        AND R0, R0, 0
        ADD R0, R0, 1       ; R0 <- 1
        RET
```

4 Circuit combinatoire et nouvelle instruction LC3

Le but de cet exercice est d'ajouter une nouvelle instruction à notre processeur LC3 construit en TP[3]. L'instruction réalisera des opérations de rotation circulaire ("shift"). Nous allons donc réaliser un circuit combinatoire qui réalise cette opération, puis modifier le contrôle.

On rappelle que les registres du LC3 sont des registres 16 bits, les bits étant numérotés de 0 à 15 (le bit de poids fort est le bit numéro 15). La rotation gauche consiste à permuter 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 gauche de k positions consiste à effectuer k rotations gauches.

Question 1:

Circuit pour les décalages

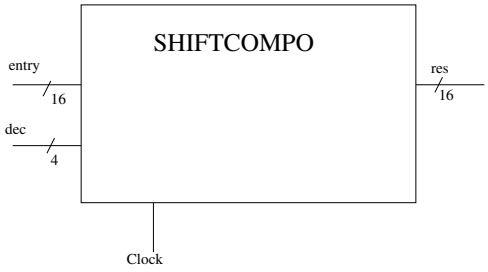
(a) Considérons que $R_1 = (0110001111111110)$, quel est le résultat d'une rotation gauche ? De deux rotations gauches ?

(b) Concevoir un circuit qui effectue une rotation gauche. *On pourra utiliser les composants* `splitter` *de logisim et judicieusement remarquer qu'il s'agit uniquement de copies de bits*

(c) Écrire un circuit commandé par un bit qui effectue une rotation gauche si ce bit de commande vaut 1 et qui n'effectue aucune rotation sinon.

3. d'après http://www.liafa.univ-paris-diderot.fr/~carton/Enseignement/Architecture/Projet/2014-2015/projet.pdf

Dans la suite on suppose qu'on dispose du composant `SHIFTCOMPO` suivant :



qui permet d'effectuer une rotation gauche de 0 à 15 positions. L'objet de la suite est d'ajouter l'instruction `SHT` à notre assembleur LC3, codée de la manière suivante :

15			12	11	9	8	6	5	4	3	2	0
1	1	0	1	DR		SR1		0	0	0	SR2	
1	1	0	1	DR		SR1		1	Imm5			

qui réalise l'opération de shift à gauche à partir de l'entrée `SR1`. Les deux modes d'adressage sont calqués sur ceux de `AND` :

— La première ligne correspond à l'adressage registre : DR reçoit SR1 auquel on a appliqué une rotation (gauche) de SR2 positions.

— la deuxième ligne correspond à l'adressage immédiat : DR reçoit SR1 auquel on a appliqué une rotation (gauche) de Imm5 positions.

Question 2:

Pourquoi le composant `SHIFTTCOMPO` suffit-il à gérer tous les décalages possibles induits par les valeurs respectives de `Imm5` et le contenu de `SR2` (16 bits) ? *On distinguera les valeurs négatives et les valeurs positives.*

Question 3:

Sur le schéma du LC3 fourni en annexe (Figure 1), surlignez le *chemin de données* correspondant au traitement de cette instruction (version avec `Imm5`).

Question 4:

Ajouter le composant `SHIFTCOMPO` dans la partie ALU.

Question 5: Bonus

En justifiant, ajouter les liens avec les différents autres composants (Mémoire, Registres, Décodage, Calcul du PC…).

Question 6: $\star$

À l'aide de cette nouvelle instruction, écrire un programme LC3 qui permet de multiplier par 12 le registre R_1. Combien gagne-t-on d'instructions par rapport à la multiplication naïve ?

Appendix

Feuille réponse à rendre avec votre copie

Bien noter votre numéro de copie sur cette feuille réponse.

Réponse exercice 3.2

```
R2 <- R1 + R2 - 1;
 while(R2 > R1) {
   if(................) {
     R0 <-  ................
     ............
   }
   R1++;
   R2--;
 }
  R0 <-  ..........
  return;
}
```

Réponses exercice 4

(page suivante)

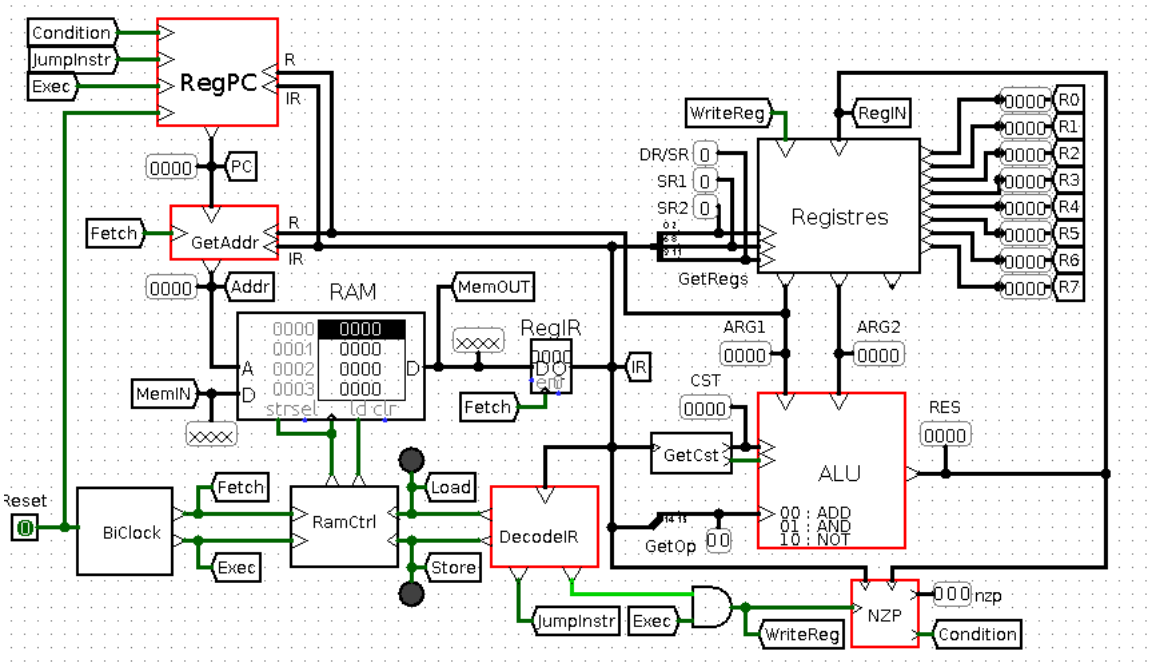


FIGURE 1 – Vue globale du LC3 de TP.

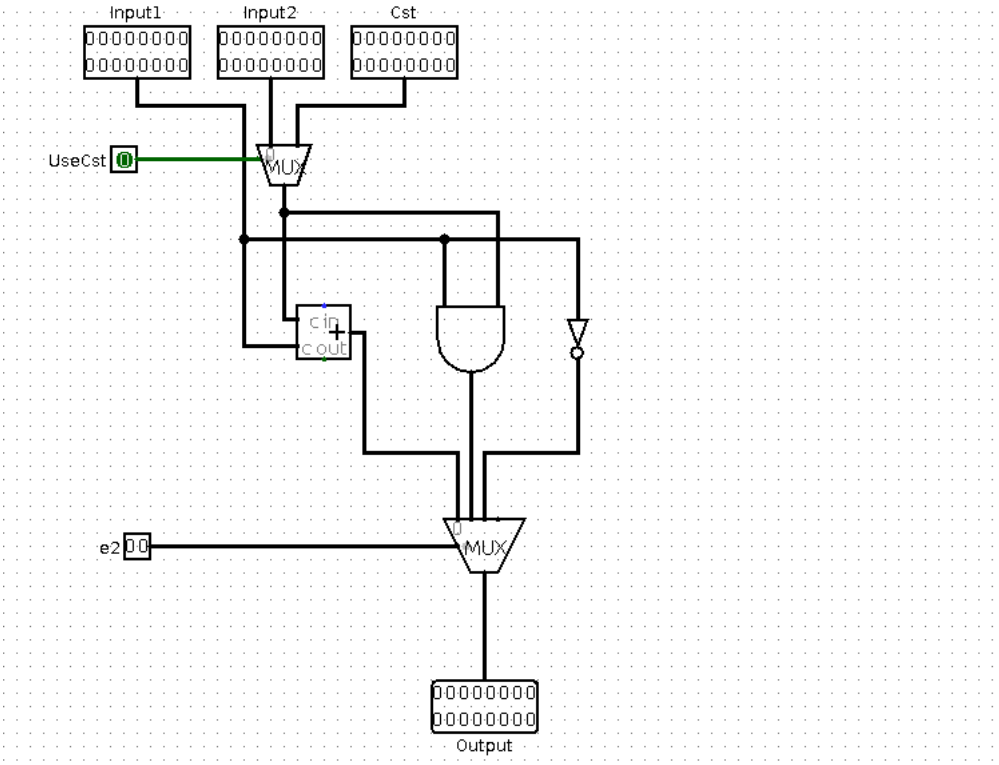


FIGURE 2 – Composant ALU du LC3 de TP.