

CC-CM-F - LIF6 Architecture matérielle et logicielle

Aucun document autorisé, durée 45 min, barème donné à titre indicatif.

Merci de respecter le plus possible l'ordre des questions sur votre copie.

I Questions de cours (8 points)

Q.I.1) - (3 pts) Dans le cours *ébauche d'un processeur*, nous avons mis au point le chemin de données rappelé ci-dessous. Rappelons également la sémantique des instructions LD et ST :

– LD : $\text{Reg}[\text{IR.r1}] \leftarrow \text{Mem}[\text{Reg}[\text{IR.r0}] + \text{add}]$,

– ST : $\text{Mem}[\text{Reg}[\text{IR.r0}] + \text{add}] \leftarrow \text{Reg}[\text{IR.r1}]$.

Détaillez, sous forme pseudo-algorithmique, les actions qui doivent être exécutées à chaque cycle d'horloge de l'exécution de chacune de ces deux instructions. Dans les deux cas, précisez à quel cycle (en partant de « c0 ») chaque action algorithmique doit être exécutée.

– LD :

```

c0 :      IR  ←  Mem[PC]
c0 :      PC  ←  PC+4
c1 :      A   ←  Reg[IR.r0]
c2 :  ALUOut ←  A + SgnExt(IR.add)
c3 :      MDR ←  Mem[ALUOut]
c4 :  Reg[IR.r1] ←  MDR

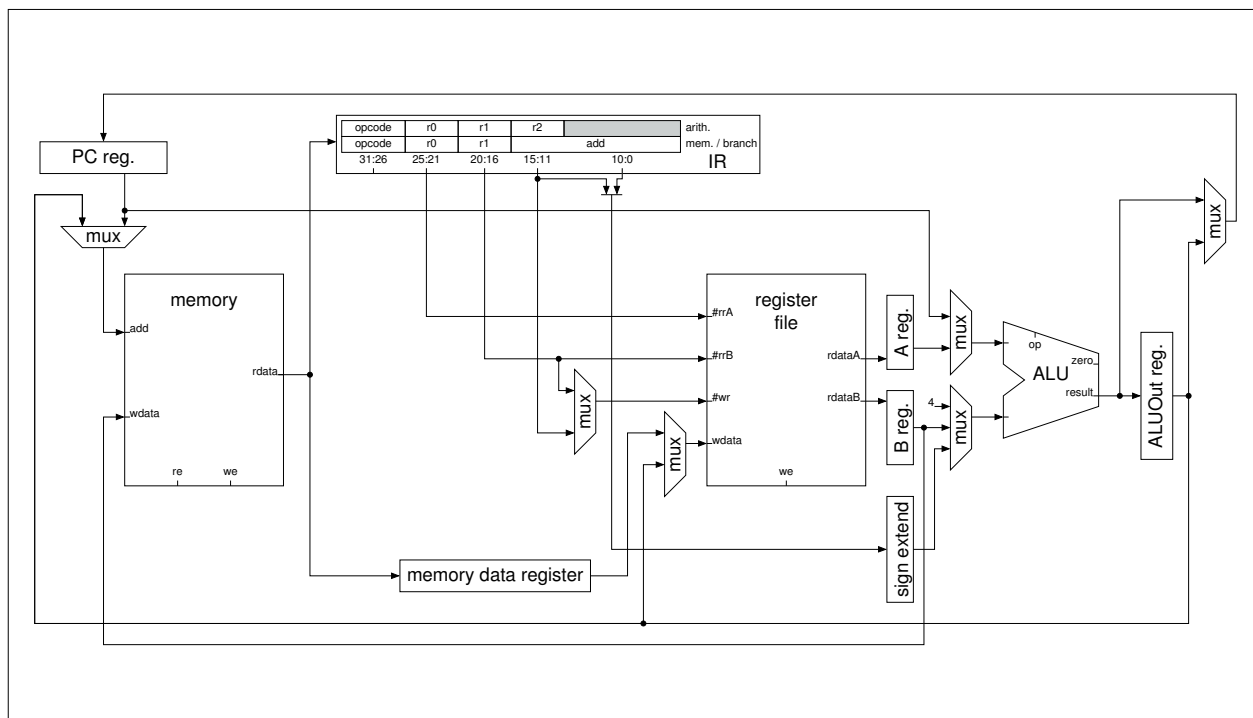
```

– ST :

```

c0 :      IR  ←  Mem[PC]
c0 :      PC  ←  PC+4
c1 :      A   ←  Reg[IR.r0]
c2 :  ALUOut ←  A + SgnExt(add)
c2 :      B   ←  Reg[IR.r1]
c3 :  Mem[ALUOut] ←  B

```



Q.I.2) - (3 pts) Après avoir rappelé brièvement comment sont codées les instructions, définissez ce qu'est le *mode d'adressage* des opérandes d'une instruction. Donnez trois exemples de modes d'adressages, en illustrant chacun par une instruction en langage d'assemblage.

Chaque instruction est codée par un mot binaire, qui se décompose en :

- un champ *opcode*, qui spécifie l'opération à réaliser ;
- éventuellement, un ou plusieurs champs *opérandes* qui définissent les emplacements où l'instruction doit lire ses sources et écrire son résultat.

La méthode de localisation des opérandes, dans la mémoire ou parmi les registres, est appelé *mode d'adressage* : on assimile à une adresse le champ de bit correspondant un chaque opérande. En d'autres termes, le *mode d'adressage* est la manière d'interpréter les bits d'un champ d'adresse en vue de la localisation de l'opérande. Une instruction qui présente k champs d'adresse est dite *instruction à k adresses*.

- `MOV R1,4` : on parle d'*adressage par registre* pour l'opérande 1, car l'emplacement désigné est simplement un registre ; on parle d'*adressage immédiat* pour l'opérande 2, car il est directement placé dans le champ d'adresse de l'instruction.
- `ADD R2,@96` : on parle d'*adressage direct* pour l'opérande 2, car le champ d'adresse désigne directement un mot en mémoire par son adresse.

Q.I.3) - (2 pts) Comment se caractérise un jeu d'instruction CISC (*Complex Instruction Set Computer*) ?

Un jeu d'instructions CISC se caractérise par

- un très grand nombre d'instructions, dont certaines très spécialisées ;
- des instructions de tailles variables ;
- les opérations arithmétiques et logiques sont susceptibles de faire des accès à la mémoire ;
- la conséquence du point précédent est l'existence de multiples modes d'adressages pour les opérandes des instructions.

II Programmation en assembleur (7 pts)

On considère le programme (incomplet) ci-dessous, écrit dans le langage d'assemblage du LC3. La routine `saisie` doit permettre de saisir une chaîne de caractères au clavier, en rangeant les caractères lus à partir de l'adresse contenue dans le registre `R1` lors de son appel. La saisie prend fin lorsque le caractère `CR` (le code ASCII de `CR` est 13) est lu, et un `0` est rangé en fin de chaîne pour en marquer la fin.

```

1      .ORIG x3000
2  ; Programme principal
3      LEA R6,stackend
4      ; affiche la chaine à l'adresse msg1
5      LEA R0,msg1
6      PUTS
7      ; saisie d'une chaine à l'adresse ch1
8      LEA R1,ch1
9      JSR saisie
10     ; affiche la chaine à l'adresse msg2
11     LEA R0,msg2
12     PUTS
13     ; affiche la chaine à l'adresse ch1
14     LEA R0,ch1
15     PUTS
16     HALT
17 ; Données
18 msg1:  .STRINGZ "Entrez une chaine : "
19 msg2:  .STRINGZ "Vous avez tapé : "
20 ch1:   .BLKW #8
21 stack: .BLKW #32
22 stackend:
23 ; routine pour saisir une chaine de caractères
24 ; paramètre d'entrée : l'adresse R1 du début de chaîne
25 saisie: ADD R6,R6,#-1
26         STR R7,R6,#0
27         AND R2,R2,#0
28         ADD R2,R1,#0      ; on copie R1 dans R2
29 loop:   GETC              ; lit un caractère au clavier, résultat dans R0
30         OUT               ; affiche le caractère contenu dans R0
31         ...               ; calcule la différence entre CR et le caractère lu
32         ...               ; si la différence est nulle, on sort de la boucle
33         ...               ; sinon, on stocke le caractère lu à l'adresse pointée par R2
34         ...               ; on incrémente le pointeur R2 en vu de l'itération suivante
35         ...               ; on passe à l'itération suivante
36 end:    ...
37         ...               ; on s'assure de placer un 0 en fin de chaine
38         LDR R7,R6,#0
39         ADD R6,R6,#1
40         RET
41         .END

```

Q.II.1) - (0.5 pt) De façon générale, quel est le rôle du registre R6 ?

R6 est utilisé comme pointeur vers le sommet de la pile d'exécution

Q.II.2) - (0.5 pt) Que fait l'instruction `LEA R6,stackend` à la ligne 3 du programme ?

Elle charge l'adresse désignée par le label `stackend` dans R6 : cela initialise le pointeur de pile.

Q.II.3) - (0.5 pt) De façon générale, que fait l'instruction `JSR`, et quel est le rôle du registre R7 ?

Lors de l'appel d'une routine, `JSR` sauvegarde l'adresse du retour de l'appel (adresse de l'instruction `JSR` en question plus un) dans le registre R7.

Q.II.4) - (0.5 pt) De façon générale, que fait l'instruction `RET` ?

L'instruction `RET` effectue un saut incondtionnel à l'adresse de retour contenue dans R7.

Q.II.5) - (0.5 pt) Expliquez le rôle joué par les deux instructions aux lignes 25 et 26 du programme ?

Elles permettent de sauvegarder l'adresse de retour de la routine sur la pile d'exécution.

Q.II.6) - (0.5 pt) Expliquez le rôle joué par les deux instructions aux lignes 38 et 39 du programme ?

Elles permettent de restaurer l'adresse de retour qui a été sauvegardé sur la pile au début de l'exécution de la routine.

- Q.II.7)** - (0.5 pt) Justifiez le fait que la constante décimale #-13 est représentable sous la forme d'un entier codé sur 5 bits en complément à 2.

$$(-13)_{10} = (-16)_{10} + (3)_{10} = (10011)_{\bar{2}}$$

- Q.II.8)** - (0.5 pt) L'instruction `ADD R4,R0,#-13` est-elle une instruction valide ? Pourquoi ?

Le troisième opérande d'une instruction `ADD` peut être un `Imm5` : comme #-13 est bien représentable sous la forme d'un immédiat en complément à 2 sur 5 bits, l'instruction est bien valide.

- Q.II.9)** - (3 pts) Donnez sur votre copie le code des lignes 29 à 37 de la routine `saisie`.

```
; routine pour saisir une chaine de caractères
; paramètre d'entrée : l'adresse R1 du début de chaîne
saisie:  ADD R6,R6,#-1
        STR R7,R6,#0    ; on empile l'adresse de retour
        AND R2,R2,#0
        ADD R2,R1,#0    ; on copie R1 dans R2
loop:   GETC            ; lit un caractère au clavier, résultat dans R0
        OUT             ; affiche le caractère lu
        ADD R4,R0,#-13  ; calcule la différence entre CR et le caractère lu
        BRz end         ; si la différence est nulle, on sort de la boucle
        STR R0,R2,#0    ; sinon, on stocke le car. lu à l'ad. pointée par R2
        ADD R2,R2,#1    ; on incrémente le pointeur en vu de l'itération suivante
        BR loop
end:     AND R0,R0,#0
        STR R0,R2,#0    ; on s'assure de placer un 0 en fin de chaîne
        LDR R7,R6,#0
        ADD R6,R6,#1    ; on restaure l'adresse de retour
        RET
```

III Questions à réponses brèves (5 pts)

- Q.III.1)** - Qu'est-ce que le chemin de données d'un processeur ?

Il s'agit de l'ensemble des circuits dédiés au traitement des données, et donc aussi au traitement des instructions.

- Q.III.2)** - Dans une certaine architecture, les instructions arithmétiques ne prennent aucune adresse : comment ces instructions peuvent être utilisées ?

Le processeur dispose d'une pile, et une instruction prend ses deux opérandes au sommet de la pile et y range le résultat.

- Q.III.3)** - Qu'appelle-t-on la compatibilité ascendante d'une architecture ?

Lorsque l'on met au point une nouvelle architecture en se basant sur une architecture existante, on peut ajouter de nouvelles instructions, mais pas en enlever, si on souhaite que les programmes compilés pour l'ancienne architecture puissent toujours s'exécuter sur la nouvelle.

- Q.III.4)** - Qu'appelle-t-on les registres architecturaux ?

Ce sont les registres qui sont documentés dans une architecture, et directement accessibles au programmeur.

- Q.III.5)** - Un banc de 32 registres 16 bits comporte 3 ports de lecture et un port d'écriture : combien de fils dédiés à l'identification des numéros de registre comporte-t'il ?

$$32 = 2^5, \text{ donc 5 fils par port, soit } 4 \times 5 = 20 \text{ fils dédiés à l'identification des numéros de registre.}$$

- Q.III.6)** - Une unité arithmétique et logique (UAL) prend deux mots de 16 bits en entrée, et fournit un résultat sur 16 bits. Elle est capable d'effectuer 3 opérations (`ADD`, `NOT`, `AND`) distinctes. Combien de fils de contrôle cette UAL doit-elle prendre en entrée ?

$$2^1 = 3 < 7 \leq 4 = 2^2, \text{ donc 2 fils sont nécessaires pour choisir un opération parmi 3.}$$

- Q.III.7)** - Quelles sont les trois principales classes d'instructions d'un jeu d'instruction RISC, comme le LC-3 ?

- opérations arithmétiques et logiques,
- accès mémoire (Load ou Store) ou branchement conditionnel,
- les instructions de saut.

Q.III.8) - Qu'est-ce qu'un label dans un langage d'assemblage ?

Le langage d'assemblage permettent d'utiliser des labels pour désigner les adresses. L'assembleur traduira chaque label en une adresse effective lors de l'assemblage.

Q.III.9) - Sur le LC-3, quelque est l'interêt de l'instruction JMP, alors que l'on dispose déjà de BR ?

Avec JMP, on peut réaliser un saut n'importe où dans la mémoire.

Q.III.10) - Dans un programme écrit en langage d'assemblage, qu'appelle-t-on une référence-avant ?

C'est un label qui est utilisé comme opérande d'une instruction avant d'avoir été défini, dans l'ordre de lecture du programme.

Récapitulatif des instructions du LC-3 utiles/utilisables dans ce sujet :

syntaxe	action	nzp	codage																
			opcode				arguments												
			F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
ADD DR,SR1,SR2	DR <- SR1 + SR2	*	0 0 0 1				DR			SR1			0	0 0		SR2			
ADD DR,SR1,Imm5	DR <- SR1 + SEXT(Imm5)	*	0 0 0 1				DR			SR1			1	Imm5					
AND DR,SR1,Imm5	DR <- SR1 and SEXT(Imm5)	*	0 1 0 1				DR			SR1			1	Imm5					
LEA DR,label	DR <- PC + SEXT(PCoffset9)	*	1 1 1 0				DR			PCoffset9									
LD DR,label	DR <- mem[PC + SEXT(PCoffset9)]	*	0 0 1 0				DR			PCoffset9									
ST SR,label	mem[PC + SEXT(PCoffset9)] <- SR		0 0 1 1				SR			PCoffset9									
LDR DR,BaseR,Offset6	DR <- mem[BaseR + SEXT(Offset6)]	*	0 1 1 0				DR			BaseR			Offset6						
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] <- SR		0 1 1 1				SR			BaseR			Offset6						
BR[n][z][p] label	Si (cond) PC <- PC + SEXT(PCoffset9)		0 0 0 0				n	z	p	PCoffset9									
RET (JMP R7)	PC <- R7		1 1 0 0				0 0 0			1 1 1			0 0 0 0 0 0						
JSR label	R7 <- PC; PC <- PC + SEXT(PCoffset11)		0 1 0 0				1	PCoffset11											