Examen d'Architecture des Ordinateurs
Majeure 1 – École Polytechnique

2004–2005

- L'examen dure 3 heures.
- Le sujet comporte 7 pages dont plusieurs rappels sur le LC2.
- Tous documents autorisés.
- Le barème est donné à titre indicatif, il sert surtout à évaluer le poids respectif des sections ; tel quel, l'examen est noté sur 20 points plus une question facultative sur 4 points.
- L'examen contient deux exercices distincts que vous devez rendre sur **2 copies séparées**.
- Il est impératif de commenter les programmes en assembleur et les microprogrammes ; la plupart des instructions doivent être suivies d'un commentaire permettant de comprendre leur rôle.

Problème 1 - Extension microprogrammable du LC-2 (14 points)

On utilise le schéma du processeur LC-2 (hors gestion des interruptions) de la Figure 1.
Note : les circuits `PC`, `IR`, `MAR`, `MDR`, `N`, `Z` et `P` sont des registres.

Le rôle des signaux de contrôle de la Figure 1 est explicité dans la liste suivante.

- $LD.MAR_{/1}$, $LD.MDR_{/1}$, $LD.IR_{/1}$, $LD.REG_{/1}$, $LD.CC_{/1}$ et $LD.PC_{/1}$ commandent l'écriture dans les divers registres du LC-2.
- $LD.BEN_{/1}$ commande l'écriture dans le registre BEN (*branch enable*) ; ce registre vaut 1 lorsque le branchement doit être pris (si l'instruction courante est un branchement conditionnel).
- $GatePC_{/1}$, $GateMDR_{/1}$, $GateALU_{/1}$ et $GateMARMX_{/1}$ commandent les accès en écriture sur le bus.
- $MIO.EN_{/1}$ doit être mis à 1 lorsque l'on souhaite accéder à la mémoire, en lecture ou en écriture.
- $R.W_{/1}$ est mis à 0 pour une lecture et 1 pour une écriture en mémoire.
- $ALUK_{/2}$: 00 pour `ADD`, 01 pour `AND`, 10 pour `NOT`, 11 pour faire « traverser » l'entrée 1 sans calcul.
- $PCMX_{/2}$ sélectionne l'une des quatre entrées du multiplexeur : de *droite* à *gauche*, 00, 01, 10 et 11.
- $MARMX_{/2}$ sélectionne l'une des trois entrées du multiplexeur : de *gauche* à *droite*, 00, 01 et 10 (11 est inutilisé).
- $SR1MX_{/2}$ sélectionne l'une des quatre entrées du multiplexeur : de *haut* en *bas*, 00 et 01 (10 et 11 ne sont pas utilisés dans cet exercice).
- $DRMX_{/2}$ vaut toujours 0 dans le cadre de cet exercice, il sélectionne le registre destination (signal DR) à partir de $IR[11:9]$.
- $SR2MX_{/1}$ est à part : ce signal détermine si la deuxième opérande vient du banc des registres ou du champ immédiat, il est par construction égal au bit 5 du registre d'instruction IR et n'intervient pas dans la microprogrammation.

On dispose du même *microcontrôleur* que celui utilisé au TD 5. Il permet de réaliser le contrôle microprogrammé du LC-2 à l'aide de microinstructions simples et de branchements conditionnels ou non. Il comporte 8 signaux de condition de branchement : le signal $\overline{BEN}$ (qui vaut 1 lorsque le branchement ne doit *pas* être pris) est sélectionné lorsque $Condition_{/3} = 000$, les 5 bits de poids fort du registre d'instruction IR sont respectivement sélectionnés lorsque $Condition_{/3}$ prend des valeurs de 011 à 111. On ajoute un nouveau signal de condition N, sélectionné lorsque $Condition_{/3} = 001$, qui est égal à la valeur du registre `N` des codes condition. Enfin, le signal associé à $Condition_{/3} = 010$ n'est pas utilisé. En sortie, ce microcontrôleur est capable de générer les 23 signaux de contrôle du LC-2. Les adresses des microinstructions sont codées sur 8 bits : $J[7:0]$ indique l'adresse de l'instruction suivante en cas de branchement. Les microinstructions sont codées sur 48 bits et leur format détaillé est indiqué Figure 2 (les `x` correspondent à des signaux inutilisés, disponibles pour étendre le microcontrôleur).

On rappelle également la liste des instructions du LC-2 (dans sa version simplifiée) en Figure 3.

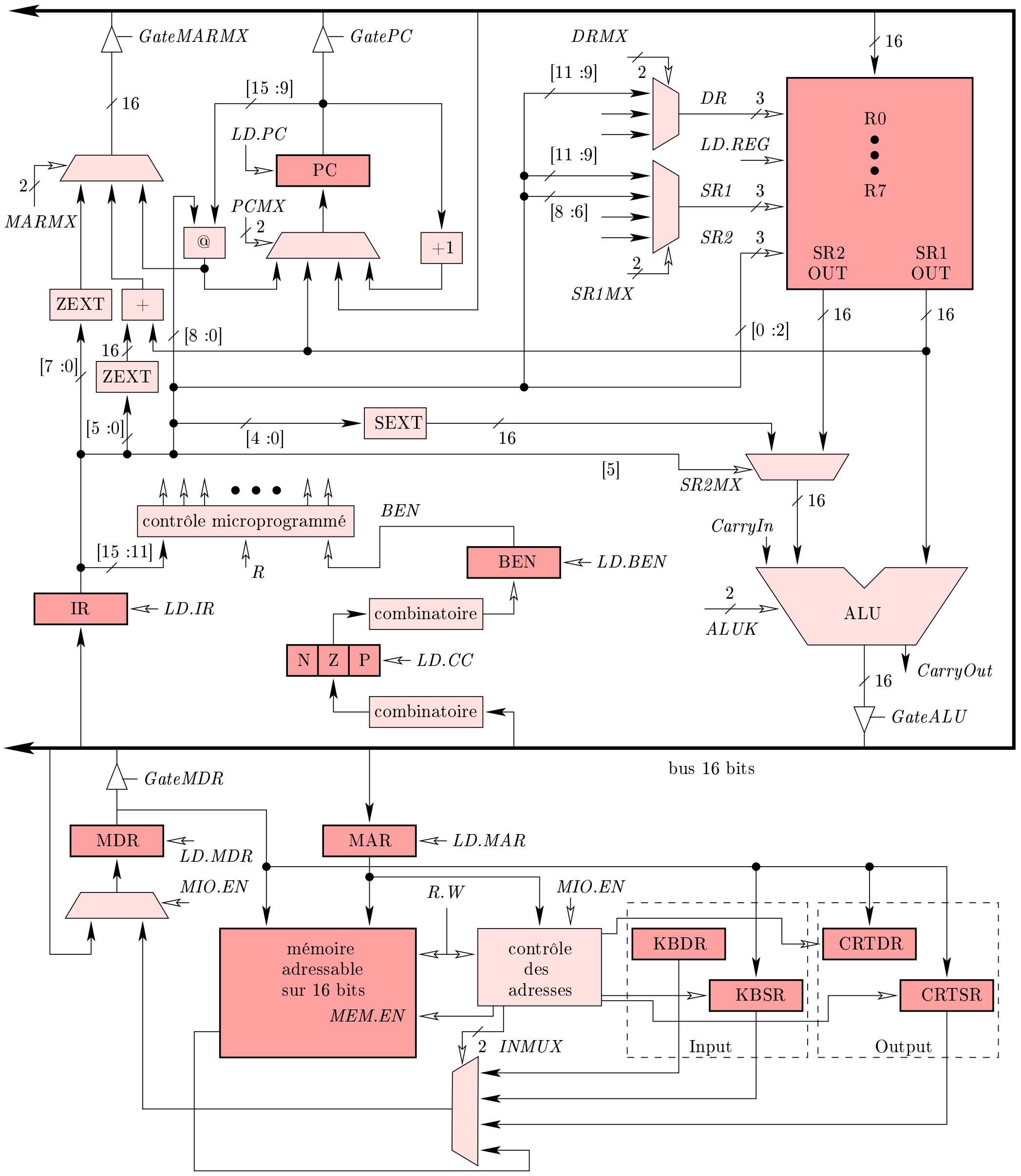

FIG. 1 – Diagramme des blocs du LC-2

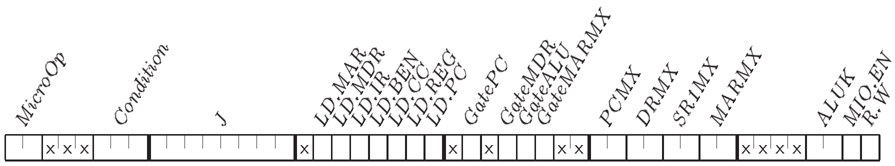

FIG. 2 – Format des micro-instructions

À titre d'exemple, on rappelle le microprogramme pour l'exécution complète de l'instruction `ADD`, vu au TD 5. L'exécution commence à l'adresse 0, et les 4 premières microinstructions ne sont pas spécifiques à l'instruction `ADD`. Les microinstructions effectuant un branchement conditionnel sont associées aux états ????, 0???, 00?? et 000? : à l'aide de 4 microinstructions conditionnelles successives, on saute à l'état $DR \leftarrow SR1 + SR2\,/\,SEXT, CC$ lorsque l'opcode (0001) correspond à l'instruction `ADD`. Les microinstructions et leurs adresses sont décrites dans le tableau de la Figure 4 (toutes les valeurs sont en binaire, sauf l'état).

Les 4 microinstructions de branchement conditionnel testent successivement les 4 bits de poids fort

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`ADD` DR, SR1, SR2	0	0	0	1		DR			SR1		0	0	0		SR2	
`ADD` DR, SR1, imm5	0	0	0	1		DR			SR1		1	imm5 : immédiat 5-bits signé				
`AND` DR, SR1, SR2	0	1	0	1		DR			SR1		0	0	0		SR2	
`AND` DR, SR1, imm5	0	1	0	1		DR			SR1		1	imm5 : immédiat 5-bits signé				
`NOT` DR, SR	1	0	0	1		DR			SR		1	1	1	1	1	1
`BR`nzp label	0	0	0	0	n	z	p	offset 9-bits non signé dans la page courante								
`JMP` label	0	1	0	0	0	0	0	offset 9-bits non signé dans la page courante								
`JSR` label	0	1	0	0	1	0	0	offset 9-bits non signé dans la page courante								
`JMPR` offset	1	1	0	0	0	0	0	BaseR			index 6-bits non signé					
`JSRR` offset	1	1	0	0	1	0	0	BaseR			index 6-bits non signé					
`LEA` DR, label	1	1	1	0		DR		offset 9-bits non signé dans la page courante								
`LD` DR, label	0	0	1	0		DR		offset 9-bits non signé dans la page courante								
`LDR` DR, BaseR, offset	0	1	1	0		DR		BaseR			index 6-bits non signé					
`ST` SR, label	0	0	1	1		SR		offset 9-bits non signé dans la page courante								
`STR` SR, BaseR, offset	0	1	1	1		SR		BaseR			index 6-bits non signé					
`RET`	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 3 – Format des instructions du LC-2

Etat	Adresse	Opcode xxx Signaux de condition	Adresse suivante	*LD.?*	*Gate?*	*?MX*	Divers
$MAR \leftarrow PC, PC \leftarrow PC+1$	00000000	01xxx000	00000000	x1000001	x1x000xx	00000000	xxxx0000
$MDR \leftarrow MAR$	00000001	01xxx000	00000000	x0100000	x0x000xx	00000000	xxxx0010
$IR \leftarrow MDR$	00000010	01xxx000	00000000	x0010000	x0x100xx	00000000	xxxx0000
opcode ????, $[\overline{BEN}]$	00000011	11xxx111	00000111	x0001000	x0x000xx	00000000	xxxx0000
opcode 0???	00000100	11xxx110	00000111	x0000000	x0x000xx	00000000	xxxx0000
opcode 00??	00000101	11xxx101	00000111	x0000000	x0x000xx	00000000	xxxx0000
opcode 000?	00000110	11xxx100	00001000	x0000000	x0x000xx	00000000	xxxx0000
Stop	00000111	00xxx000	00000000	x0000000	x0x000xx	00000000	xxxx0000
opcode 0001, $DR \leftarrow SR1 + SR2 / SEXT, CC$	00001000	10xxx000	00000000	x0000110	x0x010xx	00000100	xxxx0000

Fig. 4 – Microprogramme de l'instruction `ADD`

du registre `IR`, en posant $Condition_{/3} = 111$, puis $Condition_{/3} = 110$, puis $Condition_{/3} = 101$ et enfin $Condition_{/3} = 100$. L'addition proprement dite a lieu à la microinstruction d'adresse 00001000. L'adresse 00000111 implémente la microinstruction *Stop* ; elle est atteinte lorsque l'instruction à exécuter n'est pas une addition.

Nous étendons le jeu d'instructions du LC-2 avec une opération de multiplication. Celle-ci multiplie deux opérandes sur 16 bits, *non signées*, et donne un résultat sur 32 bits, stocké dans deux registres 16 bits. Comme les codes condition n'ont pas de sens pour une opération non signée, la valeur des registres `N`, `Z` et `P`, en fin d'exécution de l'instruction de multiplication, n'a pas d'importance. Cette nouvelle instruction est identifiée par l'opcode 1010 (celui de l'instruction `LDI`, inutilisée dans le cadre de cet exercice), et a le format suivant :

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`MUL` *DR1, DR2, SR1, SR2*	1	0	1	0		*DR1*			*SR1*			*DR2*			*SR2*	

SR1 et *SR2* sont les numéros des registres codant, respectivement, le multiplicande (la valeur multipliée)

et le multiplieur ; le registre *SR2* peut être modifié au cours de l'exécution de l'instruction `MUL`. *DR1* et *DR2* sont les numéros de registres codant, respectivement, les bits de poids fort et de poids faible du résultat.

Attention : pour simplifier la réalisation, on impose également que *SR1*, *SR2*, *DR1* et *DR2* correspondent à des registres distincts.

Question 1.1

Cette nouvelle instruction utilise 4 opérandes. Expliquez comment il faut modifier les multiplexeurs de contrôle du banc de registres pour cette instruction.

Question 1.2

Pour effectuer efficacement des additions sur 32 bits, on ajoute un registre `C` de 1 bit pour conserver la retenue en sortie de l'additionneur de l'ALU (*CarryOut*), et la réinjecter en entrée de celui-ci (*CarryIn*). Représentez les modifications à apporter au chemin de données du LC-2, en décrivant les nouveaux signaux de contrôle (faire un schéma de la partie modifiée du LC-2).

Question 1.3

Expliquez comment effectuer une addition sur 32 bits à l'aide de la retenue.

Question 1.4

Malgré l'absence d'unité de décalage, sans accéder à la mémoire, et sans autres modifications du chemin de données du LC-2, expliquez comment effectuer l'instruction `MUL` avec au plus 64 additions sur 16 bits (dont 16 additions sur 32 bits). On rappelle que $ALUK_{/2} = 00$ configure l'ALU en mode addition et que $ALUK_{/2} = 11$ permet de faire traverser la première opérande de l'ALU sans calcul. Les corrections du TD 6 pourront donner des indications.

Question 1.5

Implémentez le microprogramme pour l'exécution de la multiplication, sans décrire le décodage de l'op-code de l'instruction, et en ne précisant que les signaux de contrôle dont le rôle est important à un cycle donné (pour l'instruction `ADD`, par exemple, cela correspondrait uniquement à la dernière microinstruction).

Question 1.6

Le microprogramme de la question précédente exécute combien de microinstructions dans le pire cas ? dans le meilleur cas ?

On généralise l'approche suivie pour la microprogrammation de la multiplication non signée. En effet, de nombreux algorithmes simples et réguliers, pour le traitement du signal ou le calcul algébrique, peuvent être accélérés en les implémentant directement comme le microprogramme d'une instruction spécialisée.

Pour cela, on suppose que la *mémoire de microprogramme* est découpée en deux, selon le schéma de la Figure 5 : un premier segment de 128 microinstructions (codées sur 48 bits) situé à l'adresse 0 et un deuxième segment de 128 microinstructions (toujours sur 48 bits) situé à l'adresse 128. Le premier segment est la ROM codant le microprogramme des instructions du LC-2. Le deuxième segment est une RAM dans laquelle des programmes du LC-2 peuvent charger et exécuter un microprogramme quelconque.

Pour utiliser ce nouveau segment, on introduit deux nouvelles instructions :

- `LM` *SR1*, *SR2* (*load microprogram*) : charge le microprogramme débutant à l'adresse indiquée par le registre *SR1* dans la RAM de microprogramme (située à partir de l'adresse 128 de la mémoire de microprogramme) ; *SR2* est le registre contenant le nombre de microinstructions à charger de cette manière *moins 1* (i.e., la valeur maximale étant 127, on peut avoir au plus 128 microinstructions, soit 384 mots de 16 bits) ;
- `EM` *DR1*, *DR2*, *SR1*, *SR2* (*execute microprogram*) : exécute le microprogramme à l'adresse 128 de la mémoire de microprogramme, le registre `IR` contenant les 16 bits de l'instruction `EM` tout au long de l'exécution.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`EM` *DR1*, *DR2*, *SR1*, *SR2*	1	0	1	0		*DR1*			*SR1*			*DR2*			*SR2*	
`LM` *SR1*, *SR2*	1	0	1	0	0	0	0		*SR1*		0	0	0		*SR2*	

L'instruction `EM` remplace l'instruction `MUL` des questions précédentes.

Le microprogramme chargé et exécuté par ces instructions est libre, à condition qu'il se termine par un *branchement inconditionnel à l'adresse 0* (afin de poursuivre l'exécution du LC-2 avec la prise en charge

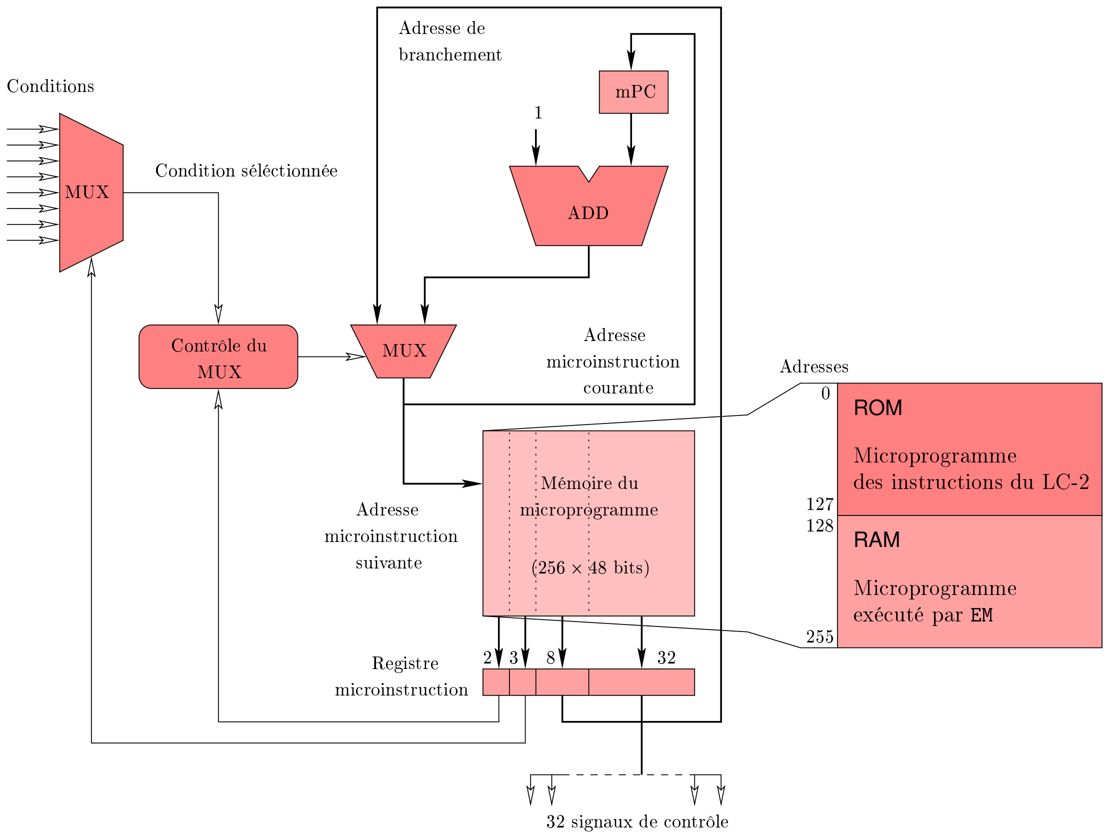

FIG. 5 – Schéma du microcontrôleur et détail des segments ROM et RAM de la mémoire de microprogramme

d'une nouvelle instruction). Les registres `R0` à `R7`, `N`, `Z`, `P` peuvent être modifiés sans contrainte. En revanche, `MDR`, `MAR`, `IR` et `PC` ne doivent pas être modifiés et les accès mémoire sont interdits.

Question 1.7

Proposez un nouveau chemin de données autorisant la mise à jour du segment de RAM du microprogramme à partir de la mémoire principale du LC-2.

Question 1.8

Afin de microprogrammer l'instruction `LM`, on a besoin d'incrémenter le registre indiqué par *SR1* (l'adresse des microinstructions à copier), ainsi qu'un mécanisme pour compter le nombre de mots mémoire à copier (3 fois le nombre de microinstructions à copier, donc 3 fois la valeur du registre *SR2* plus 3). Pour cela, proposez les modifications les plus simples possibles à apporter au chemin de données du LC-2 ; représentez ces modifications sur un schéma. Indication : on pourra utiliser le signal N introduit aux questions précédentes.

Question 1.9

Donnez le microprogramme de l'instruction `LM`, (sans décrire le décodage de l'opcode de cette instruction), et en ne précisant que les signaux de contrôle dont le rôle est important à un cycle donné.

Question 1.10

Donnez le microprogramme de l'instruction `EM`, puis expliquez comment utiliser ce nouveau mécanisme pour effectuer le calcul de la multiplication non signée (à l'aide des extensions au chemin de données introduites pour l'implémentation de l'instruction `MUL`).

Question 1.11

Quelles sont les principales restrictions sur l'utilisation de l'instruction `EM` ?

Problème 2 - Cache Instruction (6 points)

On a vu dans le cours qu'un cache est une mémoire petite, mais plus rapide que la mémoire principale, placée entre le processeur et la mémoire principale. Toutes les requêtes du processeur à la mémoire sont en fait envoyées au cache. Les processeurs actuels incluent en général deux caches (au moins), l'un contient les données et l'autre les instructions ; dans cet exercice, on s'intéressera au cache d'instructions. On a également vu en cours qu'un cache comportait deux parties : un tableau contenant les informations (ici les instructions) et un tableau contenant les étiquettes (les adresses en mémoire des informations) ; ces deux tableaux forment le *banc* du cache ; les caches peuvent contenir un ou plusieurs bancs. On rappelle également que, selon sa taille, une ligne de cache peut avoir une contenance équivalente à plusieurs cases de la mémoire, i.e., on peut éventuellement y stocker plusieurs instructions. Enfin, on rappelle que les caches utilisent une méthode particulière de placement des instructions (ou données) : normalement, un certain nombre de bits de poids faible de l'adresse correspondent au numéro d'octet dans la ligne (ce nombre de bits dépend de la taille de la ligne), et plusieurs bits suivants correspondent au numéro de ligne dans le tableau (le nombre de bits dépend ici du nombre de lignes dans le tableau). Il est recommandé de se reporter à la partie du poly du cours expliquant l'implémentation d'un cache.

Dans un cache d'instructions pour le LC-2, les choses sont légèrement différentes : l'unité de la mémoire est 16 bits et non 8 bits (un octet). Donc, les adresses envoyées par le processeur au cache ne correspondent pas à des adresses d'octets mais à des adresses d'éléments de 16 bits. Aussi, le fonctionnement du cache et notamment l'utilisation des adresses est légèrement différente : les bits de poids faible ne correspondent pas au numéro d'octet dans la ligne, mais au numéro du groupement de 16 bits dans la ligne.

Le but de l'exercice est de faire fonctionner un cache d'instructions, et de comptabiliser le nombre de *défauts de cache*, i.e., le nombre de fois où une requête du processeur ne peut être satisfaite par le cache (l'instruction n'est pas dans le cache) et doit être envoyée à la mémoire. On considère d'abord un cache à un seul banc de 32 octets, avec des lignes de cache de 4 octets ; une requête servie par le cache requiert 1 cycle, et une requête engendrant un défaut de cache coûte 20 cycles.

Le contenu initial du cache est indiqué dans la Figure 6(a), et celui d'une partie de la mémoire est indiqué dans la Figure 6(b). On suppose que le PC contient initialement `x3002` (adresse hexadécimale) et on exécute 6 instructions.

Numéro ligne	Étiquette	Ligne de cache
0	x301	xD0001020
1	x301	x10201021
2	x300	xC8101021
3	x300	x10201020
4	x300	x10201020
5	x300	x10201020
6	x300	x10201020
7	x300	x10201020

Adresse	Contenu
x3000	x1020
x3001	x1020
x3002	xE000
x3003	x1023
x3004	xC810
x3005	x1021
x3006	x1020
x3007	x1020
x3008	x1020
x3009	x1020
x300A	x1020
x300B	x1020
x300C	x1020
x300D	x1020
x300E	x1020
x300F	x1020
x3010	xD000
x3011	x1020
x3012	x1020
x3013	x1021
x3014	xD000
x3015	x1020
x3016	x1020
x3017	x1020
x3018	x1020
x3019	x1020
x301A	x1020
x301B	x1020
x301C	x1020
x301D	x1020
x301E	x1020
x301F	x1020

FIG. 6 – (a) Contenu du cache (étiquettes et lignes de cache), (b) Contenu d'une partie de la mémoire.

Question 2.1

Expliquer pourquoi le banc d'étiquettes dans la Figure 6(a) ne contient que 12 bits (3 chiffres hexadécimaux).

Question 2.2

On exécute maintenant les 6 instructions. Indiquer le contenu du tableau d'étiquettes (les étiquettes uniquement, pas les lignes de cache) avant chaque défaut de cache (en utilisant une table comme dans la Figure 6(a)). Indiquer le contenu du registre `R0` en hexadécimal après l'exécution de la 6ème instruction.

Question 2.3

On suppose maintenant que le cache est d'associativité 2 (deux bancs de 4 lignes chacun) et que la politique de remplacement est LRU (*Least Recently Used*) ; on considère cette fois-ci que le cache est entièrement vide initialement (tous les bits sont à 0 dans le banc d'étiquettes comme dans les lignes de cache). Quelle est maintenant la taille d'une étiquette ? Expliquer pourquoi la taille a changé. Indiquer le nombre de défauts de cache.

Question 2.4

Question facultative et libre (4 points supplémentaires). On considère à nouveau un cache qui ne contient qu'un seul banc. En utilisant quelques portes logiques supplémentaires, mais sans ajouter de cellules mémoires, imaginer une fonction de placement dans le cache différente qui permet de réduire le nombre de défauts de cache dans l'exemple précédent, et d'une façon plus générale, qui les réduirait pour d'autres programmes et pour les accès au cache de données. Expliquer et justifier votre approche.