Examen d'Architecture des Ordinateurs
Majeure 1 - Polytechnique

2003–2004

- L'examen dure 3 heures.
- Le sujet comporte 11 pages dont 2 pages de rappels sur le LC-2.
- 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 30.
- L'examen contient deux problèmes distincts que vous devez rendre sur **2 copies séparées**.
- Il est impératif de commenter les programmes en assembleur ; normalement, presque chaque instruction doit être suivie d'un commentaire permettant de comprendre son rôle.

Problème 1 - Contrôle câblé du LC-2 (20 points)

Le but de ce problème est d'étudier le contrôle câblé du LC-2 : d'abord pour l'implémentation du LC-2 vue en cours, puis une implémentation simplifiée. Dans un premier temps, on va donc supprimer le contrôle microprogrammé et le remplacer par un contrôle câblé, où l'on déterminera l'expression logique de chaque signal de contrôle. Dans un second temps, on fera plusieurs simplifications dans la structure et le jeu d'instructions du LC-2.

On part d'une version du LC-2 presque identique, mais pas tout à fait, à celle vue en cours et en TD ; notamment il n'y a plus d'instruction TRAP, les signaux de contrôle *MARMX, DRMX, SR1MX* ont été légèrement modifiés, on a introduit un signal *MEMX* à la place du signal *MIO.EN* (la partie entrées/sorties ayant été enlevée), et MAR, MDR sont des **registres** et non des latchs. Il est donc recommandé de bien regarder l'Annexe et la Figure 2 lors de vos réponses aux questions ci-dessous.

On suppose dans l'ensemble des questions que le temps de cycle est très grand devant la traversée d'un circuit combinatoire et devant le temps d'accès à la mémoire ; en conséquence, pendant un cycle d'horloge, on peut traverser plusieurs circuits combinatoires et/ou la mémoire.

Question 1.1

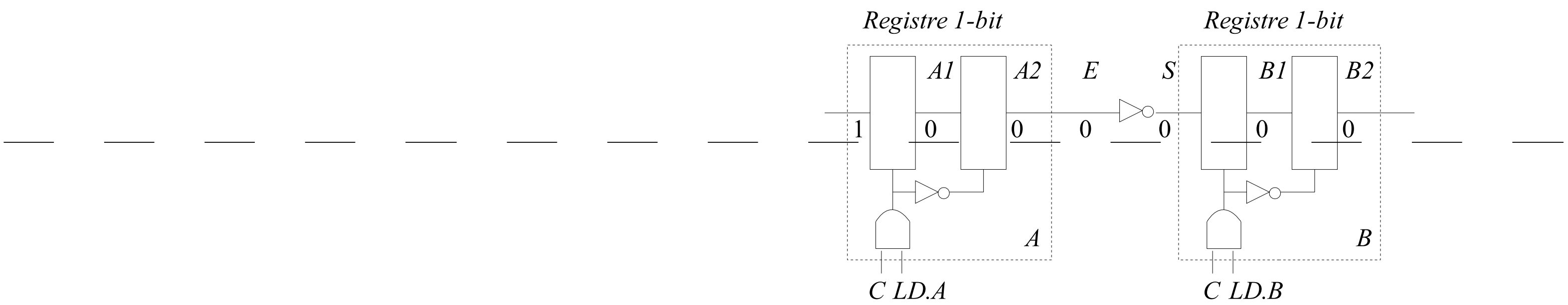

Fig. 1 – (a) 3 cycles du signal d'horloge ; (b) Structure des registres.

Dans la Figure 1(a), on représente 3 cycles de l'horloge du LC-2 (front montant d'abord), et dans la Figure 1(b), on représente la structure de deux registres 1-bit (chacun composé de 2 latchs en opposition de phase) séparant un petit circuit combinatoire (une simple porte NON ici) ; le signal $LD.x$ est un signal d'écriture, le signal C est le signal d'horloge. Les questions ci-dessous sont des rappels de cours et ont pour but de bien préciser le fonctionnement des registres.

En considérant que la configuration initiale à $t = 0$ est celle de la Figure 1(b), indiquer à quelle date les évènements suivants se produisent (approximativement à quel endroit sur le schéma du signal d'horloge de la Figure 1(a) ; vous devez reproduire le signal d'horloge sur votre copie) :

1. le signal E est à 1 ?
2. le signal S est à 1 ?
3. le signal B_1 est à 1 ?
4. le signal B_2 est à 1 ?

Réponse

1. *le signal* E *passe à 1 pendant le deuxième demi-cycle.*
2. *le signal* S *passe à 1 pendant le premier demi-cycle.*
3. *le signal* B_1 *passe à 1 pendant le premier demi-cycle.*
4. *le signal* B_2 *passe à 1 pendant le deuxième demi-cycle.*

Question 1.2

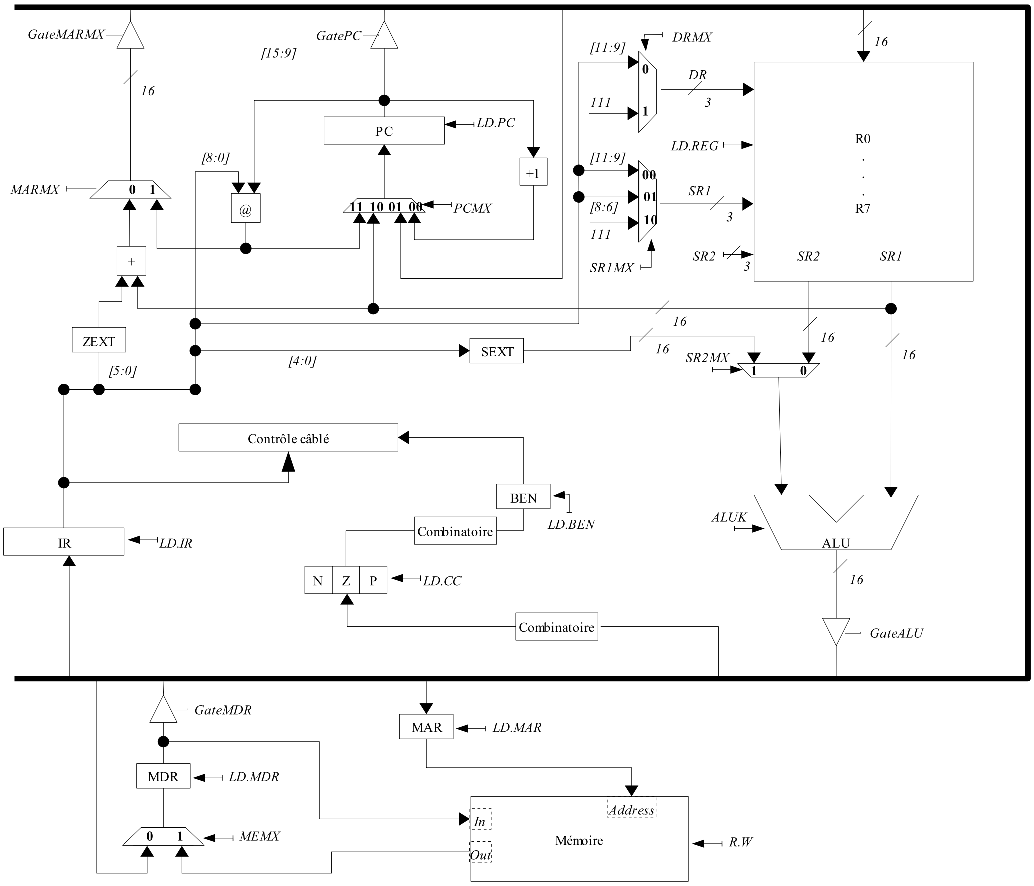

FIG. 2 – LC-2.

Pour chaque instruction du LC-2, indiquer le nombre de cycles nécessaires pour l'exécution complète de l'instruction et quelles actions sont effectuées pendant chaque cycle. Il est important de prendre en compte les hypothèses énoncées au début de l'exercice sur le délai des circuits combinatoires et de la mémoire. On présentera les résultats **uniquement** selon le format indiqué ci-dessous, une table avec en colonnes les cycles et en lignes les instructions.

Notations : les différents registres seront indiqués par leur nom : $PC, IR, MAR, MDR, NZP, BEN$. Les registres sources d'un calcul seront notés $SR1$ ou $SR2$, et le registre destination sera noté DR. S'il est nécessaire de mentionner un registre spécifique i du banc de registres, on le nommera R_i.

La mémoire sera notée MEM, l'ALU sera notée ALU. Les bits x à y d'un registre R seront notés $R[y:x]$. Tous les offsets, index et immédiats seront nommés en utilisant les bits du registre instruction IR, ainsi l'offset de 6 bits sera noté $IR[5:0]$. Un transfert d'un composant A vers un composant B sera noté $A \rightarrow B$.

Instruction	Cycle 1	Cycle 2	Cycle 3	...
ADD				
AND				
ADD (immediat)				
AND (immediat)				
NOT				
BR				
JMP				
JSR				
JMPR				
JSRR				
RET				
LD				
LDR				
ST				
STR				
LEA				

Réponse

Instruction	*Cycle 1*	*Cycle 2*	*Cycle 3*	*Cycle 4*	*Cycle 5*	*Cycle 6*
ADD	$PC \rightarrow MAR$	$MAR \rightarrow MEM \rightarrow MDR$	$MDR \rightarrow IR$	$SR1, SR2 \rightarrow ALU \rightarrow DR, NZP$		
ADD	$PC+1 \rightarrow PC$					
AND	*id*	*id*	*id*	*id*		
ADD (immediat)	*id*	*id*	*id*	$SR1, IR[4:0] \rightarrow ALU \rightarrow DR, NZP$		
AND (immediat)	*id*	*id*	*id*	*id*		
NOT	*id*	*id*	*id*	$SR1 \rightarrow ALU \rightarrow DR, NZP$		
BR	*id*	*id*	*id*	$NZP \rightarrow BEN$	$PC[16:9]@IR[8:0] \rightarrow PC$ *si* $BEN = 1$	
JMP	*id*	*id*	*id*	$PC[16:9]@IR[8:0] \rightarrow PC$		
JSR	*id*	*id*	*id*	$PC[16:9]@IR[8:0] \rightarrow PC,\ PC \rightarrow R_7$		
JMPR	*id*	*id*	*id*	$SR1 + IR[5:0] \rightarrow PC$		
JSRR				$PC \rightarrow R_7$	$SR1 + IR[5:0] \rightarrow PC$	
RET				$R7 \rightarrow PC$		
LD				$PC[16:9]@IR[8:0] \rightarrow MAR$	$MAR \rightarrow MEM \rightarrow MDR$	$MDR \rightarrow DR, NZP$
LDR				$SR1 + IR[5:0] \rightarrow MAR$	$MAR \rightarrow MEM \rightarrow MDR$	$MDR \rightarrow DR, NZP$
ST				$PC[16:9]@IR[8:0] \rightarrow MAR$	$SR1 \rightarrow MDR$	$MAR, MDR \rightarrow MEM$
STR				$SR1 + IR[5:0] \rightarrow MAR$	$SR1 \rightarrow MDR$	$MAR, MDR \rightarrow MEM$
LEA				$PC[16:9]@IR[8:0] \rightarrow DR$		

Question 1.3

On définit un certain nombre de variables booléennes ci-dessous. Donner les expressions logiques de ces variables en fonction des bits du registre instruction IR :

1. BR vaut 1 si l'instruction est BR.
2. RET vaut 1 si l'instruction est RET.
3. ADD vaut 1 si l'instruction est une addition.
4. NOT vaut 1 si l'instruction est le NON logique.
5. IMM vaut 1 si l'instruction effectue une opération arithmétique ou logique et que l'un des deux opérandes est un immédiat.
6. $BCHT$ vaut 1 si l'instruction est un branchement (i.e., peut modifier le PC).
7. MEM vaut 1 si l'instruction effectue un accès à la mémoire.
8. ALU vaut 1 si l'instruction effectue une opération arithmétique ou logique.
9. $CALL$ vaut 1 si l'instruction effectue un appel de procédure.
10. $INDEX$ vaut 1 si l'instruction utilise l'adressage indexé.
11. $DIRECT$ vaut 1 si l'instruction utilise l'adressage direct.
12. $STORE$ vaut 1 si l'instruction effectue une écriture en mémoire.
13. $INCOND$ vaut 1 si l'instruction effectue un branchement inconditionnel.

Montrer que l'on peut caractériser l'opcode de toute instruction, ainsi que les bits IR_{11} et IR_5, comme une expression logique des variables ci-dessus. En déduire que l'on peut substituer les variables ci-dessus aux bits IR_i dans la conception du circuit de contrôle.

Réponse

1. $BR = IR'_{15}IR'_{14}IR'_{13}IR'_{12}$
2. $RET = IR_{15}IR_{14}IR'_{13}IR_{12}$
3. $ADD = IR'_{15}IR'_{14}IR'_{13}IR_{12}$
4. $NOT = IR_{15}IR'_{14}IR'_{13}IR_{12}$
5. $IMM = IR'_{15}IR'_{13}IR_{12}IR_5$
6. $BCHT = IR'_{13}IR'_{12} + IR_{15}IR_{14}IR'_{13}IR_{12}$
7. $MEM = IR'_{15}IR_{13}$
8. $ALU = IR'_{13}IR_{12}(IR'_{15} + IR_{15}IR'_{14})$
9. $CALL = IR_{14}IR'_{13}IR'_{12}IR_{11}$
10. $INDEX = IR_{15}IR_{14}IR'_{13}IR'_{12} + IR'_{15}IR_{14}IR_{13}$
11. $DIRECT = IR'_{15}IR'_{13}IR'_{12} + IR_{15}IR_{14}IR_{13}IR'_{12} + IR'_{15}IR'_{14}IR_{13}$
12. $STORE = IR_{13}IR_{12}$
13. $INCOND = IR_{14}IR'_{13}(IR'_{15}IR'_{12} + IR_{15}) = IR_{14}IR'_{13}(IR'_{12} + IR_{15})$

On constate que l'on peut écrire les expressions inverses pour les bits IR_i de l'opcode, ainsi que les bits IR_{11} et IR_5. On peut donc écrire l'ensemble du circuit de contrôle en utilisant ces expressions logiques, sans utiliser IR. Bien entendu, pour les numéros de registres et pour les données (bases, offsets, immédiats), on doit toujours utiliser IR. Les bits nzp étant maintenant directement récupérés lors du transfert d'une instruction vers IR, il n'est plus nécessaire de les lire depuis IR.

Question 1.4

Pour le contrôle câblé, on a besoin de savoir à tout moment dans quel cycle de l'exécution de l'instruction on se trouve. Pour cela, on introduit de nouvelles variables booléennes C_i ($i \geq 1$), C_i valant 1 uniquement pendant que l'on exécute le i^{eme} cycle de l'instruction courante. Donner le schéma d'un circuit permettant de fournir les signaux C_i (on ne cherchera pas à minimiser la taille du circuit). On dispose de bascules toutes équipées d'un signal RAZ de remise à zéro, et d'un signal $START$ qui vaut 1 pendant un cycle lors du démarrage du processeur (on ne se préoccupera pas de l'arrêt de ce circuit). On supposera que la condition du branchement BR est calculée pendant le 4ème cycle de l'exécution de l'instruction BR.

Réponse

Il s'agit d'un circuit de contrôle que l'on peut définir à partir d'un organigramme comme pour les circuits vus en cours (notamment le multiplieur). On transforme ensuite cet organigramme en circuit séquentiel en utilisant des bascules et quelques portes combinatoires. Les tests dans l'organigramme sont liés à l'opcode et au résultat du branchement conditionnel (BEN). La plus longue instruction ayant 6 étapes, on aura donc une boucle de 6 cycles.

Question 1.5

Déterminer les expressions logiques de l'ensemble des signaux de contrôle du LC-2 de la Figure 2. On utilisera **uniquement** les variables booléennes définies à la Question 1.3, ainsi que les bits C_i définis à la Question 1.4 et le signal BEN, mais sans utiliser les bits du registre IR. Donner ces expressions selon le format de la table ci-dessous. Bien que l'on ne demande pas de minimiser les expressions logiques, on fera attention à trouver des expressions raisonnablement simples. Chaque expression devra être accompagnée d'un commentaire l'expliquant/la justifiant.

L'ensemble de ces expressions et le circuit de la Question 1.4 forment le circuit de contrôle câblé du LC-2.

Signal de contrôle	Expression logique
$DRMX$	
$MARMX$	
$MEMX$	
$PCMX_1$	
$PCMX_0$	
$SR1MX_1$	
$SR1MX_0$	
$SR2MX$	
$ALUK_1$	
$ALUK_0$	
$R.W$	
$GateALU$	
$GateMARMX$	
$GateMDR$	
$GatePC$	
$LD.BEN$	
$LD.CC$	
$LD.IR$	
$LD.MAR$	
$LD.MDR$	
$LD.PC$	
$LD.REG$	

Réponse

Signal de contrôle	*Expression logique...*
$DRMX$	$CALL$
$MARMX$	$DIRECT$
$MEMX$	$C_2 + STORE'$
$PCMX_1$	$C'_1.(RET + BCHT.DIRECT)$
$PCMX_0$	$C'_1.(DIRECT + BCHT.INDEX)$
$SR1MX_1$	RET
$SR1MX_0$	$ALU + INDEX$
$SR2MX$	IMM
$ALUK_1$	$NOT + STORE$
$ALUK_0$	$ALU.NOT'.ADD' + STORE$
$R.W$	$C_6.STORE$
$GateALU$	$C_4.ALU + C_5.STORE$
$GateMARMX$	$C_4.(INDEX.(MEM + BCHT.CALL') + DIRECT.BCHT') + C_5.CALL.INDEX$
$GateMDR$	$C_3 + C_5.MEM.STORE'$
$GatePC$	$C_1 + CALL.C_4$
$LD.BEN$	$C_4.BR$
$LD.CC$	$C_4.ALU + C_6.MEM.STORE'$
$LD.IR$	C_3
$LD.MAR$	$C_1 + C_4.MEM$
$LD.MDR$	$C_2 + C_5.MEM$
$LD.PC$	$C_1 + C_4.INCOND.(CALL.INDEX)' + C_5.(BR.BEN + CALL.INDEX)$
$LD.REG$	$C_4.(ALU + CALL + DIRECT.MEM'.BCHT') + C_6.MEM.STORE'$

Question 1.6

Normalement, l'écriture dans le registre BEN a lieu au 4ème cycle de l'exécution d'une instruction, une fois que l'on sait qu'il s'agit d'une instruction BR, et que l'on connaît la nature de l'instruction et les éventuels bits des conditions à tester nzp . En conséquence, on ne peut déterminer qu'au cycle suivant si le branchement peut être pris et donc modifier le PC. Comment modifier légèrement l'architecture pour gagner un cycle dans l'exécution de BR (faire un schéma) ?

Indiquer les modifications du circuit de contrôle engendrées par ce changement, notamment les nouvelles expressions des signaux de contrôle de la Question 1.5 affectés par ce changement.

Par la suite, on supposera que cette modification a été effectuée (si vous ne parvenez pas à répondre à la question, continuez à considérer que BEN est calculé au 4ème cycle).

Réponse

On peut calculer BEN spéculativement avant de savoir si c'est un BR. Cependant, il faut disposer des 3 bits nzp. Or ceux-ci reviennent de la mémoire au 3ème cycle, et ils sont stockés à ce moment-là dans IR. On peut donc en même temps les envoyer au circuit combinatoire devant BEN. Soit il s'agit bien d'un BR, et on aurait gagné un cycle puisque BEN serait disponible dès le début du 4ème cycle. Soit il s'agit d'une autre instruction et BEN n'est pas utilisé. BEN est donc écrit au 3ème cycle, quelle que soit l'instruction.

Cela modifie également la boucle séquentielle de la Question 1.4 : BR s'exécute maintenant en 4 cycles.

Signal de contrôle	*Expression logique...*
$LD.BEN$	C_3
$LD.PC$	$C_1 + C_4.(BCHT.CALL'.(BR' + BEN) + CALL.INDEX') + C_5.CALL.INDEX$

Question 1.7

Normalement, l'incrémentation du PC ($PC + 1 \rightarrow PC$) a lieu au premier cycle de l'exécution d'une instruction, en même temps que l'envoi du PC à la mémoire pour charger l'instruction.

1. Expliquer comment le programme ci-dessous va se comporter.

```
        .ORIG x31FE
LABEL   AND     R0, R0, 0
        JMP     LABEL
```

2. Au plus tôt, à quel cycle de l'exécution d'une instruction doit-on placer l'incrémentation du PC pour que ce problème ne se produise avec aucune instruction ? Y a-t-il des instructions affectées par ce changement ? Indiquer les modifications du circuit de contrôle engendrées par ce changement. On prendra en compte cette modification dans les questions suivantes (si vous ne parvenez pas à répondre à cette question, continuez à considérer que PC+1 est effectué au premier cycle).
3. En supposant que cette modification a été effectuée, pourquoi est-ce que l'exécution du code ci-dessous sera incorrecte malgré tout ? Quelle solution simple le compilateur pourrait apporter dans ce cas ? Cette solution peut-elle être appliquée au problème précédent ? Dans quels cas, cette modification serait insuffisante ?

```
        .ORIG x31FE
        JMP     LABEL
        BR      0
LABEL   AND     R0, R0, 0
```

Réponse

1. *Le PC du JMP est x31FF, donc les 9 bits de poids faible sont 111111111, et les 7 bits de poids fort sont 0011000 ; l'adresse du label est x31FE soit 0011000 111111110. Comme l'incrémentation du PC a lieu avant même que JMP ne soit chargé, au premier cycle, le PC vaudra 0011001 000000000 lorsqu'on va calculer l'adresse de destination, et l'on va donc faire la concaténation de 0011001 (7 bits de poids fort du PC) avec 111111110 (9 bits de poids faible de LABEL), et l'adresse de destination sera 0011001 111111110 au lieu de 001100 111111110.*
2. *Pour les instructions de branchement inconditionnel et les instructions mémoire, le PC est utilisé au cycle 4 ; pour BR, on a vu à la question précédente que l'on pouvait effectuer le calcul de BEN tôt et donc calculer l'adresse de destination au cycle 4 également. On peut donc effectuer l'incrémentation au cycle 4. Cependant, les instructions JSR et JSRR doivent stocker dans R_7 PC+1 et non PC (sinon RET ramènera à l'instruction JSR/JSRR et non à l'instruction suivante). Au cycle 4 on a donc $PC + 1 \rightarrow PC$, au cycle 5 on a $PC \rightarrow R_7$ et au cycle 6, pour JSRR, on effectue le transfert de l'adresse vers le PC par le bus. Cela modifie également la boucle séquentielle de la Question 1.4.*

Signal de contrôle	*Expression logique...*
$PCMX_1$	$DIRECT.(BR' + BEN) + RET$
$PCMX_0$	$DIRECT.(BR' + BEN) + BCHT.INDEX$
$LD.PC$	$C_4.(BCHT' + BR.BEN) + C_5.CALL.INDEX' + C_6.CALL.INDEX$
$LD.REG$	$C_4.(ALU + DIRECT.MEM'.BCHT') + C_5.CALL + C_6.MEM.STORE'$

3. *Cette fois-ci, le problème est que le JMP doit franchir une "page", c'est à dire passer dans une zone dont les 7 bits de poids fort d'adresse sont différents, ce qui est impossible avec ce mode d'adressage. Les 7 bits de poids fort du PC sont 0011000 et les 9 bits de poids faible de LABEL sont 000000000, donc l'adresse de destination sera 0011000 000000000 au lieu de 001101 000000000. Le compilateur peut monter ou descendre le programme dans la mémoire pour éviter de franchir une page, par exemple x3200 dans ce cas-là. Cette solution s'applique effectivement au cas précédent : il suffirait de remonter le programme en x31FD pour éviter le problème de la première question. Cette solution est inadéquate si la distance entre le JMP et l'adresse de destination est supérieure à une page ; une page étant déterminée par 9 bits d'adresse, il s'agit de $2^9 = 512$ instructions, soit 1024 octets.*

Question 1.8

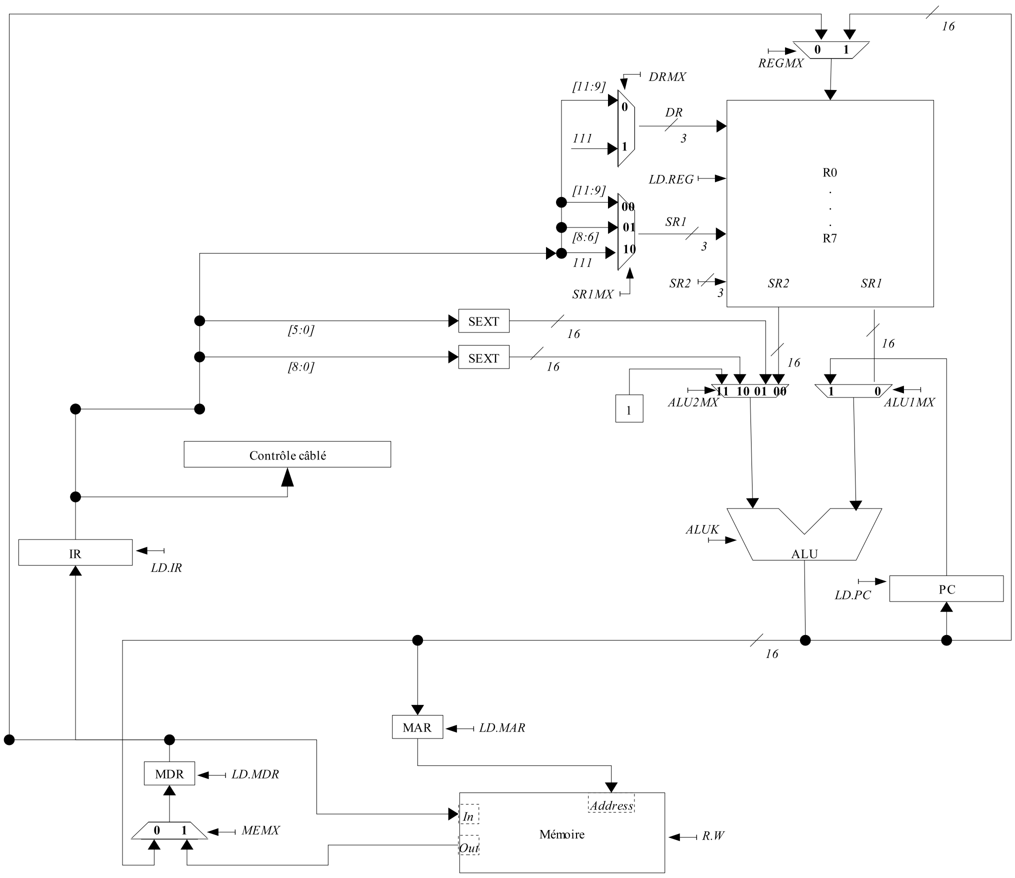

FIG. 3 – LC-2 simplifié.

On veut maintenant simplifier plusieurs autres aspects de l'architecture du LC-2. Les modifications à apporter sont les suivantes :

- On enlève les deux additionneurs spécialisés : celui utilisé pour PC+1 et celui utilisé pour le calcul d'adresse. Tous les calculs ont maintenant lieu dans l'ALU.
- Les offsets, bases et immédiats sont maintenant tous signés (en complément à 2).
- On remplace le mode d'adressage direct ($PC[15:9]@IR[8:0]$) par un mode d'adressage relatif ($PC + SEXT(IR[8:0])$).
- On supprime les registres N, Z, P, et l'instruction BR peut seulement tester si le contenu d'un registre est strictement négatif. Elle admet donc deux opérandes : un registre SR2 qui contient la valeur à tester, et un registre SR1 qui contient l'adresse de destination ; aucun offset n'est utilisé pour calculer cette adresse (il s'agit donc d'un mode spécial d'adressage). Le nouveau format de l'instruction BR est indiqué ci-dessous.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0		SR1		0	0	0		SR2	

– On supprime l'instruction RET et les instructions JSR,JSRR. On utilise les opcodes de JSR et JSRR pour respectivement le ADD immédiat et le AND immédiat. L'immédiat des instructions ADD et AND est maintenant de 6 bits. En conséquence, il ne reste plus que deux types d'immédiats/bases/offsets : [8 :0] et [5 :0], tous les deux étant maintenant signés.
  Une autre conséquence est qu'il n'y a plus que 2 entrées pour $SR1MX$, donc le signal de contrôle n'a plus qu'un seul bit : 0 pour $IR[11:9]$, 1 pour $IR[8:6]$.
– On supprime le bus. Cette suppression est compensée par un élargissement de la taille du multiplexeur devant la deuxième entrée de l'ALU (à gauche) : il comporte 4 entrées et donc 2 bits de contrôle notés $ALU2MX_1$ et $ALU2MX_0$ (en remplacement de $SR2MX$) ; les conventions sont les suivantes (de droite à gauche) : 00 pour $SR2$, 01 pour $IR[5:0]$, 10 pour $IR[8:0]$, 11 pour la valeur 1 (0000000000000001 en binaire). On doit également ajouter un multiplexeur devant l'autre port de l'ALU, contrôlé par un signal $ALU1MX$ (de droite à gauche) : 0 pour $SR1$, 1 pour PC. Enfin, on doit ajouter un multiplexeur devant le port d'écriture du banc de registres contrôlé par un signal $REGMX$ (de droite à gauche) : 0 pour l'ALU, 1 pour la mémoire.
  Le multiplexeur devant le PC devient inutile, toutes les informations envoyées au PC proviendront de l'ALU. L'ALU est également directement connectée avec les registres MAR et MDR.
  Toutes ces modifications sont indiquées Figure 3.

1. Pourquoi la suppression des instructions RET et JSR/JSRR permet-elle d'utiliser un immédiat de 6 bits pour les opérations ALU ?
2. Par quelle instruction peut-on remplacer RET ?
3. Comment réécrire le programme ci-dessous sans JSR/JSRR ni RET pour avoir un comportement équivalent ?

```
        .ORIG x3000
        JSR    LABEL
        ...
LABEL   AND    R0, R0, 0
        RET
        ...
```

4. Comment est maintenant calculée la valeur stockée dans le registre BEN ? A-t-on encore besoin du registre BEN pour conserver le signal BEN ? Peut-on conserver la modification de la Question 1.6 ?
5. On veut conserver la modification de la Question 1.7 : on utilise d'abord l'ALU pour effectuer l'opération de l'instruction, et dès que possible, on l'utilise pour effectuer la modification du PC. Indiquer pour chaque instruction à quel cycle on effectue $PC + 1$ ou la modification du PC.

Réponse

1. *On n'a plus besoin du bit IR_5 pour différencier modes immédiat et registre. On peut donc l'utiliser dans l'offset maintenant étendu à 6 bits. Cependant, les instructions d'appel de procédure (JSR et $JSRR$) sont identifiés par le bit IR_{11} qui n'est pas un bit d'opcode. On ne peut donc simplement utiliser les opcodes de JSR et $JSRR$ puisque ce sont aussi ceux de JMP et $JMPR$. Il faut donc maintenant remplacer JSR par $JMPR$, et identifier $ADDi/ANDi$ comme $JMPR/JSRR$ auparavant (même opcode plus le bit IR_{11}).*
2. *On peut remplacer RET par $JMPRR7, 0$.*
3.

```
        .ORIG x3000
        LEA    R7, 2
        JMP    LABEL
        ...
LABEL   AND    R0, R0, 0
        JMP    R7, 2
```

...

4. *BEN est simplement égal au bit de poids fort de SR2 : $SR2_{15}$. On peut continuer à calculer BEN au cycle 3, puisque $SR2$ est câblé sur les bits 2 : 0 du registre IR, i.e., toujours les mêmes, quelle que soit l'instruction. Donc on peut connecter $MDR[2:0]$ à $SR2$, sachant que SR2 n'est utilisé que pour les instructions ALU et BR : donc MDR ne sert qu'au chargement de l'instruction et n'est plus modifié ensuite. Ainsi, on obtient $SR2$, le registre à tester, dès le cycle 3, et $SR2_{15}$ sert de bit BEN.*
5. *On peut toujours retarder $PC+1$, mais on ne peut plus l'effectuer en même temps qu'une opération qui utilise l'ALU. On le reporte donc en fin d'instruction.*

Question 1.9

Reprendre la Question 1.2 pour l'architecture simplifiée du LC-2.

Réponse

Instruction	*Cycle 1*	*Cycle 2*	*Cycle 3*	*Cycle 4*	*Cycle 5*	*Cycle 6*	*Cycle 7*
ADD	$PC \to MAR$	$MAR \to MEM \to MDR$	$MDR \to IR$	$SR1, SR2 \to ALU \to DR$	$PC+1 \to PC$		
AND	*id*	*id*	*id*	*id*			
ADD (immediat)	*id*	*id*	*id*	$SR1, IR[5:0] \to ALU \to DR$	$PC+1 \to PC$		
AND (immediat)	*id*	*id*	*id*	*id*			
NOT	*id*	*id*	*id*	$SR1 \to ALU \to DR$	$PC+1 \to PC$		
BR	*id*	*id*	*id*	$SR1 \to PC$			
				si $BEN = 1$			
				sinon $PC+1 \to PC$			
JMP	*id*	*id*	*id*	$PC + IR[8:0] \to PC$			
JMPR	*id*	*id*	*id*	$SR1 + IR[8:0] \to PC$			
LD				$PC + IR[8:0] \to MAR$	$MAR \to MEM \to MDR$	$MDR \to DR$	
LD					$PC+1 \to PC$		
LDR				$SR1 + IR[5:0] \to MAR$	$MAR \to MEM \to MDR$	$MDR \to DR$	$PC+1 \to PC$
LDR					$PC+1 \to PC$		
ST				$PC + IR[8:0] \to MAR$	$SR1 \to MDR$	$MAR, MDR \to MEM$	
ST						$PC+1 \to PC$	
ST				$SR1 + IR[5:0] \to MAR$	$SR1 \to MDR$	$MAR, MDR \to MEM$	
STR						$PC+1 \to PC$	
LEA				$PC + IR[8:0] \to DR$	$PC+1 \to PC$		

Question 1.10

La boucle de la Question 1.4 doit-elle être modifiée ? Reprendre la Question 1.5 pour l'architecture simplifiée du LC-2. Le signal $DIRECT$ n'est plus utilisé puisqu'il n'y a plus d'adressage direct ; on utilisera un signal $RELATIF$ qui vaut 1 si l'instruction utilise l'adressage relatif. Sans les redéfinir, on prêtera attention au fait que l'expression de certains signaux de la Question 1.3, comme $INCOND$, $BCHT$, $INDEX$,... a changé.

Réponse

Il n'y a plus de JSR/JSRR, mais $BR, JMP, JMPR$ s'exécutent en 4 cycles, ST/STR en 6 cycles, et toutes les autres instructions en 5 cycles. Il faut ajuster en conséquence le passage du jeton dans la boucle.

$DRMX$ est devenu inutile (pas de RET).

$REGMX$ vaut 1 pour LD/LDR, 0 sinon.

Le passage à travers à l'ALU est souvent utilisé : par les opérations ALU, LD/LDR et LEA au cycle 5 pour PC+1, par ST/STR au cycle 6 pour PC+1, par ST/STR aux cycles 4 et 5 pour passer l'adresse et la donnée, et par BR au cycle 4 pour le branchement.

La valeur 1 est utilisée dans SR2MX par toutes les instructions sauf les branchements et ST/STR au cycle 5 pour PC+1, et par ST/STR au cycle 6 pour PC+1. L'offset $IR[8:0]$ est maintenant utilisé par les instructions ALU avec adressage immédiat, et l'adressage relatif (BR n'en fait plus partie). L'offset $[5:0]$ est utilisé par l'adresse indexé seulement (LDR/STR).

Annexe

Les informations ci-dessous ont été vues en TD. Attention, quelques modifications ont été apportées, voir le début de l'énoncé. Les modifications sont indiquées en gras ci-dessous.

- On rappelle le format des principales instructions du LC-2 dans la Figure 4.
  où SEXT signifie *Signed Extension* (extension signée à 16 bits en complément à 2), et ZEXT signifie *Zero Extension* (extension non signée à 16 bits) , @ est l'opérateur de concaténation (de deux champs de bits), IR[x :y] dénote les bits x à y du registre instruction (IR). On peut noter que l'instruction STR utilise deux fois SR1 : elle utilise effectivement deux fois le port SR1 du banc de registres, d'abord les bits IR[8 :6] indiquent le registre utilisé pour calculer l'adresse qui est envoyée à MAR, puis les bits IR[11 :9] indique le registre contenant la valeur envoyée à MDR.
- On rappelle le rôle des signaux de contrôle ci-dessous.

Instruction	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`ADD` DR, SR1, SR2 $DR \leftarrow SR1 + SR2$	0	0	0	1	DR			SR1			0	0	0	SR2		
`ADD` DR, SR1, imm5 $DR \leftarrow SR1 + SEXT(imm5)$	0	0	0	1	DR			SR1			1	imm5 : immédiat 5-bits signé				
`AND` DR, SR1, SR2 $DR \leftarrow ET(SR1, SR2)$	0	1	0	1	DR			SR1			0	0	0	SR2		
`AND` DR, SR1, imm5 $DR \leftarrow ET(SR1, SEXT(imm5))$	0	1	0	1	DR			SR1			1	imm5 : immédiat 5-bits signé				
`NOT` DR, SR1 $DR \leftarrow NON(SR1)$	1	0	0	1	DR			SR1			1	1	1	1	1	1
`BR`nzp label $Si \quad n.N + z.Z + p.P = 1$ $addr = PC[15:9]@ZEXT(offset9)$ $PC \leftarrow addr$	0	0	0	0	n	z	p	offset 9-bits non signé								
`JMP` label $addr = PC[15:9]@ZEXT(offset9)$ $PC \leftarrow addr$	0	1	0	0	0	0	0	offset 9-bits non signé								
`JSR` label $R7 \leftarrow PC$ $addr = PC[15:9]@ZEXT(offset9)$ $PC \leftarrow addr$	0	1	0	0	1	0	0	offset 9-bits non signé								
`JMPR` label $addr = Rs1 + ZEXT(index6)$ $PC \leftarrow addr$	1	1	0	0	0	0	0	SR1			index 6-bits non signé					
`JSRR` label $R7 \leftarrow PC$ $addr = Rs1 + ZEXT(index6)$ $PC \leftarrow addr$	1	1	0	0	1	0	0	SR1			index 6-bits non signé					
`LEA` DR, label $DR \leftarrow PC[15:9]@ZEXT(offset9)$	1	1	1	0	DR			offset 9-bits non signé								
`LD` DR, label $addr = PC[15:9]@ZEXT(offset9)$ $DR \leftarrow Mem(addr)$	0	0	1	0	DR			offset 9-bits non signé								
`LDR` DR, SR1, label $addr = SR1 + ZEXT(index6)$ $DR \leftarrow Mem(addr)$	0	1	1	0	DR			SR1			index 6-bits non signé					
`ST` SR1, label $addr = PC[15:9]@ZEXT(offset9)$ $SR1 \rightarrow Mem(addr)$	0	0	1	1	SR1			offset 9-bits non signé								
`STR` SR1, SR1, label $addr = SR1 + ZEXT(index6)$ $SR1 \rightarrow Mem(addr)$	0	1	1	1	SR1			SR1			index 6-bits non signé					
`RET` $PC \leftarrow R7$	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

FIG. 4 – Format des instructions du LC-2

- LD.MAR, LD.MDR, LD.IR, LD.REG, LD.CC, LD.BEN et LD.PC commandent l'écriture dans les divers registres et latchs du LC-2 (1 si écriture, 0 sinon) ;
- GatePC, GateMDR, GateALU et GateMARMX commandent l'écriture sur le bus ;
- R.W : 0 pour une lecture et 1 pour une écriture en mémoire ;
- **MEMX :** 0 pour la donnée provenant du processeur, 1 pour celle provenant de la mémoire ;
- ALUK (2 bits) : 00 pour ADD, 01 pour AND, 10 pour NOT, 11 pour que l'opérande de droite (SR1) traverse l'ALU sans calcul ;
- PCMX (2 bits) : 00 pour sélectionner PC+1, 01 pour le bus, 10 pour la sortie du banc de registres, 11 pour la concaténation PC[15 :9]@IR[8 :0] ;

- MARMX (**1 bit**) : 0 pour l 'addition d'un registre et IR[5 :0], 1 pour la concaténation PC[15 :9]@IR[8 :0] ;
- SR1MX (2 bits) : multiplexeur alimentant le signal SR1 (numéro registre source 1) : 00 pour sélectionner IR[11 :9], 01 pour IR[8 :6], 10 pour la valeur 111 ;
- DRMX (**1 bit**) : multiplexeur alimentant le signal DR (numéro registre destination) : 0 pour sélectionner IR[11 :9], 1 pour sélectionner la valeur 111 ;
- SR2MX (1 bit) : multiplexeur alimentant le port de gauche de l'ALU : 0 pour SR2 (registre), 1 pour immédiat.