

immédiat. Avant de pouvoir être utilisés par le processeur, ces immédiats doivent subir une transformation. Pourquoi ? Pour chaque type d'immédiat, indiquer la transformation qu'il doit subir, et retrouver le circuit correspondant sur le schéma du processeur.

Réponse

Cette question et les trois suivantes sont corrigées ensemble, à la fin de l'exercice.

Question 4.1.2

A partir du format des instructions vu en cours, énumérer les différentes façons de calculer une adresse pour une instruction de chargement. Retrouver les différents composants du circuit du processeur qui interviennent dans le calcul des adresses des instructions de chargement.

Question 4.1.3

A partir du format des instructions vu en cours, énumérer les différentes façons de calculer l'adresse de destination d'une instruction de branchement. Retrouver les différents composants du circuit du processeur qui interviennent dans le calcul de l'adresse d'une instruction.

Question 4.1.4

On suppose que $PC = 0$ et que l'adresse 0 contient le code de l'instruction ADD R1, R2, R3. Retrouver toutes les étapes nécessaires à l'exécution de cette instruction, en commençant par son chargement. Spécifier à chaque fois la valeur des différents signaux de contrôle. Indiquer à quel cycle chaque signal doit être activé, en supposant que l'on commence l'exécution à $t = 0$. Ensuite, décrire le contrôle de l'exécution de l'instruction sous la forme d'un organigramme.

Réponse

Code de l'instruction ADD R1, R2, R3 : 0001001010000011, c'est à dire x1283.

La figure 14 décrit tous les états et transitions nécessaires à l'exécution de l'instruction ADD et des instructions étudiées dans l'exercice suivant. Un signal entre crochets (par exemple [BEN]) représente une condition à tester, les étiquettes des arcs sortants correspondent aux différentes valeurs de la condition. On utilisera la numérotation des états proposée dans la spécification du LC-2.

Pour toutes les instructions, la phase de chargement comprend les quatre étapes suivantes :

1. le PC est chargé dans MAR et incrémenté,
2. puis l'instruction est lue en mémoire et rangée dans MDR,
3. puis le contenu de MDR est rangé dans IR,
4. enfin la phase de décodage sélectionne les 16 états d'arrivée possibles en fonction du code d'opération.

Par la suite, l'exécution d'une instruction comporte dans la un nombre variable d'étapes (de 1 à 5), en fonction de la complexité des tâches à effectuer.

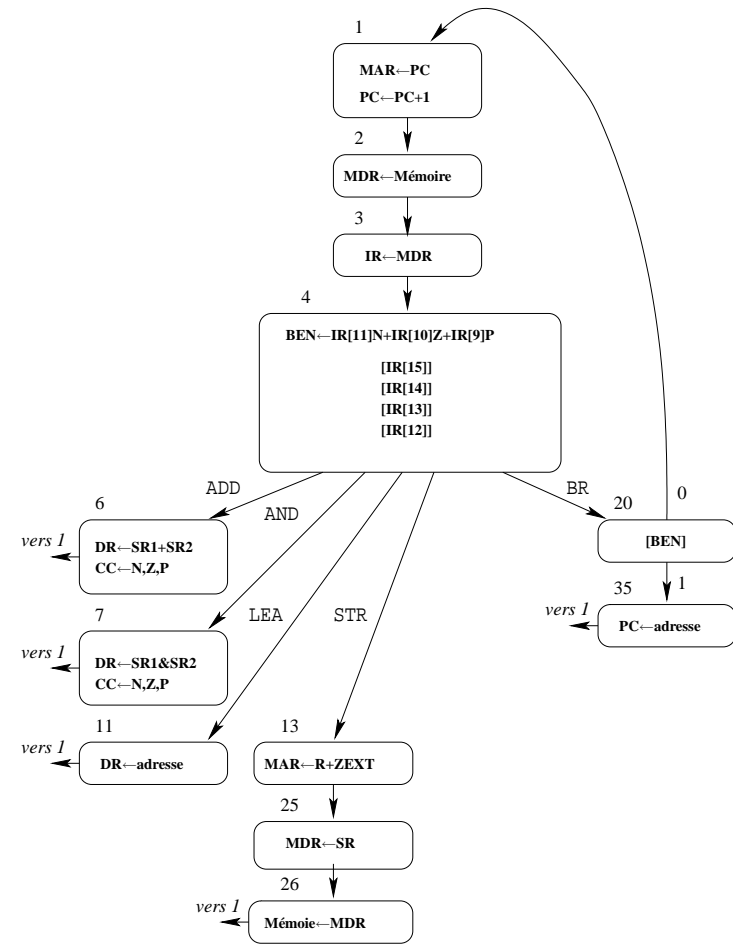


FIG. 14 – Contrôle de quelques instructions du LC-2

Dans le cas d'une addition, deux étapes sont a priori nécessaires : la première exécute l'opération proprement dite sur l'ALU et la deuxième range le résultat dans le banc de registres. Les états et les signaux utiles sont les suivants (toutes les valeurs sont en binaire) :

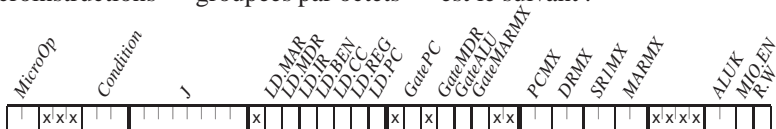
état	signaux de contrôle du LC-2
#1	GatePC ₁ = 1 LD.MAR ₁ = 1 PCMX ₂ = 00 LD.PC ₁ = 1
#2	MIO.EN ₁ = 1 R.W ₁ = 0 LD.MDR ₁ = 1
#3	GateMDR ₁ = 1 LD.IR ₁ = 1
#4	LD.BEN ₁ = 1
#5	SRIMX ₂ = 01 ALUK ₂ = 00
#6	GateALU ₁ = 1 LD.REG ₁ = 1 LD.CC ₁ = 1

Le tableau précédent est conforme à l'usage et aux règles de découpage efficace du traitement des instructions sur un processeur. En pratique, la décomposition en deux étapes de l'addition nécessite le stockage de la sortie de l'ALU dans un registre ; mais ce registre n'a pas été prévu par les concepteurs du LC-2. En conséquence, conformément au contrôle indiqué dans la figure 14, nous proposons une solution combinant en un seul état (6) la traversée de l'ALU et le rangement du résultat de l'addition dans le banc des registres :

état	signaux de contrôle du LC-2
#1	GatePC ₁ = 1 LD.MAR ₁ = 1 PCMX ₂ = 00 LD.PC ₁ = 1
#2	MIO.EN ₁ = 1 R.W ₁ = 0 LD.MDR ₁ = 1
#3	GateMDR ₁ = 1 LD.IR ₁ = 1
#4	LD.BEN ₁ = 1
#6	SRIMX ₂ = 01 ALUK ₂ = 00 GateALU ₁ = 1 LD.REG ₁ = 1 LD.CC ₁ = 1

Exercice 4.2 - Microprogrammation des instructions

On dispose d'un microcontrôleur du même type que celui construit au TD précédent. Ce microcontrôleur permet de réaliser le contrôle microprogrammé du LC-2 à l'aide de microinstructions simples et de branchements conditionnels ou non. Il comporte 6 signaux de condition de branchement : le signal \overline{BEN} (qui vaut 1 lorsque le branchement ne doit pas être pris) est sélectionné lorsque $Condition_3 = 000$, et les 5 bits de poids fort du registre d'instruction IR correspondent aux valeurs de $Condition_3$ de 011 à 111. 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. Le format détaillé des microinstructions — groupées par octets — est le suivant :



(Les 6 signaux concernant la gestion des interruptions ont été omis de cette version simplifiée.)

Question 4.2.1

En utilisant ce microséquenceur, réaliser le microprogramme qui permet de contrôler le processeur pour l'exécution de l'instruction ADD ci-dessus. Tester ce microprogramme.

Réponse

Les microinstructions effectuant un branchement conditionnel sont associées aux états 4, 0???, 00?? et 000? : à l'aide de 4 microinstructions conditionnelles successives, on saute à l'état 6 lorsque l'opcode (0001) correspond à l'instruction ADD. Les microinstructions et leurs adresses sont décrites par le tableau suivant (toutes les valeurs sont en binaire, sauf l'état) :

état	adresse	signaux de contrôle	adresse suivante	LD. ?	Gate ?	?MX	divers
#1	00000000	01...000	00000000	.1000001	-1.000..	00000000	...0000
#2	00000001	01...000	00000000	.0100000	-0.000..	00000000	...0010
#3	00000010	01...000	00000000	.0010000	-0.100..	00000000	...0000
#4	00000011	11...111	00000111	.0001000	-0.000..	00000000	...0000
0???	00000100	11...110	00000111	.0000000	-0.000..	00000000	...0000
00??	00000101	11...101	00000111	.0000000	-0.000..	00000000	...0000
000?	00000110	11...100	00001000	.0000000	-0.000..	00000000	...0000
Stop	00000111	00...000	00000000	.0000000	-0.000..	00000000	...0000
#6 (opcode 0001)	00001000	10...000	00000000	.0000110	-0.010..	00000100	...0000

Les 4 microinstructions de branchement conditionnel testent successivement les 4 bits de poids fort 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 = #8 (état 6).

L'adresse 00000111 = #7 implémente la microinstruction « Stop », il s'agit d'un opcode non reconnu (à ce stade de l'implémentation).

Voir les fichiers `add_slc2_micro_*.ram` correspondant aux 6 ROM de microprogramme, et `add_slc2_control.lgf` pour le circuit de contrôle avec les ROM pré-chargées.

Question 4.2.2

Écrire l'organigramme de contrôle des instructions AND, BR, STR et LEA, puis compléter le microprogramme précédent pour contrôler l'exécution de ces 4 instructions supplémentaires.

Réponse

L'organigramme de contrôle est décrit sur la figure 14.

- La microprogrammation de l'instruction AND à l'état 7 ne diffère de l'instruction ADD par le signal $ALUK_2 = 01$.
- Pour la microprogrammation de l'instruction BR, on doit réaliser les microinstructions correspondant aux états 20 et 35 :
 - pour l'état 20, on doit sauter à l'état 1 si $BEN_1 = 0$, c'est-à-dire $\overline{BEN}_1 = 1$, on utilise donc une microinstruction de branchement conditionnel avec $Condition_3 = 000$;
 - pour l'état 35, on range le résultat du calcul d'adresse dans PC, $LD.PC_1 = 1$ et $PCMX_2 = 11$, et on retourne à l'état 1.

état	adresse	signaux de contrôle	adresse suivante	LD. ?	Gate ?	?MX	divers
#20	000000	...0100	..111011	.0000000	.0000000	00000000	000-0000
#35	111111	...0000	..111011	.0000001	.0000000	11000000	000-0000

L'instruction LDR est implémentée par les états 16, 29 et 30 :

- pour l'état 16, l'offset sur 6 bits étendu avec des zéros est ajouté au registre de base (SRIMX=01), et le résultat est rangé dans MAR en posant LD.MAR = 1, GateMARMX = 1 et MARMX = 01 ;
- pour l'état 29, la valeur lue est rangée dans MDR (LD.MDR = 1, MIO.EN = 1, R.W = 0, ainsi que COND = 01 et J = 110001 pour boucler sur l'état 29) ;
- pour l'état 30 ; la valeur de MDR est transférée dans le banc de registres en posant LD.REG = 1, LD.CC = 1, GateMDR = 1 et DRMX=00.

état	adresse	signaux de contrôle	adresse suivante	LD. ?	Gate ?	?MX	divers
#16	000110	...0000	..110001	.1000000	.0000100	00000101	000-0000
#29	110001	...0010	..110001	.0100000	.0000000	00000000	000-0010
#30	110011	...0000	..111011	.0000110	.0010000	00000000	000-0000

Deux autres instructions sont nécessaires pour exécuter le code de la question suivante, on les décrit brièvement.

L'instruction LEA concatène l'offset de page sur 9 bits avec les 7 bits de poids fort du PC, puis traverse MARMX et le bus pour ranger le résultat dans le banc des registres :

état	adresse	signaux de contrôle	adresse suivante	LD. ?	Gate ?	?MX	divers
#11	001110	...0000	..111011	.0000010	.0000100	00000010	000-0000

L'instruction STR est implémentée par les états 13, 25 et 26. L'offset (ou la base) sur 6 bits étendu avec des zéros est ajouté au registre de base (ou d'offset), et le résultat est rangé dans MAR. La valeur issue du banc des registres traverse l'ALU (ALUK = 11) puis est rangée dans MDR puis dans la mémoire (boucle sur l'état 26) :

état	adresse	signaux de contrôle	adresse suivante	LD. ?	Gate ?	?MX	divers
#13	000110	...0000	..111101	.1000000	.0000100	00000101	000-0000
#25	111101	...0000	..111001	.0100000	.0001000	00000000	000-1100
#26	111001	...0010	..111001	.0000000	.0000000	00000000	000-0011

Le reste du microprogramme suit les mêmes règles de construction. Le microprogramme complet — avec la gestion des interruptions — est donné dans le tableau suivant.

état	adresse	signaux de contrôle	adresse suivante	LD. ?	Gate ?	?MX	divers
#1	111011	...0001	..010000	.1000001	.1000000	00000000	000-0000
#2	010000	...0010	..010000	.0100000	.0000000	00000000	000-0010
#3	010010	...0000	..110010	.0010000	.0010000	00000000	000-0000
#4	110010	...1000	..000000	.0001000	.0000000	00000000	000-0000
#5	001000	...0000	..010001	.1000000	.0001000	00001100	000-1100
#6	000001	...0000	..111011	.0000110	.0001000	00000100	000-0000
#7	000101	...0000	..111011	.0000110	.0001000	00000100	000-0100
#8	001001	...0000	..111011	.0000110	.0001000	00000100	000-1000
#9	001111	...0000	..110101	.1000000	.0000100	00000000	000-0000
#10	001101	...0000	..111011	.0000001	.0000000	10001000	000-0000
#11	001110	...0000	..111011	.0000010	.0000100	00000010	000-0000
#12	000011	...0000	..111101	.1000000	.0000100	00000010	000-0000
#13	000111	...0000	..111101	.1000000	.0000100	00000101	000-0000
#14	001011	...0000	..110100	.1000000	.0000100	00000010	000-0000
#15	001010	...0000	..111100	.1000000	.0000100	00000010	000-0000
#16	000110	...0000	..110001	.1000000	.0000100	00000101	000-0000
#17	000010	...0000	..110001	.1000000	.0000100	00000010	000-0000
#18	001100	...0110	..100000	.0000000	.0000000	00000000	000-0000
#19	000100	...0110	..100100	.0000000	.0000000	00000000	000-0000
#20	000000	...0100	..111011	.0000000	.0000000	00000000	000-0000
#21	110101	...0010	..110101	.0100010	.1000000	00010000	000-0010
#22	110111	...0000	..111011	.0000001	.0010000	01000000	000-0000
#23	110100	...0010	..110100	.0100000	.0000000	00000000	000-0010
#24	110110	...0000	..111101	.1000000	.0010000	00000000	000-0000
#25	111101	...0000	..111001	.0100000	.0001000	00000000	000-1100
#26	111001	...0010	..111001	.0000000	.0000000	00000000	000-0011
#27	111100	...0010	..111100	.0100000	.0000000	00000000	000-0010
#28	111110	...0000	..110001	.1000000	.0010000	00000000	000-0000
#29	110001	...0010	..110001	.0100000	.0000000	00000000	000-0010
#30	110011	...0000	..111011	.0000110	.0001000	00000000	000-0000
#31	100001	...0000	..100000	.0000010	.1000000	00010000	000-0000
#32	100000	...0000	..111011	.0000001	.0000100	01000101	000-0000
#33	100101	...0000	..100100	.0000010	.1000000	00010000	000-0000
#34	100100	...0000	..111011	.0000001	.0000000	11000000	000-0000
#35	111111	...0000	..111011	.0000001	.0000000	11000000	000-0000
#36	010001	...0010	..010001	.0100000	.0000000	00000000	000-0010
#37	010011	...0000	..111000	.0000100	.0010000	00000000	001-0000
#38	111000	...0000	..010100	.1000010	.0001000	00101100	100-0000
#39	010100	...0010	..010100	.0100000	.0100000	00000000	000-0010
#40	010110	...0000	..111010	.0000001	.0010000	01000000	000-0000
#41	111010	...0000	..111011	.0000010	.0001000	00101100	100-0000
#42	110000	...0000	..011001	.1000010	.0001000	00101100	010-0000
#43	011001	...0000	..010101	.0100000	.0100000	00000000	000-0000
#44	010101	...0010	..010101	.0000000	.0000000	00000000	000-0011
#45	010111	...0000	..011011	.1000010	.0001000	00101100	010-0000
#46	011011	...0000	..011000	.0100000	.0000001	00000000	000-0000
#47	011000	...0010	..011000	.0000000	.0000000	00000000	000-0011
#48	011010	...0000	..011100	.1000000	.0000010	00000000	000-0000
#49	011100	...0010	..011100	.0100000	.0000000	00000000	000-0010
#50	011110	...0000	..111011	.0000001	.0010000	01000000	000-0000

Exercice 4.3 - Programmation en assembleur

On propose d'écrire un petit programme en assembleur, et de l'utiliser pour tester le microprogramme de l'exercice précédent.

Question 4.3.1

Écrire un programme en assembleur LC-2 correspondant au code C suivant, en n'utilisant que les instructions de la question 4.2.2 :

```
for (i=0; i<10; i++) {
    a[i] = i;
}
```

Réponse

```
.ORIG    x0000
LEA     R0, Tableau ; 11100000 00001001
AND     R1, R1, #0 ; 01010010 01100000
Boucle  ADD     R2, R1, #-10 ; 00010100 01110110
BRz     Fin      ; 00000100 00001000
STR     R1, R0, #0 ; 01110010 00000000
ADD     R0, R0, #1 ; 00010000 00100001
ADD     R1, R1, #1 ; 00010010 01100001
BRnzip  Boucle   ; 00001110 00000010
Fin     ; boucle infinie (HALT n'est pas disponible)
BRnzip  Fin      ; 00001110 00001000
Tableau .BLKW   #10
.END
```

Question 4.3.2

Écrire le code machine correspondant à ce programme assembleur, et vérifier son exécution sur le LC-2.

Réponse

Le code machine est donné en partie droite du programme assembleur précédent. Il est contenu dans les fichiers `boucle_hi.ram` et `boucle_lo.ram`, à charger dans la RAM du LC-2 à l'adresse `x0000`.