Écrivez un programme LMC qui imprime le carré des nombres consécutifs entre 1 et 20.

7.2 Supposons que les instructions suivantes se trouvent aux emplacements :

```
20 LDA 50
21 ADD 51
50 724
51 006
```

a. Affichez le contenu de l'IR, du PC, du MAR, du MDR et de A à la fin de l'instruction 20.

PC = 21 MDR = 724
IR = 550
MAR = 50
A = 724

LDA 50 ⇒ 550
opCode Adresse

b. Affichez le contenu de chaque registre à chaque étape du cycle *fetch-execute* lors de l'exécution de l'instruction 21.

	Fetch	Execute
IR	151	151
PC	21	22
MAR	21	51
MDR	151	6
A	724	730

Fetch - Execute

1. Fetch

MAR ← PC
MDR ← Memoire[MAR]
IR ← MDR

F

PC = 20
MAR ← PC = 20
MDR ← Memoire[20] = 550
IR ← 550

2 Execute

MAR ← IR[adr]
MDR ← Memoire[MAR]
... Dépend des instructions
PC ← PC + 1

E

MAR ← IR(adr) = 50
MDR ← Memoire[50] = 724
Acc ← 724
PC ← 21

7.7

a. Quel est l'effet de décaler un nombre non signé dans un registre de deux bits à gauche ? D'un bit à droite ? Supposons que des 0 sont insérés pour remplacer les bits à la fin du registre qui sont devenus vides en raison du décalage.

$<< 2 \Rightarrow \times 2^2 \Rightarrow \times 4$

PC $\overset{21}{10101} << 2$ $\quad \overset{84}{1010100}$ → Risque de overflow

$>> 1 \Rightarrow // 2$ Effet: Division par 2

PC=1 $1 // 2 \Rightarrow 0$ Toujours le bon résultat

b. Supposons que le numéro soit signé, c'est-à-dire stocké en complément à 2. Quel est maintenant l'effet de décaler le nombre ?

Gauche : Multiplie (par 2) la valeur du registre => Risque de overflow

Droite : Division par 2 Risque de perdre le signe de la valeur

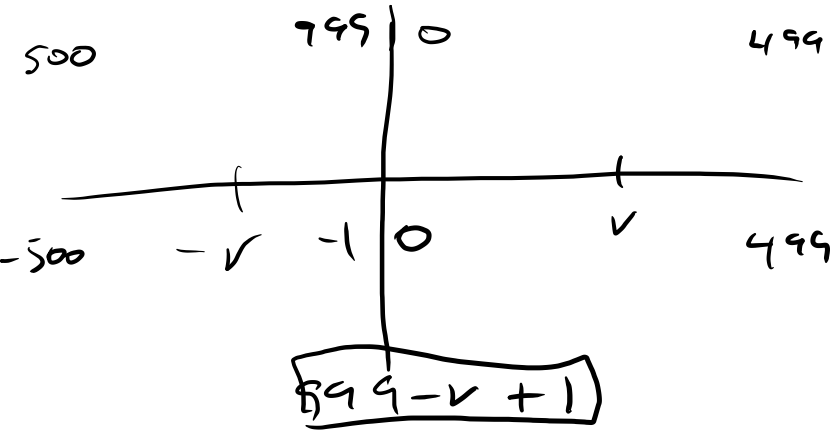
c. Supposons que le décalage exclue le bit de signe, de sorte que le bit de signe reste toujours le même. De plus, supposons que lors d'un décalage à droite, le bit de signe est toujours utilisé comme bit d'insertion à la fin du nombre (au lieu de 0). Quel est l'effet de ces changements ?

4 bits $1101 > 1 = 1110$

Ca2 $-8+4+1 \Rightarrow -3$ $\quad -8+4+2 \Rightarrow -2$

Gauche : Multiplie par 2

Droite : Division par 2 Résultats corrects



7.9 En utilisant les opérations de registre, montrez le cycle *fetch-execute* pour une instruction qui produit le complément à 2 du nombre dans A.

	Fetch	Execute
IR	Mémoire[PC_0]	Mémoire[PC_0]
MAR	PC_0	PC_0
MDR	Mémoire[PC_0]	Mémoire[PC_0]
PC	PC_0	PC_0+1
A	V	999 - V + 1

ADD	Fetch	Exec
IR	160	160
MAR	20	60
MDR	160	10
PC	20	21
A	5	5+10=15

Montrez le cycle *fetch-execute* pour une instruction qui efface A (mise à zéro de A).

	Fetch	Execute
IR	Mémoire[PC]	Mémoire[PC]
MAR	PC	PC
MDR	Mémoire[PC]	Mémoire[PC]
PC	PC	PC+1
A	✓	0

7.12 La *Little Prince Computer* est une mutation de la LMC. La LPC a une instruction supplémentaire. Cette instruction requiert deux mots consécutifs :

PC →
```
0XX
0YY
```

Cette instruction, *move*, déplace les données directement de l'emplacement `XX` vers l'emplacement `YY` sans affecté la valeur dans l'accumulateur.

Pour exécuter cette instruction, le petit prince aurait besoin de stocker les données `XX` temporairement. Il peut le faire en écrivant la valeur dans un morceau de papier et en le gardant jusqu'à ce qu'il récupère la deuxième adresse. L'équivalent dans un vrai CPU pourrait s'appeler le registre d'adresse intermédiaire (IAR).

Écrivez le cycle *fetch-execute* pour l'instruction MOVE de la LPC.

	Fetch	Execute	Fetch	Execute
IR	0xx	0xx	0yy	0yy
MAR	PC_0	xx	PC_0+1	yy
MDR	0xx	0xx	0yy	mémoire[xx]
PC	PC_0	PC_0+1	PC_0+1	PC_0+2
IAR	précédent	xx	xx	xx
A	✓	✓	✓	✓

STO 99

	Fetch	Execute
IR	399	399
MAR	1	99
MDR	399	10
PC	1	2
A	10	10