Program Counter

PC → Valeur de l'adresse courante 2 digits

Memory Address Register

MAR → Valeur d'une adresse 2 digits

Memory Data Register

MDR → Valeur d'une instruction
où opérande 3 digits

Instruction Register

IR → Valeur de l'instruction courante

Accumulateur

A → Valeur d'opérandes lues en mémoire ou autres valeur intermédiaire

Cycle Fetch - Decode - Execute

Fetch

MAR ← PC

MDR ← Mémoire [MAR]

IR ← MDR

Decode IR [opcode]

Execute

Variable selon l'instruction

MAR ← IR [adr]

MDR ← Mémoire [MAR]

...

PC++ PC = PC + 1

LDA

Fetch

MAR ← PC 20

MDR ← Mémoire[MAR] 550

IR ← MDR 550

Execute

MAR ← IR[adr]

MDR ← Mémoire[MAR]

A ← MDR

PC ← PC+1

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.

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

a) PC → 21

IR → 550

MAR → 50

MDR → 724

A → 724

b) Add

Fetch

MAR ← PC 21

MDR ← Mémoire[MAR] 151

IR ← MDR 151

PC → 22

IR → 151

MAR → 21

MDR → 151

A → 724

Execute

MAR ← IR[adr]

MDR ← Mémoire[MAR]

A ← A + MDR 730

PC ← PC+1

PC → 22

IR → 151

MAR → 51

MDR → 006

A → 730

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.

RC = 23

$16 + 4 + 2 + 1$

$2^4 \quad 2^2 \quad 2^1 \quad 2^0$

23

0 0 0 1 0 1 1 1 0

0 0 1 0 1 1 1 0 0

$2^4 + 2^2 + 2^1 + 2^0 = 23$

Multiplier la valeur par 4

Risque de overflow puisque les registres ont des positions fixes

Division de la valeur par 2

$2^6 + 2^4 + 2^3 + 2^2 = 64 + 16 + 8 + 4$

$= 92$

$(23 \times 2) \div 2$

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 ?

C2

00011100 positif

11100010 négatif

positif

Multiplications par 2 selon le nombre de bits

Restraint par la taille du registre

Multiplication par 2

Restraint par la taille du registre

00011100 Positif

Divisé par 2 selon le nombre de bits

2 bits

11100010 Ne fait pas la division par 2 car le signe change de négatif à positif

On prend pas en compte le signe

00 111100 → Pas le bon résultat

On prends en compte le signe

11 111100 → Resultat juste

1100

$\rightarrow -2^3 + 2^2 \Rightarrow -8+4 \Rightarrow -4$

1000 $\Rightarrow -2^3 \Rightarrow -8$

0000 $\Rightarrow 0$

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 ?

Les multiplications et divisions sont bien entreprises

LDA myVal
ADD otherVal
STO myVal

LDA myVal
SOR
STO myVal

00
01

INV
CA2

A ← -10

5+

LDA squared
ADD carré
STO squared

5+5+5+5+5

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.

instruction qui fait le complément
- A ← 1011
- ↓
- A ← 0101

Fetch
- MAR ← PC
- MDR ← Mémoire[PC]
- IR ← MDR

Execute
- A ← (à déterminer) (opcode)
- PC ← PC+1

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

INV 15

A ← 15

RST

A ← 0

INV 15

A ← 15

LDA zero

...

zero DAT 000

A ← 0

Fetch

MAR ← PC

MDR ← mémoire[MAR]

IR ← MDR

Execute

A ← 0

PC ← PC+1

IN

FETCH

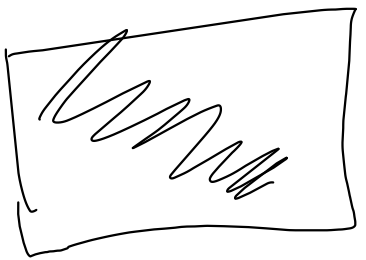
00 PC ← PC0 — Pour toutes instructions

00 MAR ← PC

001 MDR ← Mem[MAR]

101 IR ← MDR

Execute — Dépend de l'instruction



PC ← PC + 1

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

```
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.

→ MOVE adr1 → adr2 0xx 0yy

	Fetch	Execute	Fetch	Execute
PC	PC_0	PC_0+1	PC_0+1	PC_0+2
IR	0xx	0xx	0yy	0yy
MAR	PC_0	PC_0	PC_0+1	xx
MDR	Mem[PC_0] = 0xx	0xx	Mem[PC_0+1] = 0yy	Mem[xx]
A	A_0	A_0	A_0	A_0
IAR	IAR_0	xx	xx	xx

MAR MDR → Interagir avec la mémoire

IAR → Garder des adresses