Systèmes à microprocesseurs

Chapitre 3 : Jeu d'instructions du µC 8051

J.ZAIDOUNI
Université Mohammed Premier
ENSA Oujda, 2017/2018

Sommaire

1 Syntaxe générale d'une instruction

2 Modes d'adressage
- Adressage Immédiat
- Adressage Registre
- Adressage direct
- Adressage Indirect
- Adressage de Bits

3 Instructions de transfert de données
- Instruction MOV, MOVC et MOVX
- Instructions XCH, XCHD, PUSH et POP

Sommaire

4 Instructions arithmétiques
- Instructions ADD, ADDC, SUBB, INC, DEC, MUL, DIV et DA

5 Instructions logiques et booléennes
- Instructions ANL, ORL, XRL, CLR, CPL et SETB
- Instructions RL, RLC, RR, RRC et SWAP

6 Instructions de branchement non conditionnels
- Instructions ACALL, AJMP, LCALL, LJMP, SJMP et JMP

7 Instructions de branchement conditionnels
- Instructions J(N)Z, J(N)C, J(N)B, CJNE et DJNZ

Sommaire

8 Instructions Diverses
- Instructions RET, RETI et NOP

9 Instructions modifiant les drapeaux du PSW

10 Instructions manipulant les Ports : 3 Types

11 Directives
- 19 Directives de OPTIMA 51
- Directives ORG, D(C)SEG, D(S)B et DW
- Directives BIT, EQU et END

12 Exemple de programme
- Programme (Fichier.LST)
- Programme (Fichier.HEX)
- Taille du code machine
- Temps d'exécution du programme

Jeu d'instructions

- Le µC 8051 possède un jeu d'instructions composé de 111 instructions.
- Chaque instruction s'exécute sur 1, 2 ou 4 Cycles Machine (CM).
- $1\ \textit{Cycle Machine} = 1\ \textit{Cycle Instruction} = 12.\textit{Th} = 12/\textit{Fh}$.
- Avec $\textit{Fh}$ représente la fréquence de fonctionnement (horloge, quartz) et $\textit{Th} = 1/\textit{Fh}$ est la période.
- La fréquence $\textit{Fh}$ peut prendre des valeurs de $1.24\,\textit{MHz}$ à $12\,\textit{MHz}$ (au maximum).

Jeu d'instructions

- Les instructions sont réparties selon la taille de leur code :
  - 49 instructions codées sur 1 octet (Opcode)
  - 45 instructions codées sur 2 octets (Opcode+donnée)
  - 17 instructions codées sur 3 octets (Opcode+donnée1+donnée2)
- Les instructions sont réparties selon le temps d'exécution :
  - 64 instructions codées sur 1 CM
  - 45 instructions codées sur 2 CM
  - 2 instructions codées sur 4 CM (multiplication et division)

Syntaxe générale

1. Instruction :
   - Syntaxe : [label :] Mnémonique [opérandes] [;commentaire]
   - Mnémonique = nom de l'instruction (ADD, INC...).
   - Mnémonique produit du code machine.
   - label (facultatif) : étiquette de l'instruction, traduite par l'adresse dans la mémoire programme.
   - opérandes (facultatif) : les opérandes de l'instruction (0, 1, 2 opérandes)
   - commentaire (facultatif) : texte qui explique l'utilité de l'instruction dans le programme.
2. Directive :
   - Syntaxe : [label :] Directive [opérandes] [;commentaire]
   - Directive ne produit pas de code machine, utilisé seulement par l'assembleur.

Symboles utilisés

Symbole	Signification
Rn	Un des registres actifs R0 à R7
direct	Adresse directe d'un octet de RAM interne, d'un port, ou SFR
@Ri	Adresse indirecte par registre R0 ou R1
#data	Donnée 8 bits
#data16	Donnée 16 bits
bit	Adresse directe au niveau du bit
rel	Déplacement 8 bits signé relatif au PC : de -128 à +127
addr11	Adresse limitée au bloc de 2Koctets dans lequel figure l'instruction
addr16	Adresse sur l'espace de 64Koctets

Modes d'adressage

- Modes d'adressage (5 modes) :
  1. Adressage Immédiat
  2. Adressage Registre
  3. Adressage direct
  4. Adressage Indirect (Adressage indexé)
  5. Adressage de Bits

Adressage Immédiat

- Ce n'est pas réellement un adressage car la donnée constituant l'opérande est précisée dans l'instruction.
- Dans l'écriture en assembleur, on place un signe # devant la donnée pour la différentier d'une adresse (#donnee).
- Exemples :
  - MOV A,#40H ; chargement $A \leftarrow 40H$
  - MOV A,#'a' ; $A \leftarrow 61H$
  - MOV A,#40 ;$A \leftarrow (40)_{10} = 28H$
  - MOV A,#11001101B ; $A \leftarrow (1100\,1101)_2 = CDH$
  - MOV P1,#80H ; $P1 \leftarrow 80H$
  - MOV 40h,#5AH ; $[40H] \leftarrow 5AH$

Adressage Immédiat

- Exemples :
  - MOV DPTR,#4521H ; $DPTR \leftarrow 4512H$
  - MOV DPL,#21H ;équivalent à l'instruction précédente $DPL \leftarrow 21H$
  - MOV DPH,#45H ; $DPH \leftarrow 45H$
  - MOV DPTR,#68975 ;erreur Valeur $>$ 65535 $(2^{16}-1)$ (FFFFH)
- On peut utiliser la directive EQU pour accéder à une donnée immédiate

```
nbre EQU 30 ; nbre=30=1EH
caractere EQU 'a' ; caractere=61H
...
MOV R4,#nbre ;R4 ← (30)10 = 1EH
MOV A,#caractere ; A ← 61H
```

Adressage Registre

- L'opérande est contenu dans un registre : A, R0 à R7, DPTR, C (Carry) et (AB) pour MUL et DIV.
- Exemples : Adressage Registre (en rouge)
  - ADD A,R4 ; $A \leftarrow A + R4$
  - INC A ; $A \leftarrow A + 1$
  - INC R3 ; $R3 \leftarrow R3 + 1$
  - MOV A,R0 ; $A \leftarrow R0$
  - MOV R2,A ;$R2 \leftarrow A$
  - MOV DPTR,A ; erreur car instruction qui n'existe pas et DPTR et A n'ont pas la même taille

Adressage direct

- C'est l'adresse de l'opérande qui est précisée dans l'instruction.
- L'adresse n'a que 8 bits, Seule la RAM interne et les SFRs peuvent être adressés de cette manière.
- Il n'existe pas d'adressage direct pour la RAM externe, il faut passer par le registre DPTR.
- Exemples : Adressage direct (en rouge)
  - MOV A,45h ; $A \leftarrow [45H]$
  - MOV R0,30H ; $R0 \leftarrow [30H]$
  - MOV A,4 ; adressage direct $A \leftarrow [04H]$ si bank 0 est actif, $[04H] = R4$ ou —>
  - MOV A,04H ;adressage direct $A \leftarrow [04H] = R4$
  - MOV A,R4 ; adressage registre $A \leftarrow R4$
  - MOV 06H,#10H ; $[06H] \leftarrow 10H$ si bank 0 est actif, $[06H] = R6 \leftarrow 10H$

Adressage direct

- Les registres SFR (Special Function Register) sont accessible par leur nom (adressage registre ou directe) ou leur adresse (adressage directe) (de 80H à FFH).

MOV 0E0H,#55H ; $[E0H] = A \leftarrow 55H$ (adressage directe) ou
MOV A,#55h ; $A \leftarrow 55H$ (adressage registre)
... ; (adressage immédiat)
...
MOV 0F0H,R0 ; $[F0H] = B \leftarrow R0$ (adressage directe) ou
MOV B,R0 ; $B \leftarrow R0$ (adressage registre) ici B est une constante (définie par EQU) = F0H adresse de B ! ! !

- Les locations non utilisées de 80H à FFH sont réservées et ne peuvent pas être utilisées par le programmeur.

Adressage direct

- Seulement l'adressage directe peut être utilisée pour empiler ou déplier une donnée dans la pile.
- PUSH A ; est non valide
- PUSH 0E0H ; empile le registre A dans la pile (adresse de A= E0H)
- ou PUSH ACC ; avec ACC est une constante = E0H (déjà définie par EQU dans µVision).

```
PUSH 05 ;empiler R5 dans la pile
PUSH 0E0H ;empiler A dans la pile ou PUSH ACC
...
POP 0F0H ;déplier la pile dans B eq. B=A ou POP B
POP 02 ;déplier la pile dans R2 eq. R2=R5
```

Adressage Indirect (indexé)

- L'instruction spécifie un registre qui contient l'adresse de l'opérande (pointeurs).
- La mémoire interne (RAM interne) et externes (RAM et ROM) sont accessibles par ce type d'adressage (sauf les registres SFR).
- Adressage sur 8 bits (RAM interne) : Les 2 registres R0 ou R1 sont utilisées comme pointeurs de données.
- Adressage 16 bits (RAM externe et ROM interne ou externe) : Le DPTR (Data Pointer) est utilisé pour pointer sur une donnée de 16 bits).
- Le signe @ est utilisé pour désigner un adressage indirect.

Adressage Indirect (indexé)

- Exemples RAM interne (@R0 et @R1) :
  - MOV A,@R0 ; $A \leftarrow [R0]$
  - MOV @R1,P1 ; $[R1] \leftarrow P1$
  - ANL A,@R0 ; $A \leftarrow A\,AND\,[R0]$
- Exemples RAM externe (@DPTR) :
  - MOVX @DPTR,A ; $[DPTR] \leftarrow A$
  - MOVX A,@DPTR ; $A \leftarrow [DPTR]$
  - Le X du mnémonique indique qu'il s'agit d'une mémoire RAM externe (Données).
- Exemples ROM interne ou externe :
  - MOVC A,@A+DPTR ; $A \leftarrow [A+DPTR]$
  - Le C du mnémonique indique qu'il s'agit d'une mémoire ROM interne ou externe (Code).

Adressage Indirect (indexé)

- Remarque : L'utilisation des registres d'index R0 et R1 avec la mémoire externe est possible si on utilise le Port 2 comme octet haut du bus d'adresse (= adresse de la page, $2^8 = 256$ pages de 0 à 255 avec 1 page=2^8 octets).
  total=$2^8.2^8 = 2^{16} = 64\,Koctets$.
- Les 3 instructions suivantes permettent de charger l'accumulateur avec le contenu de la case mémoire externe d'adresse 43A0H

  MOV P2,#43H ; $P2 \leftarrow 43H$ = octet haut du bus d'adresse
  MOV R0,#0A0H ; $R0 \leftarrow A0H$
  MOVX A,@R0 ; $A \leftarrow [P2\ concaténé\ avec\ R0] = [43A0H]$

Adressage de Bits

- Le 8051 possède un processeur de bits qui travaille sur des bits individuels.
- Pour ce processeur l'Accumulateur est C (Carry, Retenue).
- Les bits de certains registres sont ainsi accessibles par une adresse sur 1 octet.
- D'abord les bits de la zone en début de RAM interne (16 octets de 20H à 2FH).
- Ces 128 bits qui commencent du LSB (B0) de la case 20H et se terminent au MSB (B7) de la case 2FH ont des adresses qui vont de 00 à 7FH.
- Aussi, les bits des SFR dont l'adresse en hexadécimal se termine par 0 ou 8, 80H 88H 90H 98H etc...
- L'un des opérandes est toujours le carry C.

Adressage de Bits

- Exemples (adresse registre bit C : en bleu, adresse bit : en rouge) :
  - MOV C,45H : Chargement de C avec le bit d'adresse 45H
  - MOV P1.3,C : Le contenu de C est transféré dans le bit 3 du port P1
  - MOV RS0,C : Le bit C est placé dans le bit RS0 du Registre PSW
  - CLR C : mise à zéro de C
  - CLR P3.6 : mise à zéro du bit 6 de P3
  - SETB RS1 : mise à 1 de RS1
  - CPL P1.1 : Inversion du bit 1 de P1
  - SETB P3.2 : Mise à 1 du bit 2 du port 3

Instructions de transfert de données

- Instructions de transfert de données : MOV, MOVC, MOVX, XCH, XCHD, PUSH et POP
- MOV : déplacement de donnée dans la mémoire interne.
- MOV Dest,Source : recopie Source dans Dest avec aucun drapeau n'est positionné.

Instruction MOV

Syntaxe	Code	Octets	cycles	Flags
MOV A, Rn	E8+n	1	1	-
MOV A, direct	E5	2	1	-
MOV A, @Ri	E6+i	1	1	-
MOV A, #data	74	2	1	-
MOV Rn, A	F8+n	1	1	-
MOV Rn, direct	A8+n	2	2	-
MOV Rn, #data	78+n	2	1	-
MOV direct, A	F5	2	1	-
MOV direct, Rn	88+n	2	2	-
MOV direct, direct	85	3	2	-
MOV direct, @Ri	86+i	2	2	-
MOV direct, #data	75	3	2	-
MOV @Ri, A	F6+i	1	1	-
MOV @Ri, direct	A6+i	2	2	-
MOV @Ri, #data	76+i	2	1	-
MOV DPTR, #data16	90	3	2	-
MOV bit, C	92	2	2	-
MOV C, bit	A2	2	1	C

Instructions de transfert de données

- Notons que l'on peut transférer des données directement d'une case mémoire interne dans une autre sans passer par l'accumulateur, par Adressage direct.
- Les instructions du genre MOV Rn,Rn ; ne sont pas autorisé.
- Pour réaliser M0V R2,R4 avec le bank 0 actif, il faut faire MOV 02,04.
- Si c'est le bank 1 qui est actif, il faut faire MOV 0AH,0CH

Instructions MOVX et MOVC

- Instruction MOVX (Le X = RAM externe)
  - MOVX Dest,Source permet la lecture ou l'écriture d'un octet en RAM externe en passant toujours par l'accumulateur.
  - Seulement l'adressage indirect est utilisé (soit Dest ou Source)
- Instruction MOVC (Le C = ROM interne et externe)
  - MOVC A, @A+<base-reg> permet de lire un octet dans la mémoire programme (ROM interne ou externe).

Instructions MOVX et MOVC

Syntaxe	Code	Octets	cycles
MOVX A,@Ri	E2+i	1	2
MOVX A,@DPTR	E0	1	2
MOVX @Ri,A	F2+i	1	2
MOVX @DPTR,A	F0	1	2
MOVC A,@A+DPTR	93	1	2
MOVC A,@A+PC	83	1	2

Instructions XCH, XCHD, PUSH et POP

- Instruction XCH
  - XCH A,<octet> : échange les données de l'accumulateur A et de l'octet adressé (registre, direct ou indirect).
- Instruction XCHD (D = Digit de l'Héxa = 4 bits = quartet)
  - XCHD A,@Ri : échange les quartets de poids faible entre l'accu A et l'octet adressé.
- Instructions PUSH et POP
  - Sauvegarder (Empiler) ou récupérer (Dépiler) une valeur dans la pile.
  - PUSH incrémente SP puis écrit l'opérande dans la case pointé par ce dernier
  - POP lit la valeur pointée par SP puis décrémente ce dernier

Instructions XCH, XCHD, PUSH et POP

Syntaxe	Code	Octets	cycles
PUSH direct	C0	2	2
POP direct	D0	2	2
XCH A, Rn	C8+n	1	1
XCH A, direct	C5	2	1
XCH A, @Ri	C6+i	1	1
XCHD A, @Ri	D6+i	1	1

Instructions PUSH et POP

- Pile = structure " Dernier entré premier sorti ", Last In First Out (LIFO)
- Deux instructions PUSH et POP :
  - PUSH direct :
    1. Incrémenter SP ($SP \leftarrow SP + 1$).
    2. Empiler le contenu d'une case mémoire (registre ou SFR) dans la pile (càd case d'adresse SP) ($[SP] \leftarrow [direct]$)
  - POP direct :
    1. Dépiler la pile dans une case mémoire (registre ou SFR) (càd retirer la dernière donnée enregistrée dans la case d'adresse SP ($[direct] \leftarrow [SP]$).
    2. Décrémenter SP ($SP \leftarrow SP - 1$).

Instructions PUSH et POP

- Exemples de PUSH : ; ici bank 0 est supposé sélectionné
  MOV R6, #25H
  MOV R1, #12H
  MOV R4, #0F3H ;
  PUSH 06H ; R6 = donnée à empiler dans la pile = 25H.
  PUSH 01H ; R1 = donnée à empiler dans la pile = 12H.
  PUSH 04H ; R4 = donnée à empiler dans la pile = F3H.

initial	
0BH	
0AH	
09H	
08H	
SP=07H	

PUSH 6	
0BH	
0AH	
09H	
08H	25H
SP=08H	

PUSH 1	
0BH	
0AH	
09H	12H
08H	25H
SP=09H	

PUSH 4	
0BH	
0AH	F3H
09H	12H
08H	25H
SP=0AH	

Instructions PUSH et POP

- Exemples de POP : ; ici bank 0 est supposé sélectionné
  POP 03H ; R3 = donnée d'adresse à dépiler de la pile = F3H.
  POP 05H ; R5 = donnée d'adresse à dépiler de la pile = 12H.
  POP 02H ; R2 = donnée d'adresse à dépiler de la pile = 25H.

initial	
0BH	
0AH	F3H
09H	12H
08H	25H
SP=0AH	

POP 3	
0BH	
0AH	
09H	12H
08H	25H
SP=09H	

POP 5	
0BH	
0AH	
09H	
08H	25H
SP=08H	

POP 2	
0BH	
0AH	
09H	
08H	
SP=07H	

Instructions arithmétiques

- ADD : ADD A, <octet> : additionne un octet avec l'accu A, résultat dans A.
- ADDC : ADDC A, <octet> : additionne un octet, l'accu A et la retenue, résultat dans A.
- SUBB : SUBB A, <octet> : soustrait un octet et la retenue du contenu de A, résultat dans A.
- INC : INC <octet> : incrémente un octet ou DPTR.
- DEC : DEC <octet> : décrémente un octet.

Instructions arithmétiques

- MUL : MUL AB : multiplie l'accu A et l'accu B, résultat = B :A (octet fort dans B et octet faible dans A).
- DIV : DIV AB : divise le contenu de A par le contenu de B, quotient dans A et reste dans B.
- DA : DA A : ajustement décimal de A (du binaire vers BCD).

Instructions arithmétiques

ADD A, Rn	28+n	1	1	C, AC, O
ADD A, direct	25	2	1	C, AC, O
ADD A, @Ri	26+i	1	1	C, AC, O
ADD A, #data	24	2	1	C, AC, O
ADDC A, Rn	38+n	1	1	C, AC, O
ADDC A, direct	35	2	1	C, AC, O
ADDC A, @Ri	36+i	1	1	C, AC, O
ADDC A, #data	34	2	1	C, AC, O
SUBB A, Rn	98+n	1	1	C, AC, O
SUBB A, direct	95	2	1	C, AC, O
SUBB A, @Ri	96+i	1	1	C, AC, O
SUBB A, #data	94	2	1	C, AC, O
INC @Ri	06+i	1	1	-
INC A	04	1	1	-
INC direct	05	2	1	-
INC Rn	08+n	1	1	-
INC DPTR	A3	1	2	-
DEC @Ri	16+i	1	1	-
DEC A	14	1	1	-
DEC direct	15	2	1	-
DEC Rn	18+n	1	1	-
MUL AB	A4	1	4	C, O
DIV AB	84	1	4	C, O
DA A	D4	1	1	C

Instruction ADD

- Décimal sur 8 bits : Non signé [0,255], Signé [-128,127]
- C=0 et C67=0 (OV=0)
- $\oplus + \oplus = \oplus$

HEX	Binaire									Décimal	
	C	B7	B6	B5	B4	B3	B2	B1	B0	Non Signé	Signé
	0	**0**	1	1			1	1		**correct**	**correct**
33H		0	0	1	1	0	0	1	1	51	51
+13H	+	0	0	0	1	0	0	1	1	+19	+19
=46H	=	0	1	0	0	0	1	1	0	=60	=60

Instruction ADD

- C=0 et C67=1 (OV=1)
- $\oplus + \oplus = \ominus$

HEX	Binaire									Décimal	
	C	B7	B6	B5	B4	B3	B2	B1	B0	Non Signé	Signé
	0	**1**	1	1		1	1	1		**correct**	**faux**
33H		0	0	1	1	0	0	1	1	51	51
+55H	+	0	1	0	1	0	1	0	1	+85	+19
=88H	=	1	0	0	0	1	0	0	0	=136	=-112

Instruction ADD

- C=1 et C67=0 (OV=1)
- $\ominus + \ominus = \oplus$

HEX	Binaire									Décimal	
	C	B7	B6	B5	B4	B3	B2	B1	B0	Non Signé	Signé
	1	**0**					1	1		**faux**	**faux**
B3H		1	0	1	1	0	0	1	1	179	-94
+83H	+	1	0	0	0	0	0	1	1	+131	-142
=(1)36H	=	0	0	1	1	0	1	1	0	=54	=54

Instruction ADD

- C=1 et C67=1 (OV=0)
- $\oplus + \ominus = \ominus$ *ou* $\oplus$

HEX	Binaire									Décimal	
	C	B7	B6	B5	B4	B3	B2	B1	B0	Non Signé	Signé
	1	**1**	1	1		1	1	1		**faux**	**correct**
33H		0	0	1	1	0	0	1	1	51	51
+D5H	+	1	1	0	1	0	1	0	1	+213	-43
=(1)08H	=	0	0	0	0	1	0	0	0	=8	=8

Instruction DA

- Ne fonctionne qu'après une instruction ADD ou ADDC sur des nombres déjà codés en BCD.
- BCD = Héxa avec condition que chaque digit est compris entre 0 et 9.
  1. Si 4 bits faibles est > 9 ou AC=1, ajouter 0110 aux 4 bits faibles.
  2. Si 4 bits forts est > 9 ou C=1, ajouter 0110 aux 4 bits forts.
- Elle affecte juste le bit C (Carry) du registre PSW

Instruction DA

- Exemple1 (AC=1) :
  MOV A,#29H ; $A \leftarrow 29H$ codé en BCD
  ADD A,#18H ; 18H codé en BCD
  ; $A \leftarrow 41H$, $AC \leftarrow 1$ et $C \leftarrow 0$ (avant DA)
  DA A ; $A \leftarrow 47H$ et $C \leftarrow 0$
  ; Rmq : AC n'est pas modifié par l'instruction DA
  ; AC reste à 1

HEX	BCD	
29H	0010 1001	
+18H	+ 0001 1000	
=41H	= 0100 0001	AC=1 et C=0
+6H	+ 0000 0110	
=47H	= 0100 0111	C=0

Instruction DA

- Exemple2 (4 bits forts > 9) :
  MOV A,#29H ; $A \leftarrow 29H$ codé en BCD
  ADD A,#80H ; 80H codé en BCD
  ; $A \leftarrow A9H$, $AC \leftarrow 0$ et $C \leftarrow 0$ (avant DA)
  DA A ; $A \leftarrow 09H$ et $C \leftarrow 1$

HEX	BCD	
29H	0010 1001	
+80H	+ 1000 0000	
=A9H	= 1010 1001	AC=0 et C=0
+60H	+ 0110 0000	
=(1)09H	= (1)0000 1001	C=1

Instruction MUL

- C'est une multiplication non-signée : A et B sont non-signés.
- MUL AB ;AxB, le résultat sur 16-bit est dans B et A
- B = octet haut du résultat et A = octet bas du résultat
- Elle affecte les bits C (C=0) (Carry) et OV (Overflow) du registre PSW
- OV=1 si résultat $>$ 255 (FFH), sinon OV=0.
- Exemple :
  - MOV A,#10H ;
  - MOV B,#10H ;
  - MUL AB ;10H * 10H = 100H (256) avec $B \leftarrow 01H$ et $A \leftarrow 00H$; $OV \leftarrow 1$

Instruction DIV

- C'est une division non-signée : A et B sont non-signés.
- DIV AB ;A/B
- A = quotient et B = reste
- Elle affecte les bits C (C=0) (Carry) et OV (Overflow) du registre PSW
- OV=1 si (B)=0 (division par 0, avec A et B ne changent pas suite à cette division), sinon OV=0.
- Exemple :
  - MOV A,#0FFH ; FFH=255
  - MOV B,#12H ; 12H=18
  - DIV AB ;$A \leftarrow 0EH$(quotient) et $B \leftarrow 03H$ (reste) ; OV=0

Instructions logiques et booléennes

- ANL : ANL <dest>, <source> : réalise un ET logique entre source et dest, résultat dans dest.
- ORL : ORL <dest>, <source> : réalise un OU logique entre source et dest, résultat dans dest.
- XRL : XRL <dest>, <source> : réalise un OU exclusif entre source et dest, résultat dans dest.
- CLR A ou CLR <bit> : met l'accumulateur ou un bit à 0
- CPL A ou CPL <bit> : Complémente l'accumulateur ou un bit
- SETB : SETB <bit> : met un bit à 1

Instructions logiques et booléennes

- RL, RLC, RR, RRC, SWAP :
  - RL A : rotation vers la gauche du contenu de A
  - RLC A : rotation vers la gauche du contenu de A à travers la retenue
  - RR A : rotation vers la droite du contenu de A
  - RRC A : rotation vers la droite du contenu de A à travers la retenue
  - SWAP A : échange le quartet de poids faible avec celui de poids fort de A

Instructions logiques et booléennes

ANL A, #data	54	2	1	-
ANL A, @Ri	56+i	1	1	-
ANL A, direct	55	2	1	-
ANL A, Rn	58+n	1	1	-
ANL direct, #data	53	3	2	-
ANL direct, A	52	2	1	-
ANL C, /bit	B0	2	2	C
ANL C, bit	82	2	2	C
ORL A, #data	44	2	1	-
ORL A,@Ri	46+i	1	1	-
ORL A, direct	45	2	1	-
ORL A, Rn	48+n	1	1	-
ORL direct, #data	43	3	2	-
ORL direct, A	42	2	1	-
ORL C, /bit	A0	2	2	C
ORL C, bit	72	2	2	C
XRL A, #data	64	2	1	-
XRL A, @Ri	66+i	1	1	-
XRL A, direct	65	2	1	-
XRL A, Rn	68+n	1	1	-
XRL direct, #data	63	3	2	-
XRL direct, A	62	2	1	-

Instructions logiques et booléennes

CLR A	E4	1	1	-
CLR bit	C2	2	1	-
CLR C	C3	1	1	C=0
CPL A	F4	1	1	-
CPL bit	B2	2	1	-
CPL C	B3	1	1	C
RL A	23	1	1	-
RLC A	33	1	1	C
RR A	03	1	1	-
RRC A	13	1	1	C
SWAP A	C4	1	1	-
SETB bit	D2	2	1	-
SETB C	D3	1	1	-

ACALL : Absolute CALL [saut dans un bloc de 2K]

- ACALL addr11 : réalise un saut absolu inconditionnel vers un sous programme commençant à l'adresse addr11.
- On peut sauter dans l'intervalle le même bloc de 2K *de numéro donné par* PC_{15-11} que l'instruction courante.
- Son exécution se déroule en 3 étapes :
  1. $PC \leftarrow PC + 2$; adresse suivante <— adresse courante + 2.
  2. Empiler PC dans la pile
     - $SP \longleftarrow SP + 1$; incrémenter SP
     - $[SP] \longleftarrow PC_{7-0}$;empiler octet bas de PC dans la pile
     - $SP \longleftarrow SP + 1$; incrémenter SP
     - $[SP] \longleftarrow PC_{15-8}$;empiler octet haut de PC dans la pile
  3. $PC_{10-0} \longleftarrow addr11$; 11 bits de poids faibles du PC sont changés par les 11 bits de addr11.
     et PC_{15-11} *reste constant.*

Exemple 1 (à tester sur µVision) : Programm.A51

```
$MOD51 ; This includes 8051 definitions for the Metalink assembler
ORG 0 ; adresse 0 ; pas obligatoire
LJMP debut ; saut long vers debut (programme principal)

org 7598H ;
debut :
ACALL sous_programme ; appel du sous programme
stop : SJMP stop ; boucle infinie qui marque un arrêt

org 7700H ;
sous_programme :
NOP ; 1ère instruction du sous programme
RET ; retour vers le programme principal

END ; fin
```

Fichier Listing : Pogramme.LST

```
LOC  OBJ             LINE     SOURCE

                        1     $mod51  ; This includes 8051 definitions fo
                             Metalink assembler
0000                    2     ORG 0
0000 027598             3     LJMP debut
                        4
7598                    5     org 7598H ;
7598                    6     debut:
7598 F100               7     ACALL sous_programme;
759A 80FE               8     stop: SJMP stop
                        9
7700                   10     org 7700H ;
7700                   11     sous_programme:
7700 00                12     NOP
7701 22                13     RET
                       14
                       15     END
FFA51 MACRO ASSEMBLER  POGRAMME
```

ACALL : Absolute CALL [saut dans un bloc de 2K]

- Le codage se fait d'une façon particulière ($COL = 11H$) :
  - *octet haut du code = COL+ 20H*P = 11H+ 20H*P*
  - *octet bas du code = octet bas de addr11 = Addr8*
- Avec $P = Page = (Ad10\, Ad9\, Ad8)_2$ (P varie entre 0 et 7)

Code instruction ACALL (2 octets)	
octet haut	octet bas
$(Ad10\, Ad9\, Ad8\, 10001)_2$	$(Ad7\, Ad6\, Ad5\, Ad4\, Ad3\, Ad2\, Ad1\, Ad0)_2$
11H+20H*P	Addr8

- Pour l'exemple : nous avons $P = (111)_2 = (7)_{10} = 7H$
  - *octet haut du code = 11H+20H*7H=11H+E0H=F1H*
  - *octet bas du code = octet bas de addr11 = Addr8 = 00H*
  - *donc le code = F100H*

AJMP : Absolute JUMP [saut dans un bloc de 2K]

- AJMP addr11 : réalise un saut absolu inconditionnel vers l'instruction d'adresse addr11.
- *La différence avec ACALL* est que ici on ne saute pas vers un sous programme, *il n'est pas nécessaire d'empiler le PC pour pouvoir retourner* après l'exécution du sous programme.
- Le codage se fait de la même façon que ACALL sauf que *COL = 01H* :
  - *octet haut du code = COL+ 20H*P = 01H+ 20H*P*
  - *octet bas du code = octet bas de addr11 = Addr8*
    Avec $P = Page = (Ad10\ Ad9\ Ad8)_2$ (P varie entre 0 et 7)

Code instruction AJMP (2 octets)	
octet haut	octet bas
$(Ad10\ Ad9\ Ad8\ 00001)_2$	$(Ad7\ Ad6\ Ad5\ Ad4\ Ad3\ Ad2\ Ad1\ Ad0)_2$
01H+20H*P	Addr8

Exemple 2 (à tester sur µVision) : Pogramme.A51

```
$MOD51 ; This includes 8051 definitions for the Metalink assembler
ORG 0 ; adresse 0 ; pas obligatoire
LJMP debut ; saut long vers debut (programme principal)

org 7598H ;
debut :
AJMP etiquette ; appel du sous programme
stop : SJMP stop ; boucle infinie qui marque un arrêt

org 7700H ;
etiquette :
NOP ; 1ère instruction du sous programme
; ATTENTION RET retour vers le programme principal
AJMP stop ;

END ; fin
```

Fichier Listing : Pogramme.LST

```
                          1     $mod51  ; This includes 8051 definitions for
                         Metalink assembler
                          2
0000                      3     ORG 0 ; adresse 0 ; pas obligatoire
0000 027598               4     LJMP debut ; saut long vers debut (programme
                          5
7598                      6     org 7598H ;
7598                      7     debut:
7598 E100                 8     AJMP etiquette; appel du sous programme
759A 80FE                 9     stop: SJMP stop ; boucle infinie qui marque
                         10
7700                     11     org 7700H ;
7700                     12     etiquette:
7700 00                  13     NOP ; 1ère instruction du sous programme
                         14     ; ATTENTION RET n'est pas utilisée
7701 A19A                15     AJMP stop;
                         16
                         17     END; fin
FFA51 MACRO ASSEMBLER  POGRAMME
```

AJMP : Absolute JUMP [saut dans un bloc de 2K]

- Pour l'exemple AJMP etiquette : nous avons
  $P = (111)_2 = (7)_{10} = 7H$
  - *octet haut du code = 01H+20H*7H=01H+E0H=E1H*
  - *octet bas du code = octet bas de addr11 = Addr8 = 00H*
  - *donc le code = E100H*
- Pour l'exemple AJMP stop ; nous avons
  $P = (101)_2 = (5)_{10} = 5H$
  - *octet haut du code = 01H+20H*5H=01H+A0H=A1H*
  - *octet bas du code = octet bas de addr11 = Addr8 = 9AH*
  - *donc le code = A19AH*

LCALL : Long CALL [saut dans 64K]

- LCALL addr16 : réalise un saut long inconditionnel vers un sous programme commençant à l'adresse addr16.
- Lors de l'exécution de cette instruction :
  1. Le PC est incrémenté de 3 (adresse instruction suivante) puis empilé dans la plie (SP est incrémenté de 2) .
  2. $PC_{15-0} \leftarrow addr16$
- L'instruction RET dépile la donnée (adresse instruction suivante) de la pile dans PC (SP est décrémenté de 2).
- Son code (3 octets) :
  - octet haut = 12H
  - octet milieu = octet haut de addr16
  - octet bas = octet bas de addr16

LCALL : Long CALL [saut dans 64K]

- Exemple : remplacer dans Exemple 1 ACALL par LCALL
- Le code de LCALL sous_programme ; (3 octets) :
  - octet haut = 12H
  - octet milieu = 77H
  - octet bas = 00H
- donc le code est :127700H

Long JUMP [saut dans 64K]

- LJMP addr16 : réalise un saut long absolu inconditionnel vers la position d'adresse addr16.
- Comme LCALL, sauf ici il s'agit d'un saut sans retour (la pile et RET ne sont pas utilisés).
- Lors de l'exécution de cette instruction :
  1. $PC_{15-0} \leftarrow addr16$
- Son code (3 octets) :
  - octet haut = 02H
  - octet milieu = octet haut de addr16
  - octet bas = octet bas de addr16
- Exemple : instruction LJMP debut ; de l'exemple 1
  - octet haut = 02H
  - octet milieu = 75H
  - octet bas = 98H
- donc le code est :027598H

SJMP : Short JUMP [PC-128, PC+127]

- SJMP rel : réalise un saut court relatif au PC (adresse suivante).
- On peut donc sauter dans l'intervalle : [PC-128, PC+127]
- Son code (2 octets) :
  - octet haut = 80H
  - octet bas = octet bas de adr_relative = adr_relative(8bits) = rel8.
- rel = étiquette = adresse absolue appartient à l'espace 64 Ko = adr_absolue.
- adr_relative = adr_absolue - PC (adresse suivante)

SJMP : Short JUMP [PC-128, PC+127]

- Lors de l'exécution de cette instruction (à partir du code 80H et rel8) :
  1. $PC \leftarrow PC + 2$; adresse suivante <— adresse courante +2.
  2. $PC_{15-0} \leftarrow PC_{15-0} + adr\_relative(sur\ 16\ bits)$;

  Avec extension de signe :

$$adr\_relative(sur\ 16\ bits) = [00H\ rel8] si\ rel8 \geq 0$$
$$= [FFH\ rel8]\ si\ rel8 < 0$$

Exemple

```
LOC  OBJ                  LINE     SOURCE

                             1     $mod51  ; This includes 8051 defin
                            Metalink assembler
0000                         2     ORG 0
0000 806E                    3     SJMP etiquette;
0002 00                      4     NOP; instructions
                             5
0070                         6     ORG 70H
0070 00                      7     etiquette: NOP;
0071 80FE                    8     stop: SJMP stop
                             9
                            10
                            11     END
FFA51 MACRO ASSEMBLER  POGRAMME
```

Exemple

- Code de l'instruction SJMP etiquette ; :
  - etiquette_relative = etiquette_absolue - PC (adresse de NOP) = 0070H - 0002H = 006EH (110 en décimal)
  - octet haut = 80H
  - octet bas = octet bas de etiquette_relative = 6EH.
  - donc le code = 806EH
- Code de l'instruction SJMP stop ; :
  - stop_relative = stop_absolue - PC (adresse suivante) = 0071H - 0073H = FFFEH = (-2 en décimal)
  - octet haut = 80H
  - octet bas = octet bas de stop_relative = FEH.
  - donc le code = 80FEH

Instructions de branchement non conditionnels

- JMP : JUMP
  - JMP @A+DPTR : réalise un saut long inconditionnel vers la position d'adresse (A)+(DPTR). (Adressage indirect)
  - Son code = 73H

Syntaxe	Code instruction	Octets	cycles
ACALL adr11	11h+20hxP Addr8	2	2
AJMP addr11	01h+20hxP Addr8	2	2
LCALL addr16	12h addr16	3	2
LJMP addr16	02h addr16	3	2
SJMP rel	80h rel	2	2
JMP @A+DPTR	73h	1	2

Instructions de branchement conditionnels

- rel : déplacement est déterminé de la même façon que SJMP : adr_relative = adr_absolue - PC (adresse suivante) avec adresse suivante= adresse courante +2 ou +3 selon la taille de l'instruction.
- JZ rel : saut si $A = 0$
- JNZ rel : saut si $A \neq 0$
- JC rel : saut si retenue à 1
- JNC rel : saut si retenue à 0
- JB bit,rel : saut si bit à 1
- JNB bit,rel : saut si bit à 0
- JBC bit,rel : saut si le bit est à 1 et mise à zéro de celui-ci
- CJNE octet1, octet2, rel : saut si octet1 et octet2 sont différents
- DJNZ octet, rel : décrémente octet et saut si résultat différent de 0

Instructions de branchement conditionnels

JZ rel	60	2	2
JNZ rel	70	2	2
JC rel	40	2	2
JNC rel	50	2	2
JB bit, rel	20	3	2
JNB bit, rel	30	3	2
JBC bit,rel	10	3	2
CJNE @Ri, #data, rel	B6+i	3	2
CJNE A, #data, rel	B4	3	2
CJNE A, direct, rel	B5	3	2
CJNE Rn, #data, rel	E8+n	3	2
DJNZ direct, rel	D5	3	2
DJNZ Rn, rel	D8+n	2	2

Instruction CJNE

- L'instruction CJNE octet1, octet2, etiquette ; compare 2 octets et réaliser un saut vert l'etiquette si les 2 octets sont différents
- Elle affecte le bit C :
  - C=1 si octet1 <octet2 ; comparaison non-signée
  - C=0 sinon
- Exemple :
  CJNE A,#34H, different ; saut vers différent si A est différent de 34H.
  ... ;
  different : JC inferieur ; saut vers inférieur si A $<$ 34H.
  ; ... instructions ; ici A est $>=$ 34H...
  inferieur : SJMP inferieur ;

Instructions Diverses

- Instructions Diverses
  - RET : Retour de sous programme
  - RETI : Retour d'interruption
  - NOP : No Operation

Syntaxe	CO	Octets	cycles
RET	22h	1	2
RETI	32h	1	2
NOP	00	1	1

Instructions modifiant les drapeaux du PSW

instruction	drapeaux			instruction	drapeaux		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

Type 1 : Lire les pins d'un port

- Type 1 : utilisé si le Port (ou une pin) est configuré en entrée.
- Instruction de branchement conditionnel avec un opérande = "direct " ou "bit" : JB, JNB et CJNE.
- Autres instructions avec le 2ème opérande (opérande de droite) = "direct " ou "bit" : MOV, ADD, ADDC, SUBB, ANL, ORL et XRL.
- Exemple : MOV A,direct —> MOV A,PX

Instruction	Exemple	Explication
MOV A,PX	MOV A,P0	$A \leftarrow$ *état des pins de* $P0$
JNB PX.Y,etiquette	JNB P1.1,etiquette	Saut vers etiquette si la pin P1.1=0
JB PX.Y,etiquette	JB P3.1,etiquette	Saut vers etiquette si la pin P3.1=1
MOV C,PX.Y	MOV C,P2.3	$C \leftarrow$ *pin* $P2.3$

Type 2 : Modifier les Latchs (et les Pins) d'un port

- Type 2 : utilisé si le Port (ou une pin) est configuré en sortie.
- Instruction avec le 1er opérande (opérande de gauche) = "direct " ou "bit" : MOV, INC, DEC, CLR et SETB.
- Exemple : MOV direct,A —> MOV PX,A

Instruction	Exemple	Explication
MOV PX,#data	MOV P0,#0FFH	$P0 \leftarrow FFH$
INC PX	INC P1	$P1 \leftarrow P1 + 1$
DEC PX	DEC P1	$P1 \leftarrow P1 - 1$
MOV PX.Y,C	MOV P2.2,C	$P2.2 \leftarrow C$
CLR PX.Y	CLR P2.2	$P2.2 \leftarrow 0$
SETB PX.Y	SETB P2.2	$P2.2 \leftarrow 1$

Type 3 : Lire et Modifier les Latchs d'un port (et les Pins)

- Type 3 : utilisé si le Port (ou une pin) est configuré en sortie.
- Instruction avec le 1er opérande (opérande de gauche) = "direct " ou "bit" : ANL, ORL, XRL, CPL, JBC et DJNZ.
- Exemple : ANL direct, #data —> ANL P0, #10

Instruction	Exemple	Explication
ANL PX,A	ANL P0,A	$P0 \leftarrow P0\, AND\, A$
ORL PX,A	ORL P0,A	$P0 \leftarrow P0\, OR\, A$
XRL PX,A	XRL P0,A	$P0 \leftarrow P0\, XOR\, A$
CPL PX.Y	CPL P1.2	$P1.2 \leftarrow \overline{P1.2}$
JBC PX.Y,etiquette	JBC P1.2,etiquette	Saut vers etiquette si $P1.2 = 1$, puis $P1.2 \leftarrow \overline{P1.2}$
DJNZ PX,etiquette	DJNZ P2,etiquette	$P2 \leftarrow P2 - 1$, puis Saut vers etiquette si $P2 \neq 0$

Directives de OPTIMA 51

- Assembleur Optima51 : 19 Directives.
- Assembleur µVision3 : vers 43 Directives (voir : help-> sommaire-> Control Statements).
- ORG : Modification du compteur de programme (µVision)
- DSEG : Section données de programme (µVision)
- CSEG : Section code de programme (µVision)
- DS : Réservation de zone mémoire (µVision)
- DZ : (*) Réservation de mémoire avec mise à zéro
- DB : (*) Initialisation d'octets (µVision)
- DW : (*) Initialisation de mots (2 octets) (µVision)
- DC : (*) Initialisation de chaîne de caractères
- CNOP : Alignement du compteur de programme

Directives de OPTIMA 51

- BIT : Définition d'une adresse de bit (µVision)
- EQU : Définition d'un symbole (µVision)
- END : Fin d'assemblage (µVision)
- FILL : (*) Initialisation d'une zone mémoire
- IFD : Assemblage conditionnel (IF-ELSE-ENDIF dans µVision)
- IFND : ELSE : Alternative d'assemblage conditionnel ENDIF : Fin d' assemblage conditionnel
- INCLUDE : Insertion d'un autre fichier
- OPT : Modification d'options d'assemblage
- PAGE : Insertion d'un saut de page
- USING : Définition du numéro de la page des registres (µVision)

(*) : Les directives avec initialisation ne peuvent pas être utilisées dans la section "données" (DSEG).

Directive ORG

- ORG : Réglage de l'adresse de début de stockage d'un bloc d'instructions.
  - Syntaxe : [label :] ORG <adresse>
  - Cette directive permet de spécifier une adresse <adresse> à partir de laquelle les instructions qui suivent vont être stockées dans la ROM.
  - Par défaut, l'assembleur initialise le compteur de programme à la valeur zéro (équivalent à ORG 0).

Directives DSEG

- DSEG : Section données de programme
  - Syntaxe : DSEG
  - Cette directive suspend la génération de code de programme.
  - Le compteur de programme est repositionné à la dernière adresse de la section (par défaut zéro).
  - Cette directive est utile pour définir les adresses externes au programme.
  - On ne peut donc y mettre que des directives ne générant pas de code.

Directive CSEG

- CSEG : Section code de programme, Syntaxe CSEG
  - Cette directive initialise ou reprend la génération de code de programme.
  - Le compteur de programme est repositionné à la dernière adresse de la section (par défaut zéro).

Directive DS

- DS : Réservation d'une zone mémoire
  - Syntaxe : [label :] DS <nbe>
  - Cette directive permet de réserver une zone mémoire de <nbe> octets.
  - Les octets réservés ont une valeur indéfinie au début de l'exécution du programme.
  - Cette directive se traduit en fait par un <ORG> à l'adresse courante plus le nombre d'octets réservés.
- Exemple :
  - Zone1 : DS 100 ; réservation de 100 octets
    ;càd comme ORG Zone1+100

Directive DZ

- DZ : Réservation de mémoire avec mise à zéro (n'est pas définie dans µvision)
- Syntaxe : [label :] DZ <nbre>
- Cette directive réserve <nbre> octets et les initialise tous à zéro.

Directive DB

- DB : Initialisation d'octets, Syntaxe [label :] DB <liste d'octets>
  - Cette directive permet d'initialiser des octets avec différentes valeurs initiales.
  - La liste d'octets est une suite de valeurs ou de chaînes de caractères entre guillemets (simple : ' ou double : ") séparées par des virgules.
  - Exemples :
    ORG 4090H
    adresse1 : DB 10, 30 ; adresse1= 4090H ; (adresse1)=10 ; (adresse1+1)=30 ;
    adresse2 : DB 1, 2, "l'adresse", 0DH, 0AH
    adresse3 : DB 'nom', 0

Directive DW

- DW : Initialisation de mots
- Syntaxe : [label :] DW <liste de mots>
- Cette directive permet d'initialiser des mots (2 octets) avec différentes valeurs initiales.
- La liste de mots est une suite de valeurs séparées par des virgules.
- Exemple :
  - tab : DW 'ab',1,"12"; a=61H et b=62H code ASCII
  - tab : DW "abcdef",1,"12"; erreur plus de 2 octets

Directive DC

- DC : Initialisation de chaîne de caractères (n'est pas définie dans µvision)
  - Syntaxe : [label :] DC "<chaîne de caractères>"
  - Cette directive permet d'initialiser des octets par une chaîne de caractères.
- Exemples :
  - DC "tableau"

Directive BIT

- BIT : Définition d'une adresse de bit, Syntaxe : <label> BIT <expression>
  - < *label* > étiquette équivalente à une adresse d'un bit.
  - < *expression* > peut être :
    - un symbole
    - une adresse d'un octet suivi d'un indice de bit
    - une adresse hexa.
- Exemple :
  - Port3 EQU 0B0H ; adresse du port 3
  - FLAG1 BIT 5FH ; bit d'adresse 5FH dans l'espace ; adressable par bit.
  - FLAG2 BIT 2FH.5 ; bit 5 de l'octet d'adresse 2FH
  - LED BIT Port3.7 ; bit 7 de l'octet d'adresse Port3 = état d'une LED

Directive EQU

- EQU : Définition d'un symbole
- Syntaxe : <label> EQU <expression>
- Le paramètre <expression> est une expression absolue qui sera la valeur du symbole <label>.
- Il ne doit pas comporter de symbole indéfini.
- Exemples :
  - AA EQU 0B0H ; Symbole AA=B0H
  - BB EQU 10+AA ; définition correcte
  - CC EQU AA+ZZ ; erreur 'ZZ' n' est pas encore défini
  - avec définition de ZZ : ZZ EQU 1AH ;on corrige l'erreur.

Directive END

- END : Fin d'assemblage
- Syntaxe : [label :] END
- Cette directive indique à l' assembleur la fin du texte à assembler.

Programme (Fichier.LST)

```
LOC  OBJ                  LINE     SOURCE

                             1     $mod51  ; This includes 8051
                            Metalink assembler
                             2
0000                         3     org 0
0000 E4                      4           CLR A;
0001 7864                    5           MOV R0,#100;
0003 28                      6     boucle: ADD A, R0;
0004 D8FD                    7           DJNZ R0,boucle;
                             8
                             9     ;stop: sjmp stop
                            10
                            11     end
FFA51 MACRO ASSEMBLER  POGRAMME
```

Programme (Fichier.HEX)

- Contenu :
  :06000000E4786428D8FD3D
  :00000001FF
- Explication :

Nbre	Adresse	Type	Données	CS
:06	0000	00	E4786428D8FD	3D
:00	0000	01		FF

Nbre = Nbre d'octets de données dans la ligne
Adresse = adresse du premier octet de données de la ligne
Type = 00 pour "il reste des lignes après" ou 01 pour "la dernière ligne"
Données = instructions ou données du code machine
CS = Check Sum = Somme de touts les octets + CS=00H

Taille du code machine

- Taille du code machine = 6 octets

Contenu de la ROM	
Adresse	Contenu
0006H	
0005H	FD
0004H	D8
0003H	28
0002H	64
0001H	78
0000H	E4

Temps d'exécution du programme

- Programme :

```
org 0
CLR A ; 1 CM
MOV R0,#100 ; 1 CM
boucle : ADD A, R0 ; 1 CM
DJNZ R0,boucle ; 2 CM
END
```

- Nbre de cycles = 1 (CLR) + 1 (MOV) + boucle = 302
- avec : boucle = 100 (R0) * (1 (ADD) + 2 (DJNZ))= 300
- T=Temps d'exécution = Nbre de cycles * CM = 302 * CM
  - pour Fh=12 MHz , 1 CM= 12 * Th = 12 /12 µs =1 µs ; donc T=302 µs
  - pour Fh=4 MHz , 1 CM= 12 * Th = 12 /4 µs =3 µs ; donc T=906 µs