ÉLECTRONIQUE

8051 : jeu d'instruction

Modes d'adressage

ADD A,<byte>

	code
▸ registre	
♦ <byte> = nom d'un registre	
♦ ADD A,R0 et R0 = 55H	28H
♦ A ← A + R0 = A + 55H	
▸ immédiat (constante immédiate)	
♦ <byte> est une constante de 1 byte	
♦ ADD A,#4AH	24H
♦ A ← A + 4AH	4AH
▸ direct	
♦ <byte> = adresse d'un byte à accès direct	
♦ ADD A,4AH et (4AH) = 9EH	25H
♦ A ← A + (4AH) = A + 9EH	4AH
▸ indirect	
♦ <byte> = @R0 ou <byte> = @R1	
♦ ADD A,@R0 si R0 = 8EH et (8EH) = BAH	26H
♦ A ← A + (8EH) = A + BAH	

(..)= contenu de l'adresse ..
A = contenu de A
Ri= contenu du reg. Ri

Une des notions importantes du jeu d'instruction d'un processeur est la notion de mode d'adressage. Une même instruction peut ainsi être déclinée en plusieurs variantes, suivant la manière de rechercher la ou les opérandes.
Illustrons cette notion à l'aide de ADD A l'instruction arithmétique la plus simple, qui consiste à additionner un octet <byte> à l'accumulateur et à replacer le résultat dans l'accumulateur. L'opérande <byte> peut avoir plusieurs provenances.

1°) Un registre
C'est l'instruction la plus compacte car l'octet d'instruction contient le numéro du registre; pour obtenir l'opérande, il n'y a donc aucune lecture supplémentaire à faire en mémoire .

2°) Une constante
Une opérande constante peut être stockée en mémoire morte; elle est placée immédiatement après l'octet d'instruction dans la mémoire programme, d'où le nom d'***adressage immédiat***. Il y a donc une lecture en mémoire programme à faire pour connaître l'opérande.

3°) Un octet situé à une adresse connue (variable statique) dans la mémoire DATA ou un des SFR.
Pour obtenir l'opérande, il y a donc une lecture à faire en mémoire programme pour connaître l'adresse, plus une lecture en mémoire DATA à cette adresse. On parle d'***adressage direct***.

4°) Un octet à une adresse contenue dans un registre d'adresse (adressage indirect) qui constitue un pointeur vers cet octet. R0 et R1 peuvent servir de registre d'adresse. On parle d'***adressage indirect***.
Comme pour l'adressage en registre, tout est contenu dans l'instruction, aucune opérande n'est nécessaire.

Dans le 8051, le temps d'exécution de ces 4 instructions est identique, soit un cycle machine. Toutefois, l'adressage indirect coûte un cycle supplémentaire pour charger le registre R0 ou R1

Transfert de données internes

Mnémonique	Opération	Modes adress. dir	ind	reg	imm	Cycles
MOV A,<src>	A = <src>	X	X	X	X	1
MOV <dest>,A	<dest> = A	X	X	X		1
MOV <dest>,<src>	<dest> = <src>	X	X	X	X	2
MOV DPTR,#data16	DPTR = 16-bit immediate constant				X	2
PUSH <src>	INC SP:MOV"@SP",<src>	X				2
POP <dest>	MOV <dest>,"@SP":DEC SP	X				2
XCH A,<byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A,@Ri	ACC and @Ri exchange low nibbles			X		1

Le transfert de données interne se fait par l'instruction MOV pour (MOVe)

MOV dispose de tous les modes d'adressage. Si l'accumulateur est impliqué, 1 seul cycle est nécessaire, sinon MOV prend 2 cycles.

MOV DPTR n'a de sens qu'en constante immédiate puisqu'il s'agit de charger le pointeur de données pour un échange avec la mémoire externe. (Voir MOVX à la dia suivante)

PUSH et POP permettent de sauvegarder un octet sur la pile, puis de l'en retirer.

XCH est une instruction de permutation de données

XCHD est particulière à l'arithmétique BCD (voir manuel 8051 pour plus de détails)

Transferts avec RAM externe XDATA

MOVX = **MOV**e e**X**ternal

Mnémonique	Opération	Bits d'adresse	Cycles
MOVX A,@Ri	Read external RAM @Ri	(P2)+8 bits	2
MOVX @Ri,A	Write external RAM @Ri	(P2)+8 bits	2
MOVX A,@DPTR	Read external RAM @DPTR	16 bits	2
MOVX @DPTR,A	Write external RAM @DPTR	16 bits	2

```
MOV  P2,#3      ; choix de la page
MOV  R0,#F2H    ; offset
MOVX A,@R0      ; transfert
```

```
MOV  DPTR,#0EF2H ; adresse
MOVX A,@DPTR     ; transfert
```

- restrictions
  - uniquement de (vers) A
  - R0 et R1 seulement

Si l'on veut transférer un octet de (vers) la mémoire XDATA (mémoire vive externe) on ne peut le faire que par adressage indirect.

Pour un adressage en mémoire paginée, il faut
- précharger P2 avec le N° de la page — MOV P2,#N°_PAGE (1 cycle)
- précharger R0 ou R1 avec l'offset dans la page — MOV R0, #offset (1 cycle)
- faire un transfert indirect dans l'accumulateur — MOVX A,@R0 (2 cycles)

TOTAL : 4 cycles pour le premier transfert et 3 cycles tant que l'on reste dans la même page

Pour un adressage sur 16 bits , il faut
- précharger DPTR avec l'adresse — MOV DPTR,#addr (2 cycles)
- faire un transfert indirect dans l'accumulateur — MOVX A,@DPTR (2 cycles)

TOTAL : 4 cycles

Remarquons que :
- on ne peut faire de transfert que depuis (ou vers) l'accumulateur A
- seuls R0, R1 et DPTR peuvent servir de registre d'adresse

"Lookup tables" en ROM

- utilités : tables de constantes
  - sinus, forme arbitraire, calibration
  - en mémoire programme (256 bytes max)

adresse absolue

```
        MOV   DPTR,TBL
        MOV   A,#54
        MOVC  @A+DPTR

TBL:    table[0]
        table[1]
```

≠0

```
MOV  A,#54      ; 54=indice
CALL TBL            (PC)=TBL+1
.....
TBL: MOVC  @A+PC
     RET = table[0]
     table[1]
     table[2]
```

MOVC = MOVe Constant

Mnémonique	Opération	Mode adress. indirect.	Cycles
MOVC A,@A+DPTR	Read prog. mem. at (A + DPTR)	indexé + offset	2
MOVC A,@A+PC	Read prog. mem. at (A + PC)	relatif au PC	2

L'instruction MOVC (MOVe Constant) est unidirectionnelle de la mémoire programme vers l'accumulateur.

Elle permet de lire une table de constante de 256 valeurs de 1 octet (table de sinus, valeurs de calibration, fonction non-linéaire,...)

Si la table peut être placée à une adresse absolue, on précharge DPTR avec l'adresse du début de la table en mémoire programme et l'accumulateur A avec l'indice du tableau (d'où le nom d'***adressage indexé***). Le résultat est placé dans A

Si l'on veut pouvoir placer la mémoire programme à n'importe quelle addresse, on se sert du compteur de programme comme pointeur vers le début de la table (d'où le nom d'***adressage relatif au PC***)

- on charge l'accumulateur avec l'indice du tableau (la valeur 0 est interdite)
- on fait un CALL à l'adresse du début de la table
- la première instruction est un MOV @A,PC; comme PC est automatiquement post-incrémenté, il pointe vers l'octet qui suit, qui est l'instruction RET qui rendra la main au programme principal; si A vaut au moins 1, on va chercher l'octet situé à l'adresse PC+A.

Opérations arithmétiques

Mnémonique	Opération	Modes adress. dir	ind	reg	imm	Cycles
ADD A,<byte>	A = A + <byte>	X	X	X	X	1
ADDC A,<byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A,<byte>	A = A - <byte> - C	X	X	X	X	1
INC A	A = A + 1	Accumulator only				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A = A - 1	Accumulator only				1
DEC <byte>	<byte> = <byte> - 1	X	X	X		1
MUL AB	B:A = B x A	ACC and B only				4
DIV AB	A = Int[A/B] B = Mod[A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

Les instructions arithmétiques comprennent classiquement l'addition et la soustraction, avec report "CARRY" et emprunt "BORROW".

Le 8051 était, à l'époque de sa création, un des seuls processeurs 8 bits offrant une multiplication et une division non signées rapides (4 cycles).

L'incrémentation du pointeur de données DPTR est particulièrement utile pour adresser des octets consécutifs en mémoire vive externe XDATA.

L'instruction DA est destinée à ajuster le résultat d'une addition de 2 octets contenant du BCD (binary coded decimal) c'est-à-dire considérés comme 2 groupes de 4 bits (nibbles) codant chacun un chiffre de 0 à 9 (et non de 0 à F). Les règles de report sont évidemment différentes.

Opérations logiques

Mnémonique	Opération	Modes adress. dir	ind	reg	imm	Cycles
ANL A,<byte>	A = A AND <byte>	X	X	X	X	1
ANL <byte>,A	<byte> = <byte> AND A	X				1
ANL <byte>,#data	<byte> = <byte> AND #data	X				2
ORL A,<byte>	A = A OR <byte>	X	X	X	X	1
ORL <byte>,A	<byte> = <byte> OR A	X				1
ORL <byte>,#data	<byte> = <byte> OR #data	X				2
XRL A,<byte>	A = A XOR <byte>	X	X	X	X	1
XRL <byte>,A	<byte> = <byte> XOR A	X				1
XRL <byte>,#data	<byte> = <byte> XOR #data	X				2
CLR A	A = 00H	Accumulator only				1
CPL A	A = NOT A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

Les opérations logiques sont le AND, le OR, le XOR et portent sur l'accumulateur (en 1 cycle) ou sur un octet de mémoire (en 2 cycles) ;

La fonction logique s'applique bit-à-bit, entre bits de même poids des deux opérandes.

Le NOT (appelé CPL pour ComPLement) ne porte que sur A.

Remarquons l'absence d'instructions de type SHIFT (décalage), seul le ROTATE existe.

SWAP est de nouveau lié à l'arithmétique BCD et permute les deux nibbles de l'accumulateur.

Manipulation de bit

Mnémonique	Opération	Modes adress.	Cycles
ANL C,bit	C = C AND bit	direct bit	2
ANL C,/bit	C = C AND (NOT bit)		2
ORL C,bit	C = C OR bit		2
ORL C,/bit	C = C OR (NOT bit)		2
MOV C,bit	C = bit		1
MOV bit,C	bit = C		2
CLR C	C = 0		1
CLR bit	bit = 0		1
SETB C	C = 1		1
SETB bit	bit = 1		1
CPL C	C = NOT C		1
CPL bit	bit = NOT bit		1
JC rel	Jump if C = 1		2
JNC rel	Jump if C = 0		2
JB bit,rel	Jump if bit = 1		2
JNB bit,rel	Jump if bit = 0		2
JBC bit,rel	Jump if bit = 1;CLR bit		2

L'originalité du 8051 est la présence d'instructions spécifiques sur des bits, introduisant donc un nouveau mode d'adressage assimilable à l'adressage direct, mais visant un seul bit, via son adresse de bit (voir chapitre sur la mémoire du 8051).

Dans ces opérations, le Carry, qui est un des bits du registre de statut, joue le rôle de l'accumulateur. Paradoxalement, les opérations logiques AND et OR sur les bits prennent 2 cycles, alors qu'elles n'en prennent qu'un sur des octets !

Les instructions de branchement relatif conditionnels peuvent tester soit le Carry C soit n'importe quel booléen adressable par bit.

Branchement inconditionnel

Mnémonique	Opération	Modes adress.	Cycles
JMP addr	Jump to addr		2
JMP @A+DPTR	Jump to A + DPTR		2
CALL addr	Call subroutine at addr		2
RET	Return from subroutine		2
RETI	Return from interrupt		2
NOP	No operation		1

JMP est utilisé pour une rupture inconditionnelle du déroulement du programme; c'est l'instruction type que l'on placera dans un vecteur d'interruption.

CALL est l'appel d'une fonction (appelée aussi sous-routine)
RET termine une fonction et RETourne au programme principal, juste après le CALL

RETI (RETurn from INTERRUPT) termine une routine de service d'interruption. Elle est différente de RET car l'unité de contrôle doit effectuer une autre séquence d'opération qu'au retour d'une routine nomale.

NOP ne fait rien et sert en général de bourrage pour ajuster la durée d'un bout de code.

Branchement conditionnel

Mnémonique	Opération	Modes adress. dir	ind	reg	imm	Cycles
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump if A ≠ 0	Accumulator only				2
DJNZ <byte>,rel	Decr.and jump if not zero	X		X		2
CJNE A,<byte>,rel	Jump if A ≠ <byte>	X			X	2
CJNE <byte>,#data,rel	Jump if <byte> ≠ #data		X	X		2

Les instructions de banchement conditionnel permettent de réaliser les instructions de haut niveau de type

if
switch
while
for

pour tester les conditions d'exécution ou non d'un segment de code ou d'une boucle.