Hiệu đính từ slide của thầy Hồ Trung Mỹ
Bộ môn Điện tử - DH BK TPHCM

CHƯƠNG 3

HỌ VI ĐIỀU KHIỂN 8051

3.4 Tập lệnh 8051

Nội dung

3.4.1 Các lệnh số học

3.4.2 Các lệnh luận lý

3.4.3 Các lệnh chuyển dữ liệu

3.4.4 Các lệnh với biến Boole

3.4.5 Các lệnh rẽ nhánh chương trình

3.4.6 Một số thí dụ

Ý nghĩa các ký hiệu viết tắt trong tập lệnh

Ký hiệu	Ý nghĩa
Rn hay Rr	Định địa chỉ thanh ghi bằng các thanh ghi R0 đến R7
direct	Địa chỉ nội 8 bit (00H–FFH)
@Ri	Định địa chỉ gián tiếp dùng thanh ghi R0 hoặc R1
#data8	Hằng số 8 bit ở ngay trong lệnh. Chú ý với số Hex nếu bắt đầu bằng chữ thì phải thêm số 0 ở phía trước. Thí dụ: – Hằng số thập phân như #10, #-12 – Hằng số nhị phân như #01011010B – Hằng số thập lục phân (số Hex) như #1FH, #0A6H
#data16	Hằng số 16 bit ở ngay trong lệnh. Thí dụ: #0FF25H
bit	Địa chỉ trực tiếp (8 bit) của bit
byte	Một trong các kiểu: direct, @Ri, Rn hoặc #data8
rel	Offset 8 bit có dấu trong định địa chỉ tương đối
addr11	Địa chỉ tuyệt đối 11 bit trong trang 2K hiện hành
addr16	Địa chỉ 16 bit
src	Toán hạng nguồn (source) có thể là Rn, direct hoặc @Ri
dest	Toán hạng đích (destination) có thể là Rn, direct hoặc @Ri
(X)	Nội dung của X. Thí dụ: (A) nội dung thanh ghi A.
((X))	Nội dung ô nhớ có địa chỉ là nội dung của X.
DIR	Cách định địa chỉ trực tiếp (direct)
IN	Cách định địa chỉ gián tiếp (indirect)
REG	Cách định địa chỉ thanh ghi (register)
IMM	Cách định địa chỉ tức thời (Immediate)

Các lệnh ảnh hưởng đến thanh ghi trạng thái PSW (CY, OV, AC)

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

1. Tóm tắt các lệnh số học
(giả sử 8051 với thạch anh 12 MHz)

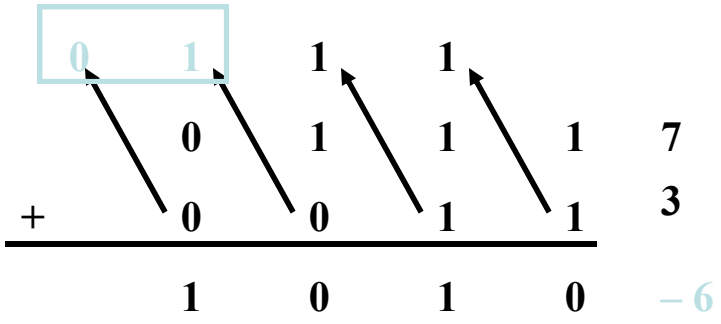
MNEMONIC (Từ gợi nhớ)	OPERATION (Phép toán/Tác vụ)	ADDRESSING MODES (Các cách định địa chỉ)				EXECUTION TIME (μs) (Thời gian thực thi)
		DIR	IND	REG	IMM	
ADD A, byte	(A) = (A) + (byte)	X	X	X	X	1
ADDC A, byte	(A) = (A) + (byte) + (C)	X	X	X	X	1
SUBB A, byte	(A) = (A) – (byte) – (C)	X	X	X	X	1
INC A	(A) = (A) + 1	Chỉ với thanh ghi tích lũy ACC				1
INC byte	(byte) = (byte) + 1	X	X	X		1
INC DPTR	(DPTR) = (DPTR) + 1	Chỉ với DPTR				2
DEC A	(A) = (A) – 1	Chỉ với thanh ghi tích lũy ACC				1
DEC byte	(byte) = (byte) – 1	X	X	X		1
MUL AB	(B:A) = (B) x (A)	Chỉ với ACC và B				4
DIV AB	(A) = Int[(A)/(B)] (B) = Mod[(A)/(B)]	Chỉ với ACC và B				4
DA A	Điều chỉnh thập phân	Chỉ với thanh ghi tích lũy ACC				1

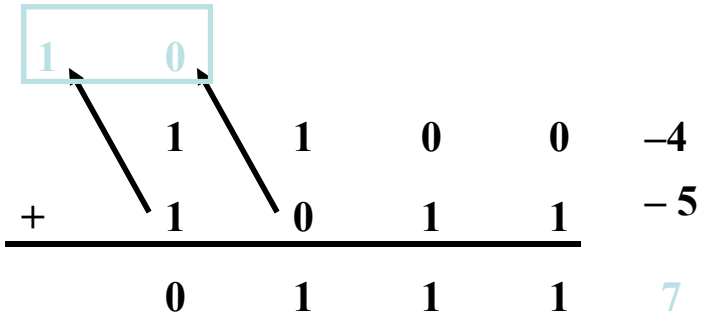
Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive
- Consider the operations A + B, and A – B
  - Can overflow occur if B is 0 ?
  - Can overflow occur if A is 0 ?

Overflow Detection

- Overflow: the result is too large (or too small) to represent properly
  - Example: - 8 < = 4-bit binary number <= 7
- When adding operands with different signs, overflow cannot occur!
- Overflow occurs when adding:
  - 2 positive numbers and the sum is negative
  - 2 negative numbers and the sum is positive
- On your own: Prove you can detect overflow by:
  - Carry into MSB ° Carry out of MSB





Các lệnh ADD

```
add a, byte          ; a ← a + byte
addc a, byte         ; a ← a + byte + C
```

các lệnh này ảnh hưởng 3 bit trong PSW:

C = 1 nếu kết quả cộng > FF

AC = 1 nếu có nhớ tại bit 3

OV = 1 nếu có nhớ từ bit 7 mà không từ bit 6 hoặc ngược lại.

Program Status Word (PSW)

Bit	7	6	5	4	3	2	1	0
Flag	CY	AC	F0	RS1	RS0	OV	F1	P
Name	Carry Flag	Auxiliary Carry Flag	User Flag 0	Register Bank Select 1	Register Bank Select 0	Overflow flag	User Flag 1	Parity Bit

Lệnh ADD và SUBB

ADD A, Source ;A=A+SOURCE

```
ADD A,#6          ;A=A+6

ADD A,R6          ;A=A+R6

ADD     A,6       ;A=A+[6]  or  A=A+R6

ADD     A,0F3H    ;A=A+[0F3H]
```

SUBB A, Source ;A=A-SOURCE-C

```
SUBB    A,#6      ;A=A-6-(CY)

SUBB    A,R6      ;A=A-R6-(CY)
```

Subtract

SUBB A, byte	subtract with borrow

Example:

```
SUBB A, #0x4F    ;A ← A – 4F – C
```

Notice that
There is no subtraction WITHOUT borrow.
Therefore, if a subtraction without borrow is desired,
it is necessary to clear the C flag.

Example:

```
Clr  c
SUBB A, #0x4F     ;A ← A – 4F
```

Thí dụ với ADD

```
mov a, #3FH
add a, #0D3H
```

```
  0011 1111
  1101 0011
-------------
  0001 0010
```

C = 1

AC = 1

OV = 0

- Cho biết các giá trị của các cờ C, AC và OV sau khi lệnh thứ hai được thực thi?

Cộng có dấu và tràn (OV)

2's complement:

0000 0000	00	0
…		
0111 1111	7F	127
1000 0000	80	-128
…		
1111 1111	FF	-1

```
 0111 1111  (positive 127)
 0111 0011  (positive 115)
 1111 0010  (overflow
cannot represent 242 in 8
bits 2's complement)
```

```
 1000 1111  (negative 113)
 1101 0011  (negative  45)
 0110 0010  (overflow)
```

```
0011 1111  (positive)
1101 0011  (negative)
0001 0010 (never overflows)
```

Decimal Adjust

```
DA a          ; decimal adjust a
```

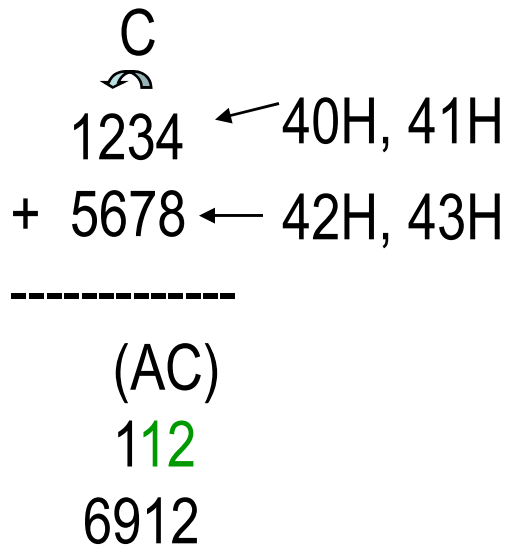
Used to facilitate BCD addition.
Adds “6” to either high or low nibble after an addition to create a valid BCD number.

Example:

```
      mov a, #23h
      mov b, #29h
      add a, b          ; a ← 23h + 29h = 4Ch (wanted
  52)
      DA a              ; a ← a + 6 = 52
```

Thí dụ: Cộng 2 số BCD (mỗi số có 4 ký số)

```
MOV     A, 43H      ; num1 ở các ô nhớ 40, 41H
ADD     A, 41H      ; num2 ở các ô nhớ 42, 43H
DA      A           ; kết quả đặt ở các ô nhớ 40, 41H
MOV     41H, A
MOV     A, 42H
ADDC    A, 40H
DA      A
MOV     40H,A
```



Tính Z = X + Y

Thí dụ: Tính Z = X + Y với Z, X, Y là số 1 byte trong RAM nội.
Giả sử X được cất ở 40H, Y ở 41H, và Z ở 42H.

Cách 1:

```
        MOV   A, 40h
        ADD   A, 41h
        MOV   42h, A
```

Cách 2:

```
X       EQU   40h
Y       EQU   41h
Z       EQU   42h
        MOV   A, X
        ADD   A, Y
        MOV   Z, A
```

Cộng 2 số 16 bit

Thí dụ: Cộng 2 số 16 bit VarX và VarY (có địa chỉ là RAM nội). Cất kết quả vào VarX.

```
; Các số 16 bit cất ở VarX và VarX+1, VarY và VarY+1
   MOV    A, VarX          ; lấy byte thấp
   ADD    A, VarY          ; cộng các byte thấp
   MOV    VarX, A          ; cất byte thấp
   MOV    A, VarX+1        ; lấy byte cao
   ADDC   A, VarY+1        ; cộng có nhớ của phép
   cộng trước
   MOV    VarX+1, A        ; cất kết quả
```

Increment and Decrement

INC A	increment A
INC byte	increment byte in memory
INC DPTR	increment data pointer
DEC A	decrement accumulator
DEC byte	decrement byte

- The increment and decrement instructions do NOT affect the C flag.
- Notice we can only INCREMENT the data pointer, not decrement.

Lệnh INC và DEC

DEC byte ;byte=byte-1

INC byte ;byte=byte+1

```
INC     R7
DEC     A
DEC     40H     ; [40]=[40]-1
```

Lệnh DEC

Chú ý với DPTR chỉ có lệnh INC còn lệnh DEC thì không có. Do đó muốn thực hiện việc giảm DPTR đi 1 thì ta phải sử dụng chuỗi lệnh sau

```
        DEC     DPL                     ; Giảm byte thấp của DPTR đi 1
        MOV     R7, DPL                 ; chép vào R7
        CJNE    R7, #0FFH, SKIP ; Nếu tràn dưới thành FF thì phải mượn
        DEC     DPH             ; do đó cũng phải giảm byte cao đi 1
SKIP:   (tiếp tục)
```

Ta phải giảm đi 1 riêng cho các byte cao và byte thấp của DPTR; tuy nhiên byte cao (DPH) chỉ bị giảm nếu byte thấp (DPL) tràn dưới từ 00H sang FFH.

Example: Increment 16-bit Word

- Assume 16-bit word in R3:R2

```
mov a, r2
add a, #1        ; use add rather than increment to affect C
mov r2, a
mov a, r3
addc a, #0       ; add C to most significant byte
mov r3, a
```

Lệnh MUL & DIV

- **MUL AB** ;B|A = A*B

```
MOV     A,#25H
MOV     B,#65H
MUL     AB              ;25H*65H=0E99
                        ;B=0EH, A=99H
```

- **DIV AB** ;A = A/B, B = A mod B

```
MOV     A,#25
MOV     B,#10
DIV AB          ;A=2, B=5
```

OV - used to indicate a divide by zero condition.
C – set to zero

2. Tóm tắt các lệnh logic
(giả sử 8051 với thạch anh 12 MHz)

MNEMONIC (Từ gợi nhớ)	OPERATION (Phép toán/Tác vụ)	ADDRESSING MODES (Các cách định địa chỉ)				EXECUTION TIME (μs) (Thời gian thực thi)
		DIR	IND	REG	IMM	
ANL A, byte	(A) = (A) AND (byte)	X	X	X	X	1
ANL byte,A	(byte) = (byte) AND (A)	X				1
ANL byte, #data8	(byte) = (byte) AND #data8	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,#data8	(byte) = (byte) OR #data8	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,#data8	(byte) = (byte) XOR #data8	X				2
CLR A	(A) = 00H	Chỉ với thanh ghi tích lũy ACC				1
CPL A	(A) = NOT(A) (lấy bù 1 của A)	Chỉ với thanh ghi tích lũy ACC				1
RL A	Xoay ACC qua trái 1 bit	Chỉ với thanh ghi tích lũy ACC				1
RLC A	Xoay trái qua cờ nhớ	Chỉ với thanh ghi tích lũy ACC				1
RR A	Xoay ACC qua phải 1 bit	Chỉ với thanh ghi tích lũy ACC				1
RRC A	Xoay phải qua cờ nhớ	Chỉ với thanh ghi tích lũy ACC				1
SWAP A	Hoán đổi 2 nửa byte trong A	Chỉ với thanh ghi tích lũy ACC				1

Logic Instructions

- Bitwise logic operations
  - (AND, OR, XOR, NOT)
- Clear
- Rotate
- Swap

Logic instructions do NOT affect the flags in PSW

Bitwise Logic

ANL → AND

ORL → OR

XRL → XOR

CPL → Complement

Examples:

```
  00001111
ANL 10101100
    --------
  00001100

  00001111
ORL 10101100
    --------
  10101111

  00001111
XRL 10101100
    --------
  10100011

CPL 10101100
    --------
  01010011
```

Address Modes with Logic

ANL – AND

ORL – OR

XRL – eXclusive oR

a, byte

direct, reg. indirect, reg, immediate

byte, a

direct

byte, #constant

CPL – Complement

a ex: cpl a

Uses of Logic Instructions

- Force individual bits low, without affecting other bits.

```
anl PSW, #0xE7              ;PSW AND 11100111
```

- Force individual bits high.

```
orl PSW, #0x18              ;PSW OR 00011000
```

- Complement individual bits

```
xrl P1, #0x40               ;P1 XRL 01000000
```

The 8051 Instruction Set

EXAMPLE

Determine the contents of the accumulator after the execution of the following program segments.

```
MOV A, #3CH   ; A ←0011 1100
MOV R4, #66H  ; R4 ←0110 0110
ANL A, R4     ; A ←A AND R4
```

Solution:

A = 0011 1100

R4 = 0110 0110

A AND R4 = 0010 0100 = 24H

Other Logic Instructions

CLR - clear

RL - rotate left

RLC - rotate left through Carry

RR - rotate right

RRC - rotate right through Carry

SWAP - swap accumulator nibbles

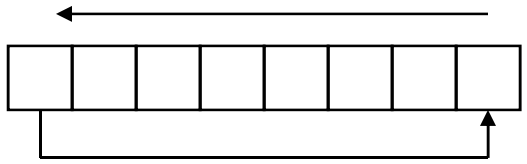
CLR (Set all bits to 0)

```
CLR A
CLR byte        (direct mode)
CLR Ri          (register mode)
CLR @Ri         (register indirect mode)
```

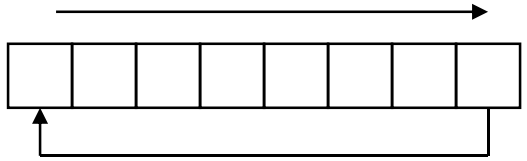
Rotate

- Rotate instructions operate only on **a**

RL a



```
Mov a,#0xF0   ; a← 11110000
RR a          ; a← 11100001
```

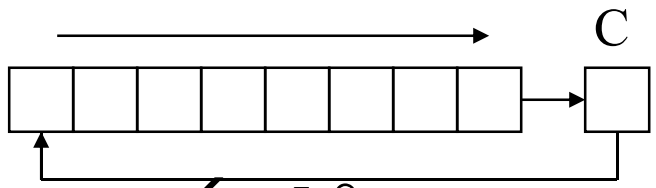


RR a

```
Mov a,#0xF0   ; a← 11110000
RR a          ; a← 01111000
```

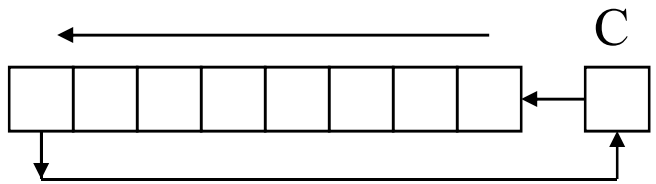
Rotate through Carry

RRC a




```
mov a, #0A9h     ; a ← A9
add a, #14h              ; a ← BD (10111101),
  C←0
rrc a            ; a ← 01011110, C←1
```

RLC a




```
mov a, #3ch              ; a ← 3ch(00111100)
setb c            ; c ← 1
rlc a            ; a ← 01111001, C←1
```

The 8051 Instruction Set

EXAMPLE

Determine the contents of the accumulator after the execution of the following program segments.

```
MOV A, #0C3H ; A ←1100 0011
RLC A        ; Rotate left through carry
```

Solution:

Assume carry = 0 initially

A = 1100 0011

Rotate left through carry

A = 1000 0110, carry = 1

Rotate and Multiplication/Division

- Note that a shift left is the same as multiplying by 2, shift right is divide by 2

```
mov a, #3   ; A← 00000011 (3)
clr C       ; C← 0
rlc a       ; A← 00000110 (6)
rlc a       ; A← 00001100 (12)
rrc a       ; A← 00000110 (6)
```

Swap

SWAP a

```
mov a, #72h      ; a ← 27h
swap a           ; a ← 27h
```

3. Tóm tắt các lệnh chuyển dữ liệu

Bảng 3.10 Tóm tắt các lệnh chuyển dữ liệu truy cập vùng nhớ dữ liệu nội (giả sử 8051 sử dụng thạch anh 12 MHz)

MNEMONIC (Từ gợi nhớ)	OPERATION (Phép toán/Tác vụ)	ADDRESSING MODES (Các cách định địa chỉ)				EXECUTION TIME (µs)
		DIR	IND	REG	IMM	(Thời gian thực thi)
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)= hằng số dữ liệu 16 bit				X	2
PUSH direct	INC SP: MOV "@SP", direct	X				2
POP direct	MOV direct,"@SP": DEC SP	X				2
XCH A, byte	Hoán đổi (A) và (byte)	X	X	X		1
XCHD A, @Ri	Hoán đổi 2 nửa byte thấp của (A) và ((Ri))		X			1

Bảng 3.11 Tóm tắt các lệnh chuyển dữ liệu truy cập vùng nhớ dữ liệu ngoài (giả sử 8051 sử dụng thạch anh 12 MHz)

ADDRESS WIDTH (Độ rộng địa chỉ)	MNEMONIC (Từ gợi nhớ)	OPERATION (Phép toán/Tác vụ)	EXECUTION TIME (µs) (Thời gian thực thi)
8 bits	MOVX A,@Ri	Đọc RAM ngoài gián tiếp qua Ri	2
8 bits	MOVX @Ri,A	Đọc RAM ngoài gián tiếp qua Ri	2
16 bits	MOVX A,@DPTR	Đọc RAM ngoài qua DPTR	2
16 bits	MOVX @DPTR,A	Ghi RAM ngoài qua DPTR	2

Bảng 3.12 Tóm tắt các lệnh tra bảng (giả sử 8051 sử dụng thạch anh 12 MHz)

MNEMONIC (Từ gợi nhớ)	OPERATION (Phép toán/Tác vụ)	EXECUTION TIME (µs) (Thời gian thực thi)
MOVC A,@A+DPTR	Đọc bộ nhớ tại ((A) + (DPTR))	2
MOVC A,@A+PC	Đọc bộ nhớ tại ((A) + (PC))	2

Data Transfer

- MOV A,source ;move source
- MOV A,#data ;to destination
- MOV dest,A
- MOV dest,#data (2)
- MOV DPTR,#data16 (2)
- MOVC A,@A+DPTR (2)
- MOVC A,@A+PC (2)

Data Transfer

- MOVX A,@Ri ;move from
- MOVX A,@DPTR ;data memory
- MOVX @Ri,A
- MOVX @DPTR,A
- PUSH direct
- POP direct
- XCH A,source ;exchange bytes
- XCHD A,@Ri ;exchange low-order digits

What are A & B after prog?

```
MOV     0F0H,#12H       ;B=12H
MOV     R0,#0F0H        ;
MOV     A,#34H          ;A=34H
XCH     A,0F0H          ;A=12H, B=34H
XCHD    A,@R0           ;A=14H, B=32H
```

Read 10F4H & 10F5H into R6,R7

```
MOV     DPTR,#10F4H
MOVX    A,@DPTR
MOV     R6,A
INC     DPTR
MOVX    A,@DPTR
MOV     R7,A
```

Thí dụ: Dịch số BCD (10 ký số BCD) chứa ở các ô nhớ 2AH, 2BH, 2CH, 2DH, 2EH sang phải 2 ký số BCD (giả sử số BCD ban đầu là 0012345678).

– Dùng các lệnh MOV trực tiếp

```
MOV     A,2EH
MOV     2EH,2DH
MOV     2DH,2CH
MOV     2CH,2BH
MOV     2BH,#0
```

2A	2B	2C	2D	2E	ACC
00	12	34	56	78	78
00	12	34	56	56	78
00	12	34	34	56	78
00	12	12	34	56	78
00	00	12	34	56	78

Chuỗi lệnh trên chiếm 14 byte và thực thi trong 9 μs.

– Dùng các lệnh XCH

```
CLR     A
XCH     A,2BH
XCH     A,2CH
XCH     A,2DH
XCH     A,2EH
```

2A	2B	2C	2D	2E	ACC
00	12	34	56	78	00
00	00	34	56	78	12
00	00	12	56	78	34
00	00	12	34	78	56
00	00	12	34	56	78

Chuỗi lệnh trên chiếm 9 byte và thực thi trong 5 μs

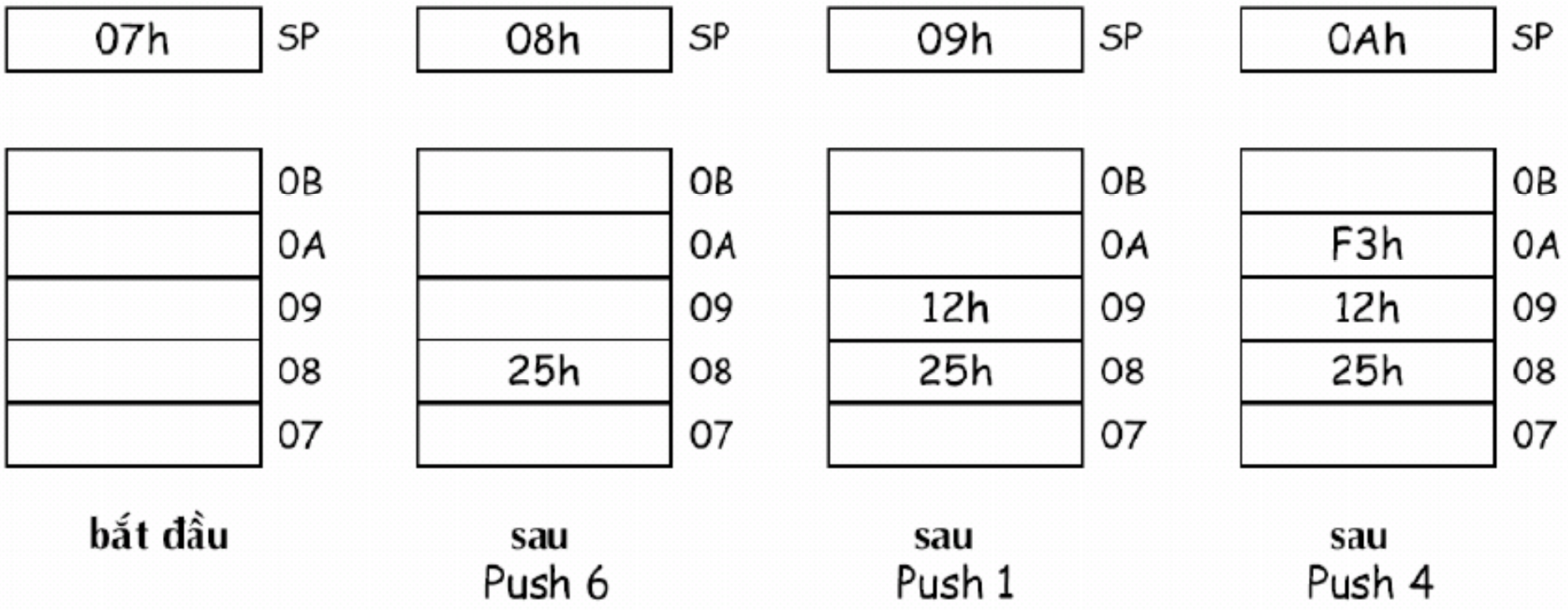
Minh họa về tác động của PUSH và POP

- Sau khi reset hệ thống thì SP=07H.
- Có 3 cách để truy cập: PUSH (cất vào ngăn xếp), POP (lấy ra khỏi ngăn xếp) và CALL (chuyển điều khiển chương trình và quay về [RET, RETI]).
- Để cất thông tin vào ngăn xếp, ta sử dụng lệnh PUSH. Khi đó các bước sau xảy ra
    1. SP được tăng thêm 1.
    2. Thông tin được đặt vào địa chỉ được trỏ bởi SP.
- Để lấy thông tin ra khỏi ngăn xếp, ta sử dụng lệnh POP. Khi đó các bước sau xảy ra
    1. SP được giảm đi 1.
    2. Thông tin có địa chỉ được trỏ bởi SP được lấy ra.

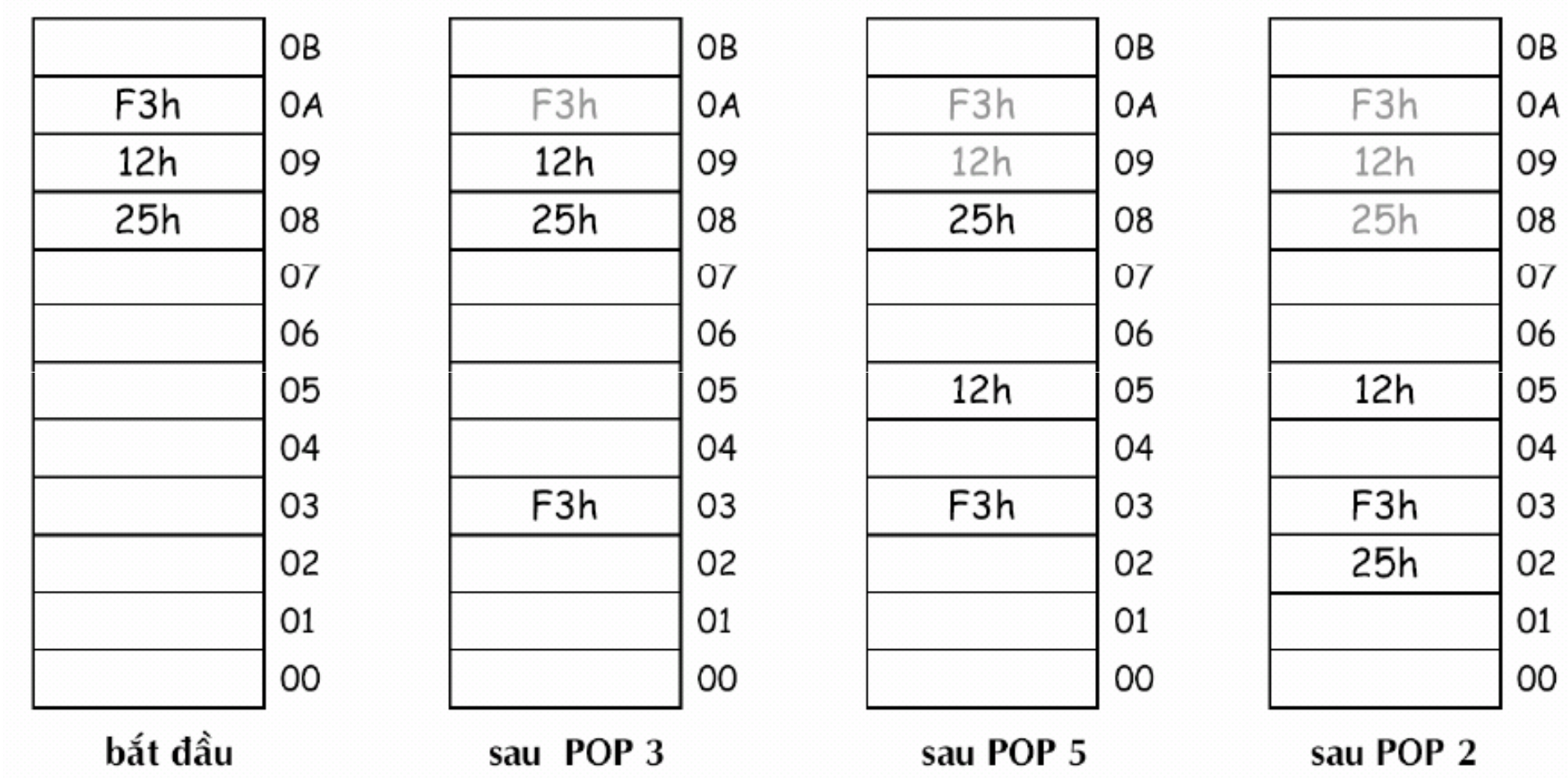
Thí dụ về PUSH và POP

```
MOV     R6,#25h
MOV     R1,#12h
MOV     R4,#0F3h
PUSH    6
PUSH    1
PUSH    4
POP     3 ; lấy từ stack chép vào R3
POP     5 ; lấy từ stack chép vào R5
POP     2 ; lấy từ stack chép vào R2
```

Ảnh hưởng của các lệnh PUSH với ngăn xếp



Ảnh hưởng của các lệnh POP với ngăn xếp



Chú ý về stack trong 8051

- Ngăn xếp trong 8051 tăng trưởng theo hướng địa chỉ tăng (08H, 09H, ...)
- Ngăn xếp chiếm cùng vị trí với băng thanh ghi 1, do đó khi sử dụng ngăn xếp phải cẩn thận khi sử dụng băng thanh ghi 1 (không thể sử dụng được lúc này). Tuy nhiên ta có thể giải quyết vấn đề này bằng cách ghi lại trị bắt đầu mới cho SP (các trị từ 30H đến 7FH).
- Khi ngăn xếp tăng trưởng có khả năng lọt vào vùng có địa chỉ bit (từ byte có địa chỉ 30H).

Look-up Tables

```
                MOV         A,#ENTRY
                CALL        LOOK_UP
                .
                .
LOOK_UP:        INC         A
                MOVC        A,@A+PC
                RET
TABLE:          DB          data,data,data,...
```

Thí dụ: Đọc một số X từ Port 1 và xuất giá trị X^2 ra Port 2

```
        ORG 0                   ; assembler directive
        MOV DPTR, #LUT          ; 300H là địa chỉ đầu bảng
        MOV A, #0FFH
        MOV P1, A       ; Lập trình cổng P1 để nhập dữ liệu
Again:  MOV   A, P1             ; Đọc X
        MOVC  A, @A+DPTR        ; Lấy X2 bằng tra bảng
        MOV   P2, A             ; Xuất X2 ra P2
        SJMP  Again     ; Lặp lại mãi mãi đoạn Again đến SJMP
        ORG 300H                ; Bảng tra bắt đầu ở 0300H
LUT:    DB 0, 1, 4, 9, 16, 25, 36, 49, 64, 81
        DB 100, 121, 144, 169, 196, 225
```

; LUT=Look-Up Table=Bảng tra cứu

EXAMPLE

Write the 8051 instruction to perform the following operations.
(a) Move the contents of the accumulator to register 5.
(b) Move the contents of RAM memory location 42H to port 1.
(c) Move the value at port 2 to register 3.
(d) Send FFH to port 0.
(e) Send the contents of RAM memory, whose address is in register 1, to port 3

Solution:

(a) MOV R5, A
(b) MOV P1, 42H
(c) MOV R3, P2
(d) MOV P0, #0FFH
(e) MOV P3, @R1

TD: Đảo ngược các bit của A

```
        MOV     R7,#8
Loop:   RLC     A
        XCH     A,0F0H
        RRC     A
        XCH     A,0F0H
        DJNZ    R7,LOOP
        XCH     A,0F0H
```

4. Tóm tắt các lệnh với biến Boole
(giả sử 8051 với thạch anh 12 MHz)

MNEMONIC (Từ gợi nhớ)	OPERATION (Phép toán/Tác vụ)	EXECUTION TIME (μs) (Thời gian thực thi)
ANL C,bit	(C) = (C) AND (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	Nhảy nếu (C) = 1	2
JNC rel	Nhảy nếu (C) = 0	2
JB bit,rel	Nhảy nếu (bit) = 1	2
JNB bit,rel	Nhảy nếu (bit) = 0	2
JBC bit,rel	Nhảy nếu (bit) = 1; CLR bit	2

Boolean Variable Manipulation

- CLR C ;clear bit
- CLR bit
- SETB C ;set bit
- SETB bit
- CPL C ;complement bit
- CPL bit
- ANL C,bit ;AND bit with C
- ANL C,/bit ;And NOT bit with C

Boolean Variable Manipulation

- ORL C,bit ;OR bit with C
- ORL C,/bit ;OR NOT bit with C
- MOV C,bit ;move bit to bit
- MOV bit,C
- JC rel ;jump if C is set
- JNC rel ;jump if C is not set
- JB bit,rel ;jump if bit is set
- JNB bit,rel ;jump if bit is not set
- JBC bit,rel ;jump if set then clear

Phép toán XOR với biến Boole

Chú ý là các lệnh với biến Boole bao gồm các phép toán ANL (logic AND) và ORL (logic OR), mà không có phép toán XRL (logic XOR). Tuy nhiên ta có thể thực hiện phép toán XOR bằng cách kết hợp các phép toán AND, OR, NOT hoặc sử dụng cách sau (thí dụ muốn thực hiện BIT1 $\oplus$ BIT2)

```
        MOV   C, BIT1
        JNB   BIT2, SKIP
        CPL   C
SKIP:   (tiếp tục)
```

Trước hết BIT1 được chuyển vào cờ nhớ. Nếu BIT2 = 0 thì C chứa kết quả đúng; nghĩa là BIT1 $\oplus$ BIT2 = BIT1 nếu BIT2 = 0. Nếu BIT2 = 1 thì C chứa phủ định của kết quả (vì BIT1 $\oplus$ BIT2 = NOT BIT1 nếu BIT2 = 1).

The 8051 Instruction Set

EXAMPLE

Use the SETB, CLR, and CPL instructions to do the following operations:

(a) Clear bit 7 of the accumulator

(b) Output a 1 on bit 0 of port 3

(c) Complement the parity flag (bit 0 of the PSW)

Solution:

(a) CLR ACC.7

(b) SETB P3.0

(c) CPL PSW.0

Thí dụ: Định trị một hàm Boole F = WX'Y + XY'Z'

Cách 1:

```
      W   EQU     P1.3
      X   EQU     P1.2
      Y   EQU     P1.1
      Z   EQU     P1.0
      F   EQU     P1.7
      TMP EQU     00h  ; bit 0 của ô nhớ
          20H
          ORG     8000H
          MOV     A, #0Fh
          MOV     P1, A
BACK:     MOV     C, W
          ANL     C, /X
          ANL     C, Y
          MOV     TMP, C
          MOV     C, X
          ANL     C, /Y
          ANL     C, /Z
          ORL     C, TMP
          MOV     F, C
          SJMP    BACK
```

Cách 2: Dùng các lệnh điều kiện

```
      W     EQU     P1.3
      X     EQU     P1.2
      Y     EQU     P1.1
      Z     EQU     P1.0
      F     EQU     P1.7
      TMP   EQU     00h ; bit 0 của ô nhớ 20H
            ORG     8000H
            MOV     A, #0Fh
            MOV     P1, A
BACK:       JNB     W, CAL2
            JNB     W, CAL2
            JB      X, CAL2
            JB      Y, SETIT
CAL2:       JNB     X, CLEAR
            JB      Y, CLEAR
            JB      Z, CLEAR
SETIT:      SETB    F
            SJMP BACK
CLEAR:      CLR     F
            SJMP    BACK
```

Tóm tắt các lệnh rẽ nhánh không điều kiện
(giả sử 8051 với thạch anh 12 MHz)

MNEMONIC (Từ gợi nhớ)	OPERATION (Phép toán/Tác vụ)	EXECUTION TIME (µs) (Thời gian thực thi)
JMP addr	Nhảy đến địa chỉ ***addr*** Có các dạng: AJMP addr11 LJMP addr16 SJMP rel	2
JMP @A+DPTR	Nhảy đến ((A) + (DPTR))	2
CALL addr	Gọi trình con ở địa chỉ ***addr*** Có các dạng: ACALL addr11 LCALL addr16	2
RET	Quay về từ chương trình con	2
RETI	Quay về từ chương trình phục vụ ngắt	2
NOP	Không làm gì cả	1

5. Tóm tắt các lệnh rẽ nhánh có điều kiện
(giả sử 8051 với thạch anh 12 MHz)

MNEMONIC (Từ gợi nhớ)	OPERATION (Phép toán/Tác vụ)	ADDRESSING MODES (Các cách định địa chỉ)				EXECUTION TIME (μs)
		DIR	IND	REG	IMM	(Thời gian thực thi)
JZ rel	Nhảy nếu (A) = 0	Chỉ với thanh ghi tích lũy ACC				2
JNZ rel	Nhảy nếu (A) ≠ 0	Chỉ với thanh ghi tích lũy ACC				2
DJNZ byte, rel	Giảm (byte) đi 1 và nhảy nếu ≠ 0	X		X		2
CJNE A, byte, rel	Nhảy nếu (A) ≠ (byte)	X			X	2
CJNE byte, #data8, rel	Nhảy nếu (byte) ≠ data8		X	X		2

Program Branching

- ACALL addr11 ;call subroutine
- LCALL addr16
- RET ;return from subroutine
- RETI ;return from interrupt
- AJMP addr11 ;absolute jump
- LJMP addr16 ;long jump
- SJMP rel ;short (relative) jump
- JMP @A+DPTR
- JZ rel ;jump if A = 0

Program Branching

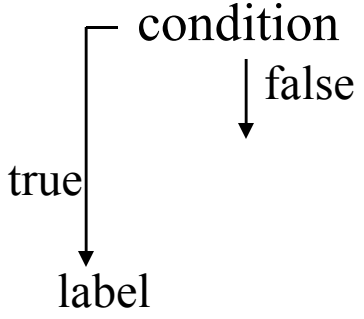
- JZ rel ;jump if A = 0
- JNZ rel ;jump if A not = 0
- CJNE A,direct,rel ;compare and
- CJNE A,#data,rel ;rel jump if not ;equal
- CJNE Rn,#data,rel
- CJNE @Ri,#data,rel
- DJNZ Rn,rel ;decrement & jump
- DJNZ direct,rel ;if not zero
- NOP ;no operation

Conditional jumps

Mnemonic	Description
`JZ <rel addr>`	Jump if a = 0
`JNZ <rel addr>`	Jump if a != 0
`JC <rel addr>`	Jump if C = 1
`JNC <rel addr>`	Jump if C != 1
`JB <bit>, <rel addr>`	Jump if bit = 1
`JNB <bit>,<rel addr>`	Jump if bit != 1
`JBC <bir>, <rel addr>`	Jump if bit =1, clear bit
`CJNE A, direct, <rel addr>`	Compare A and memory, jump if not equal

Conditional Jumps for Branching

if condition is true
    goto label
else
    goto next instruction



if a = 0 is true
    send a 0 to LED
else
    send a 1 to LED

```
            jz led_off
            setb C
            mov P1.6, C
            sjmp skipover
led_off:    clr C
            mov P1.6, C
skipover:   mov A, P0
```

EXAMPLE

Write a program that continuously reads a byte from port 1 and writes it to port 0 until the byte read equals zero.

Solution:

```
READ:   MOV   A, P1 ; A ←P1
        MOV   P0, A ; P0←A
        JNZ   READ ; Repeat until A = 0
        NOP ; Remainder of program
        etc.
```

More Conditional Jumps

Mnemonic	Description
`CJNE A, #data <rel addr>`	Compare A and data, jump if not equal
`CJNE Rn, #data <rel addr>`	Compare Rn and data, jump if not equal
`CJNE @Rn, #data <rel addr>`	Compare Rn and memory, jump if not equal
`DJNZ Rn, <rel addr>`	Decrement Rn and then jump if not zero
`DJNZ direct, <rel addr>`	Decrement memory and then jump if not zero

EXAMPLE

Repeat the previous example, except stop the looping when the number 77H is read

Solution:

```
READ:   MOV A, P1 ; A ←P1
        MOV P0, A ; P0←A
        CJNE A, #77, READ ; Repeat until A = 77H
        NOP ; Remainder of program
        etc.
```

EXAMPLE

Repeat the previous example, except stop the looping when bit 3 of port 2 is set.

Solution:

```
READ:   MOV A, P1 ; A ←P1
        MOV P0, A ; P0←A
        JNB P2.3, READ ; Repeat until P2.3 = 1
        NOP ; Remainder of program
        etc.
```

Iterative Loops

For A = 0 to 4 do

{…}

```
      clr a
loop: ...
      inc a
      cjne a, #4, loop
```

For A = 4 to 0 do

{…}

```
      mov R0, #4
loop: ...
      ...
      djnz R0, loop
```

Execute Loop N Times

```
        MOV     R7,#10 (SAY N=10)
LOOP:   (begin loop)
        .
        .
        (end loop)
        DJNZ    R7,LOOP
        (continue)
```

EXAMPLE

Write a program that will produce an output at port 0 that counts down from 80H to 00H.

Solution:

```
        MOV R0, #80H ; R0 ←80H
COUNT:  MOV P0, R0 ; P0 ←R0
        DJNZ R0, COUNT ; Decrement R0, jump to
        ; COUNT if not 0
        NOP ; Remainder of program
        etc.
```

Jump Tables

```
        MOV     DPTR, #JUMP_TABLE
        MOV     A, #INDEX_NUMBER
        RL      A
        JMP     @A+DPTR

JUMP_TABLE:  AJMP    CASE0
             AJMP    CASE1
             AJMP    CASE2
```

Compare & Jump

- Compare two unsigned bytes:

```
        CJNE    A,B,$+3
        JNC     BIG
LE:     .               ;less than or
        .               ;equal to

BIG:    .               ;bigger than
```

Thí dụ: Đổi từ binary sang biểu diễn số qua ASCII

Chương trình lấy một số trong ACC và cất biểu diễn ASCII của nó vào RAM nội ở địa chỉ trong R1. Thí dụ để chuyển 239 thành các ký tự ASCII '2', '3', '9'; trước hết lấy ký số đầu bằng cách chia cho 100 (được 2 và đổi thành '2'); lấy phần dư từ phép chia để có 39; lấy 39 chia 10 được 3 (và đổi thành '3'), và phần dư của phép chia này cho 9 (và đổi thành '9').

Bài giải.

```
        ORG     00h
        MOV     A,#239          ; Nạp giá trị thử vào ACC
        MOV     R1,#040h        ; và địa chỉ đích vào R1
        LCALL   TODEC           ; TODEC(239, 040h);
        MOV     A,#17           ; Bây giờ thí dụ với 2 ký số
        MOV     R1, #050h       ; và địa chỉ đích ở R1
        LCALL   TODEC
        NOP                     ; thêm lệnh NOP này khi chạy mô phỏng
```

CT con TODEC

```
;================================================
; Chương trình con này đổi 1 byte thành 3 ký tự ASCII có thể in được từ 000–255
; Lấy số trong ACC và cất biểu diễn ASCII của nó trong RAM nội bắt đầu ở địa chỉ
; trong R1.
TODEC: MOV     B, #100  ; Lấy số chia để vào B
       DIV     AB               ; ACC =  num/100 -> 2 , B = ACC % 100 -> 39
       LCALL   TOASCII          ; đổi sang Ascii và cất trong bộ nhớ
       MOV     A, B             ; Lấy phần dư vào A (39)
       MOV     B, #10           ;
       DIV     AB               ; ACC = 39 / 10 -> 3,   phần dư = 39 % 10 -> 9
       LCALL TOASCII  ; Đổi và cất 3
       MOV     A, B             ; Lấy giá trị cuối cùng đưa vào A
       LCALL TOASCII
       RET                      ; Thực hiện xong
```

CT con TOASCII

```
;===========================================
; Đổi 1 số trong ACC thành một ký tự ASCII
; Cất nó vào ô nhớ có địa chỉ cho bởi R1

TOASCII:    ADD    A, #'0'   ; đổi từ nhị phân sang ASCII
                              ; cũng có thể sử dụng "ADD  A,#030h”
            MOV    @R1, A    ; Cất vào bộ nhớ nội
            INC    R1        ; Tăng con trỏ địa chỉ
            RET

            END
```

Thí dụ

Thí dụ: Viết chương trình điền vào các ô nhớ từ 048H đến 057H với các giá trị 0, 2, 4 . . . 30 (0–1EH), đặt 0 vào ô nhớ 48H, 2 vào 49H, ...

Bài giải.

```
                ORG     0
        MOV     R0, #048H   ; Đặt địa chỉ của byte thứ nhất vào R0

        MOV     A, #0       ; Điền giá trị vào Acc
LOOP:   MOV     @R0, A      ; lưu trữ byte (tương tự C:-  *R0 = A)
        CJNE    A, #30, NEXT        ; Lưu trữ giá trị sau cùng
        SJMP  DONE
NEXT:   ADD     A, #2       ; Tăng ACC thành giá trị kế
        INC   R0            ; Chỉ R0 đến byte kế
        SJMP    LOOP                ; Ghi giá trị kế
DONE:   NOP                         ; NOP không làm gì cả
        END
```