MSE352: Digital Logic & Microcontrollers

Lecture 5
Addressing Modes

Mohammad Narimani, *Ph.D., P.Eng.*
Lecturer
School of Mechatronic Systems Engineering
Simon Fraser University

SFU SIMON FRASER UNIVERSITY
SURREY

Outline

- Addressing modes
- Accessing memory
- Bit addresses
- Extra 128 byte on-chip RAM in 8052

Addressing modes

- The CPU can access data in various ways, which are called addressing modes :
  - Immediate
  - Register
  - Direct
  - Register indirect
  - Indexed

Accessing memories

Addressing modes

- The source operand is a constant
  - The immediate data must be preceded by the pound sign, “#”
  - Can load information into any registers, including 16-bit DPTR register
    - DPTR can also be accessed as two 8-bit registers, the high byte DPH and low byte DPL

```
MOV A,#25H        ;load 25H into A
MOV R4,#62        ;load 62 into R4
MOV B,#40H        ;load 40H into B
MOV DPTR,#4521H   ;DPTR=4512H
MOV DPL,#21H      ;This is the same
MOV DPH,#45H      ;as above

;illegal!! Value > 65535 (FFFFH)
MOV DPTR,#68975
```

Addressing modes

- We can use EQU directive to access immediate data

```
          Count   EQU 30
          ...     ...
          MOV     R4,#COUNT          ;R4=1EH
          MOV     DPTR,#MYDATA       ;DPTR=200H

          ORG     200H
MYDATA:   DB      "America"
```

- We can also use immediate addressing mode to send data to 8051 ports

```
          MOV P1,#55H
```

Addressing modes

- Use registers to hold the data to be manipulated

```
MOV A,R0      ;copy contents of R0 into A
MOV R2,A      ;copy contents of A into R2
ADD A,R5      ;add contents of R5 to A
ADD A,R7      ;add contents of R7 to A
MOV R6,A      ;save accumulator in R6
```

- The source and destination registers must match in size
  - MOV DPTR,A will give an error

```
MOV DPTR,#25F5H
MOV R7,DPL
MOV R6,DPH
```

- The movement of data between Rn registers is not allowed
  - MOV R4,R7 is invalid

Outline

- Addressing modes
- Accessing memory
- Bit addresses
- Extra 128 byte on-chip RAM in 8052

Accessing memory

- It is most often used the direct addressing mode to access RAM locations 30 – 7FH
  - The entire 128 bytes of RAM can be accessed
  - The register bank locations are accessed by the register names

```
MOV A,4      ;is same as
MOV A,R4     ;which means copy R4 into A
```

- Contrast this with immediate addressing mode
  - There is no “#” sign in the operand

```
MOV R0,40H   ;save content of 40H in R0
MOV 56H,A    ;save content of A in 56H
```

Accessing memory

- The SFR (Special Function Register) can be accessed by their names or by their addresses

```
MOV 0E0H,#55H    ;is the same as
MOV A,#55h       ;load 55H into A

MOV 0F0H,R0      ;is the same as
MOV B,R0         ;copy R0 into B
```

- The SFR registers have addresses between 80H and FFH
  - Not all the address space of 80 to FF is used by SFR
  - The unused locations 80H to FFH are reserved and must not be used by the 8051 programmer

Accessing memory

Special Function Register (SFR) Addresses

Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B register	0F0H
PSW*	Program status word	0D0H
SP	Stack pointer	81H
DPTR	Data pointer 2 bytes	
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H
P3*	Port 3	0B0H
IP*	Interrupt priority control	0B8H
IE*	Interrupt enable control	0A8H
…	…	…

Accessing memory

Special Function Register (SFR) Addresses

Symbol	Name	Address
TMOD	Timer/counter mode control	89H
TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
T2MOD	Timer/counter mode control	OC9H
TH0	Timer/counter 0 high byte	8CH
TL0	Timer/counter 0 low byte	8AH
TH1	Timer/counter 1 high byte	8DH
TL1	Timer/counter 1 low byte	8BH
TH2	Timer/counter 2 high byte	0CDH
TL2	Timer/counter 2 low byte	0CCH
RCAP2H	T/C 2 capture register high byte	0CBH
RCAP2L	T/C 2 capture register low byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON	Power ontrol	87H

Accessing memory

Example 5-1

Write code to send 55H to ports P1 and P2, using
(a) their names (b) their addresses

Solution :

(a)
```
MOV  A,#55H        ;A=55H
MOV  P1,A          ;P1=55H
MOV  P2,A          ;P2=55H
```

(b) From Table 5-1, P1 address=80H; P2 address=A0H
```
MOV  A,#55H        ;A=55H
MOV  80H,A         ;P1=55H
MOV  0A0H,A        ;P2=55H
```

Accessing memory

- Only direct addressing mode is allowed for pushing or popping the stack
  - PUSH A is invalid
  - Pushing the accumulator onto the stack must be coded as PUSH 0E0H

Example 5-2

Show the code to push R5 and A onto the stack and then pop them back them into R2 and B, where B = A and R2 = R5

Solution:

```
PUSH 05        ;push R5 onto stack
PUSH 0E0H      ;push register A onto stack
POP  0F0H      ;pop top of stack into B
               ;now register B = register A
POP  02        ;pop top of stack into R2
               ;now R2=R6
```

Accessing memory

- A register is used as a pointer to the data
  - Only register R0 and R1 are used for this purpose
  - R2 – R7 cannot be used to hold the address of an operand located in RAM
- When R0 and R1 hold the addresses of RAM locations, they must be preceded by the “@” sign

```
MOV A,@R0    ;move contents of RAM whose
             ;address is held by R0 into A
MOV @R1,B    ;move contents of B into RAM
             ;whose address is held by R1
```

Accessing memory

<u>Example 5-3</u>

Write a program to copy the value 55H into RAM memory locations 40H to 41H using
(a) direct addressing mode, (b) register indirect addressing mode without a loop, and (c) with a loop

Solution:

(a)

```
    MOV A,#55H   ;load A with value 55H
    MOV 40H,A    ;copy A to RAM location 40H
    MOV 41H,A    ;copy A to RAM location 41H
```

(b)

```
    MOV A,#55H   ;load A with value 55H
    MOV R0,#40H  ;load the pointer. R0=40H
    MOV @R0,A    ;copy A to RAM R0 points to
    INC R0       ;increment pointer. Now R0=41h
    MOV @R0,A    ;copy A to RAM R0 points to
```

(c)

```
       MOV A,#55H     ;A=55H
       MOV R0,#40H    ;load pointer.R0=40H,
       MOV R2,#02     ;load counter, R2=3
AGAIN: MOV @R0,A      ;copy 55 to RAM R0 points to
       INC R0         ;increment R0 pointer
       DJNZ R2,AGAIN  ;loop until counter = zero
```

Accessing memory

- The advantage is that it makes accessing data dynamic rather than static as in direct addressing mode
  - Looping is not possible in direct addressing mode

Example 5-4

Write a program to clear 16 RAM locations starting at RAM address 60H

Solution:

```
        CLR A           ;A=0
        MOV R1,#60H     ;load pointer. R1=60H
        MOV R7,#16      ;load counter, R7=16
AGAIN:  MOV @R1,A       ;clear RAM R1 points to
        INC R1          ;increment R1 pointer
        DJNZ R7,AGAIN   ;loop until counter=zero
```

Accessing memory

Example 5-5

Write a program to copy a block of 10 bytes of data from 35H to 60H

Solution:

```
       MOV R0,#35H   ;source pointer
       MOV R1,#60H   ;destination pointer
       MOV R3,#10    ;counter
BACK:  MOV A,@R0     ;get a byte from source
       MOV @R1,A     ;copy it to destination
       INC R0        ;increment source pointer
       INC R1      ;increment destination pointer
       DJNZ R3,BACK ;keep doing for ten bytes
```

Accessing memory

- R0 and R1 are the only registers that can be used for pointers in register indirect addressing mode

- Since R0 and R1 are 8 bits wide, their use is limited to access any information in the internal RAM

- Whether accessing externally connected RAM or on-chip ROM, we need 16-bit pointer
  - In such case, the DPTR register is used

Accessing memory

- Indexed addressing mode is widely used in accessing data elements of look-up table entries <u>located in the program ROM</u>

- The instruction used for this purpose is MOVC A,@A+DPTR
  - Use instruction MOVC, "C" means code
  - The contents of A are added to the 16-bit register DPTR to form the 16-bit address of the needed data

Accessing memory

Example 5-6

In this program, assume that the word "USA" is burned into ROM locations starting at 200H. And that the program is burned into ROM locations starting at 0. Analyze how the program works and state where "USA" is stored after this program is run.

Solution:

```
        ORG   0000H       ;burn into ROM starting at 0
        MOV   DPTR,#200H  ;DPTR=200H look-up table addr
        CLR   A           ;clear A(A=0)
        MOVC  A,@A+DPTR   ;get the char from code space
        MOV   R0,A        ;save it in R0
        INC   DPTR        ;DPTR=201 point to next char
        CLR   A           ;clear A(A=0)
        MOVC  A,@A+DPTR   ;get the next char
        MOV   R1,A        ;save it in R1
        INC   DPTR        ;DPTR=202 point to next char
        CLR   A           ;clear A(A=0)
        MOVC  A,@A+DPTR   ;get the next char
        MOV   R2,A        ;save it in R2
Here:   SJMP  HERE        ;stay here
;Data is burned into code space starting at 200H

        ORG   200H
MYDATA:DB "USA"
        END               ;end of program
```

DPTR=200H, A=0

DPTR=200H, A=55H

DPTR=201H, A=55H

DPTR=201H, A=0

DPTR=201H, A=53H

DPTR=202H, A=53H

R0=55H

R1=53H

R2=41H

202	A
201	S
200	U

Accessing memory

- The look-up table allows access to elements of a frequently used table with minimum operations

Example 5-8

Write a program to get the x value from P1 and send x^2 to P2, continuously

Solution:

```
      ORG   0
      MOV   DPTR,#300H    ;LOAD TABLE ADDRESS
      MOV   A,#0FFH       ;A=FF
      MOV   P1,A          ;CONFIGURE P1 INPUT PORT
BACK:MOV   A,P1          ;GET X
      MOV   A,@A+DPTR     ;GET X SQAURE FROM TABLE
      MOV   P2,A          ;ISSUE IT TO P2
      SJMP  BACK          ;KEEP DOING IT

      ORG   300H
XSQR_TABLE:
      DB    0,1,4,9,16,25,36,49,64,81
      END
```

Accessing memory

- In many applications we use RAM locations 30 – 7FH as scratch pad
  - We use R0 – R7 of bank 0
  - Leave addresses 8 – 1FH for stack usage
  - If we need more registers, we simply use RAM locations 30 – 7FH

Example 5-10

Write a program to toggle P1 a total of 200 times. Use RAM location 32H to hold your counter value instead of registers R0 – R7

Solution:

```
        MOV     P1,#55H    ;P1=55H
        MOV     32H,#200   ;load counter value
                           ;into RAM loc 32H
LOP1:   CPL     P1         ;toggle P1
        ACALL   DELAY
        DJNZ    32H,LOP1   ;repeat 200 times
```

Outline

- Addressing modes
- Accessing memory
- Bit addresses
- Extra 128 byte on-chip RAM in 8052

Bit addresses

- Many microprocessors allow program to access registers and I/O ports in byte size only
  - However, in many applications we need to check a single bit

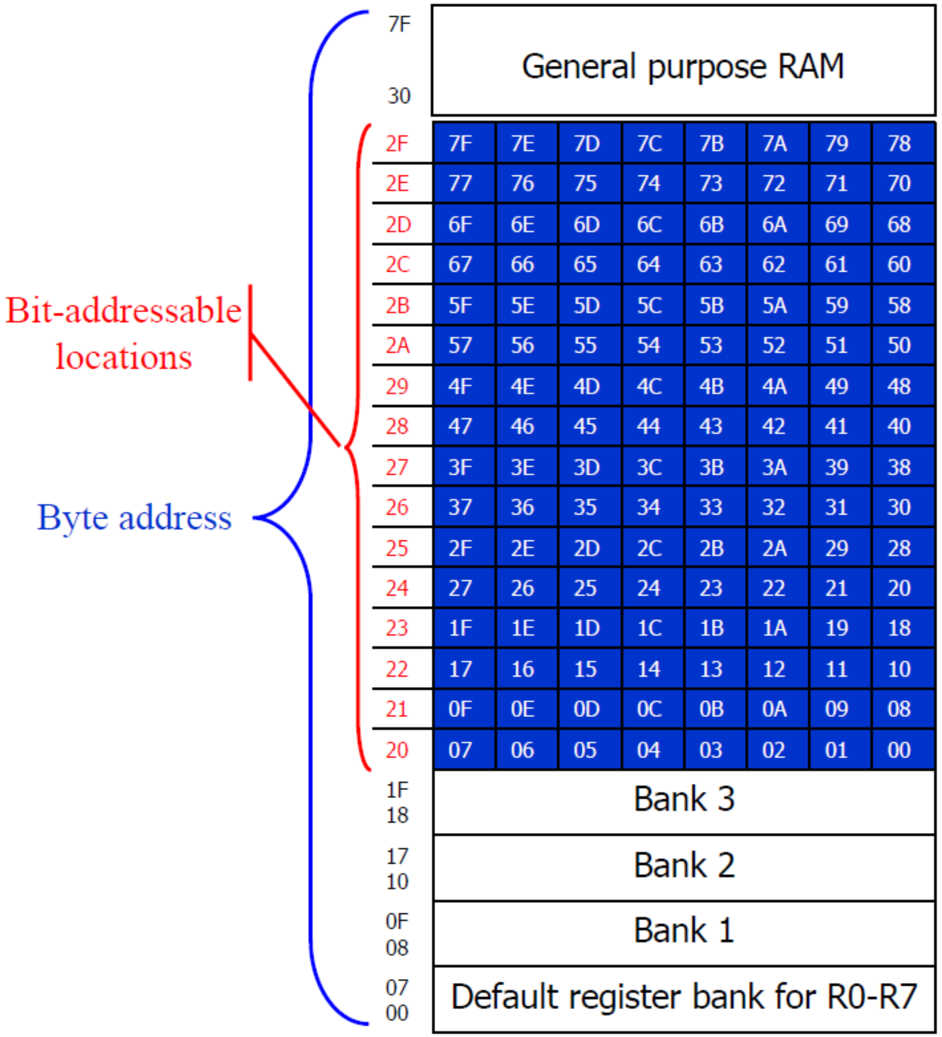
- One unique and powerful feature of the 8051 is single-bit operation
  - Single-bit instructions allow the programmer to set, clear, move, and complement individual bits of a port, memory, or register
  - Registers, RAM, and I/O ports need to be bit-addressable
  - ROM, holding program code for execution, is not bit-addressable

Bit addresses

- The bit-addressable RAM location are 20H to 2FH
  - These 16 bytes provide 128 bits of RAM bit-addressability, since $16 \times 8 = 128$
    - 0 to 127 (in decimal) or 00 to 7FH
  - The first byte of internal RAM location 20H has bit address 0 to 7H
  - The last byte of 2FH has bit address 78H to 7FH

- Internal RAM locations 20-2FH are both byte-addressable and bit-addressable
  - Bit address 00-7FH belong to RAM byte addresses 20-2FH
  - Bit address 80-F7H belong to SFR P0, P1, …

Bit addresses



Bit addresses

Example 5-11

Find out to which by each of the following bits belongs. Give the address of the RAM byte in hex

(a) `SETB 42H`, (b) `CLR 67H`, (c) `CLR 0FH`

(d) `SETB 28H`, (e) `CLR 12`, (f) `SETB 05`

Solution:

(a) D2 of RAM location 28H

(b) D7 of RAM location 2CH

(c) D7 of RAM location 21H

(d) D0 of RAM location 25H

(e) D4 of RAM location 21H

(f) D5 of RAM location 20H

	D7	D6	D5	D4	D3	D2	D1	D0
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00

Bit addresses

- To avoid confusion regarding the addresses 00 – 7FH
  - The 128 bytes of RAM have the byte addresses of 00 – 7FH can be accessed in byte size using various addressing modes
    - Direct and register-indirect
  - The 16 bytes of RAM locations 20 – 2FH have bit address of 00 – 7FH
    - We can use only the single-bit instructions and these instructions use only direct addressing mode

Bit addresses

- Instructions that are used for signal-bit operations are as following

Single-Bit Instructions

Instruction	Function
SETB bit	Set the bit (bit = 1)
CLR bit	Clear the bit (bit = 0)
CPL bit	Complement the bit (bit = NOT bit)
JB bit, target	Jump to target if bit = 1 (jump if bit)
JNB bit, target	Jump to target if bit = 0 (jump if no bit)
JBC bit, target	Jump to target if bit = 1, clear bit (jump if bit, then clear)

Bit addresses

- While all of the SFR registers are byte-addressable, some of them are also bit-addressable
  - The P0 – P3 are bit addressable
  - We can access either the entire 8 bits or any single bit of I/O ports P0, P1, P2, and P3 without altering the rest

- When accessing a port in a single-bit manner, we use the syntax SETB X.Y
  - X is the port number P0, P1, P2, or P3
  - Y is the desired bit number from 0 to 7 for data bits D0 to D7
  - ex. SETB P1.5 sets bit 5 of port 1 high

Bit addresses

- Notice that when code such as SETB P1.0 is assembled, it becomes SETB 90H
  - The bit address for I/O ports
    - P0 are 80H to 87H
    - P1 are 90H to 97H
    - P2 are A0H to A7H
    - P3 are B0H to B7H

Single-Bit Addressability of Ports

P0	P1	P2	P3	Port Bit
P0.0 (80)	P1.0 (90)	P2.0 (A0)	P3.0 (B0)	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7 (87)	P1.7 (97)	P2.7 (A7)	P3.7 (B7)	D7

Bit addresses

SFR RAM Address (Byte and Bit)

Bit addresses 80 – F7H belong to SFR of P0, TCON, P1, SCON, P2, etc

Byte address	Bit address	
FF		
F0	F7 F6 F5 F4 F3 F2 F1 F0	B
E0	E7 E6 E5 E4 E3 E2 E1 E0	ACC
D0	D7 D6 D5 D4 D3 D2 D1 D0	PSW
B8	-- -- -- BC BB BA B9 B8	IP
B0	B7 B6 B5 B4 B3 B2 B1 B0	P3
A8	AF AE AD AC AB AA A9 A8	IE
A0	A7 A6 A5 A4 A3 A2 A1 A0	P2
99	not bit addressable	SBUF

Byte address	Bit address	
98	9F 9E 9D 9C 9B 9A 99 98	SCON
90	97 96 95 94 93 92 91 90	P1
8D	not bit addressable	TH1
8C	not bit addressable	TH0
8B	not bit addressable	TL1
8A	not bit addressable	TL0
89	not bit addressable	TMOD
88	8F 8E 8D 8C 8B 8A 89 88	TCON
87	not bit addressable	PCON
83	not bit addressable	DPH
82	not bit addressable	DPL
81	not bit addressable	SP
80	87 86 85 84 83 82 81 80	P0

Special Function Register

Bit addresses

- Only registers A, B, PSW, IP, IE, ACC, SCON, and TCON are bit-addressable
  - While all I/O ports are bit-addressable
- In PSW register, two bits are set aside for the selection of the register banks
  - Upon RESET, bank 0 is selected
  - We can select any other banks using the bit-addressability of the PSW

CY	AC	--	RS1	RS0	OV	--	P

RS1	RS0	Register Bank	Address
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Bit addresses

Example 5-13

Write a program to save the accumulator in R7 of bank 2.

Solution:

```
CLR     PSW.3
SETB    PSW.4
MOV     R7,A
```

Example 5-14

While there are instructions such as JNC and JC to check the carry flag bit (CY), there are no such instructions for the overflow flag bit (OV). How would you write code to check OV?

Solution:

```
JB      PSW.2,TARGET    ;jump if OV=1
```

CY	AC	--	RS1	RS0	OV	--	P

Example 5-18

Write a program to save the status of bit P1.7 on RAM address bit 05.

Solution:

```
MOV     C,P1.7
MOV     05,C
```

Bit addresses

Example 5-15

Write a program to see if the RAM location 37H contains an even value. If so, send it to P2. If not, make it even and then send it to P2.

Solution:

```
        MOV     A,37H       ;load RAM 37H into ACC
        JNB     ACC.0,YES   ;if D0 of ACC 0? If so jump
        INC     A           ;it's odd, make it even
YES:    MOV     P2,A        ;send it to P2
```

Example 5-17

The status of bits P1.2 and P1.3 of I/O port P1 must be saved before they are changed. Write a program to save the status of P1.2 in bit location 06 and the status of P1.3 in bit location 07

Solution:

```
        CLR     06          ;clear bit addr. 06
        CLR     07          ;clear bit addr. 07
        JNB     P1.2,OVER   ;check P1.2, if 0 then jump
        SETB    06          ;if P1.2=1,set bit 06 to 1
OVER:   JNB     P1.3,NEXT   ;check P1.3, if 0 then jump
        SETB    07          ;if P1.3=1,set bit 07 to 1
NEXT:   ...
```

Bit addresses

- The BIT directive is a widely used directive to assign the bit-addressable I/O and RAM locations
  - Allow a program to assign the I/O or RAM bit at the beginning of the program, making it easier to modify them

Example 5-22

A switch is connected to pin P1.7 and an LED to pin P2.0. Write a program to get the status of the switch and send it to the LED.

Solution:

```
LED     BIT     P1.7    ;assign bit
SW      BIT     P2.0    ;assign bit
HERE:   MOV     C,SW    ;get the bit from the port
        MOV     LED,C   ;send the bit to the port
        SJMP    HERE    ;repeat forever
```

Bit addresses

Example 5-20

Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

Solution:

```
OVEN_HOT   BIT P2.3
BUZZER     BIT P1.5
HERE:    JNB      OVEN_HOT,HERE ;keep monitoring
         ACALL    DELAY
         CPL      BUZZER   ;sound the buzzer
         ACALL    DELAY
         SJMP     HERE
```

Bit addresses

- Use the EQU to assign addresses
  - Defined by names, like P1.7 or P2
  - Defined by addresses, like 97H or 0A0H

Example 5-24

A switch is connected to pin P1.7. Write a program to check the status of the switch and make the following decision.
(a) If SW = 0, send "0" to P2
(b) If SW = 1, send "1" to P2

Solution:

```
SW       EQU  P1.7
MYDATA   EQU  P2
HERE:    MOV      C,SW
         JC       OVER
         MOV      MYDATA,#'0'
         SJMP     HERE
OVER:    MOV      MYDATA,#'1'
         SJMP     HERE
         END
```

```
SW      EQU 97H
MYDATA  EQU 0A0H
```

Outline

- Addressing modes
- Accessing memory
- Bit addresses
- Extra 128 byte on-chip RAM in 8052

Extra 128 byte on-chip RAM in 8052

- The 8052 has another 128 bytes of on-chip RAM with addresses 80 – FFH
  - It is often called upper memory
  - Use indirect addressing mode, which uses R0 and R1 registers as pointers with values of 80H or higher
    - – MOV @R0, A and MOV @R1, A
  - The same address space assigned to the SFRs
    - Use direct addressing mode
  - – MOV 90H, #55H is the same as

    MOV P1, #55H

Extra 128 byte on-chip RAM in 8052`

Example 5-27

Assume that the on-chip ROM has a message. Write a program to copy it from code space into the upper memory space starting at address 80H. Also, as you place a byte in upper RAM, give a copy to P0.

Solution:

```
        ORG     0
        MOV     DPTR,#MYDATA
        MOV     R1,#80H         ;access the upper memory
B1:     CLR     A
        MOVC    A,@A+DPTR       ;copy from code ROM
        MOV     @R1,A           ;store in upper memory
        MOV     P0,A            ;give a copy to P0
        JZ      EXIT            ;exit if last byte
        INC     DPTR            ;increment DPTR
        INC     R1              ;increment R1
        SJMP    B1              ;repeat until last byte
EXIT:   SJMP    $               ;stay here when finished
;---------------
        ORG     300H
MYDATA: DB      "The Promise of World Peace",0
        END
```