Microprocessor
Ch.5 Addressing Modes

OUTLINE

- Immediate and register addressing modes
- Accessing memory using various addressing mode
- Bit addresses for I/O and RAM
- Extra 128-byte on-chip RAMs

IMMEDIATE AND REGISTER: ADDRESSING MODE

- **Addressing mode**
  - The CPU can access data in various ways.
    - E.g. the data can be in a register, in memory, or provided as immediate value (#34H).
  - There are 5 different addressing modes for 8051
    - Immediate (ch. 5.1)
    - Register (ch. 5.1)
    - Direct (ch. 5.2)
    - Register indirect (ch. 5.2)
    - Indexed (ch. 5.2)

IMMEDIATE AND REGISTER: IMMEDIATE

- **Immediate addressing mode**
  - The source operand is a constant
  - E.g.

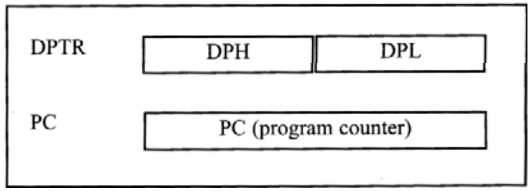
```
MOV A, #25H
MOV R3, #62
MOV DPTR, #4521H     ; DPTR is a 16-bit register
```

  - DPTR: (data pointer)
    - A 16-bit register, usually used to store ROM address (recall: PC is 16-bit)
    - High byte: DPH, low byte: DPL.

```
MOV DPH, #45H
MOV DPL, #21H   ; the same as MOV DPTR, #4521H
```

    - The following instructions are illegal

```
MOV DPTR, #9F235H   ; require more than 16-bit
MOV DPTR, #68975    ; FFFFH = 65535
```



IMMEDIATE AND REGISTER: IMMEDIATE

- **Immediate (Cont'd)**
  - Some special cases of immediate addressing mode
  - 1. Using the EQU directive

```
COUNT  EQU     25H
MOV    A, #COUNT        ; opcode: 7425H
```

    - The above two lines are exactly the same as MOV A, #25H (opcode: 7425H)
    - CPU doesn't know the existence of "COUNT EAU 25H" (pseudo-code)
    - The name COUNT is only used to improve program readability and programming efficiency.
  - 2. Using address labels

```
            MOV DPTR, #MYDATA     ; (DPTR) = 200H
            ORG 200H
MYDATA:     DB 23H, 35H
```

    - #MYDATA is the address of the contents 2335H in ROM
    - NOTE: MOV DPTR, MYDATA is illegal
  - 3. ASCII code

```
            MOV A, #'A'       ; the ASCII code of 'A' is loaded into register A.
```

IMMEDIATE AND REGISTER: REGISTER

- **Register addressing mode**
  - Use register to hold the data to be manipulated
  - Example

    MOV A, R0

    MOV R2, A

    MOV R7, DPL

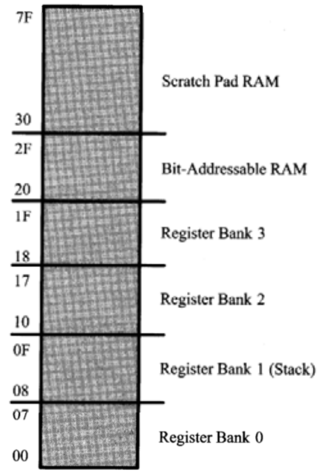
    MOV DPH, #23
  - Notes
    - MOV R2, R5 is invalid.
    - MOV A, DPTR is invalid (why?)

OUTLINE

- Immediate and register addressing modes
- **Accessing memory using various addressing mode**
- Bit addresses for I/O and RAM
- Extra 128-byte on-chip RAMs

ADDRESSING MEMORY: DIRECT

- **There are 3 different addressing mode to access memory**
  - Direct: use the address of the memory
  - Register indirect: use register to store RAM address
  - Indexed: use DPTR register to store ROM address
- **Direct addressing mode (used to access RAM)**
  - Review: RAM (128 bytes. Address 00H – 7FH)
    - 00H – 1FH (32 bytes, Register banks and stack)
    - 20H – 2FH (16 bytes, bit addressable space, will be discussed in Sect. 5.3)
    - 30H – 7FH (80 bytes, scratch pad, temporary store data)
  - Example 1 (demo memory address)



MOV R0, 40H ; move the RAM contents with address 40H into R0
; (R0) = (40H)
MOV R0, #40H ; move 40H into R0. (R0) = 40H

  - Example 2

MOV 40H, #0F3H ; (40H) = F3H
MOV A, 40H ; (A) = (40H) = F3H
MOV 35H, A ; (35H) = (A) = F3H
MOV 56H, 35H ; (56H) = (35H) = F3H
MOV #23H, 35H ; illegal

ADDRESSING MEMORY: DIRECT

- **Direct address mode for R0 – R7**
  - Register R0 – R7 can be access by either using their names (Rx) or their addresses
  - E.g. MOV A, 2 ; the same as MOV A, R2, but different from MOV A, #2

    MOV 7, 2 ; copy the contents of R2 to R7: (R7) = (R2)

    recall: MOV R7, R2 is invalid, but MOV 7, 2 is valid!
- **Direct address mode for SFR (special function registers)**
  - Special function registers: A, B, PSW, DPTR, P0, P1, P2, P3, ……
  - Each SFR has its own address in the range between 80H – FFH (recall: address range for 128-byte RAM is: 00H – 7FH)
  - The SFRs can either be accessed by their names (e.g. MOV A, #24H) or their addresses (e.g. MOV 0E0H, #24H)
  - The address of some commonly used SFRs
    - A: E0H, B: F0H, P0: 80H,
    - P1: 90H, P2: A0H, P3: B0H

      MOV 90H, A ; the same as MOV P1, A

      MOV 0F0H, R0 ; the same as MOV B, R0

ADDRESSING MODE: DIRECT

- **Stack and direct addressing mode**
  - One of the main application of direct addressing mode is for stack
  - The operand of PUSH and POP must be addresses instead of register names
    - PUSH *addr*
  - Example
    - PUSH A is illegal
    - PUSH 0E0H ; push the contents of register A into stack

```
PUSH 5        ; push R5 of register bank 0 into stack
PUSH 6        ; push R6 of register bank 0 into stack
PUSH 0E0H     ; push register A into stack
PUSH 0F0H     ; pop top of stack into register B
POP 2         ; pop top of stack into R2 of register bank 0
POP 15        ; pop top of stack into R5 of register bank 2.
```

ADDRESSING MODE: REGISTER INDIRECT

- **Register indirect addressing mode**
  - The register is used as a pointer to the data (similar to the pointer in C language)
    - The register stores the address of the data to be accessed
  - E.g.

```
MOV R0, #05H
MOV A, @R0        ; move the contents of RAM location whose address is
                  ; stored in R0 into A:  (A) = (05H), equivalent to MOV A, 05H
MOV A, R0         ; (A) = (R0) = 05H, equivalent to MOV A, #05H
```

  - If the data is in the uC (e.g. 128 bytes RAM), only R0 and R1 can be used for register indirect addressing mode
  - E.g. Find the contents in RAM and registers after each step

```
MOV 32H, #10H
MOV R0, 32H
MOV R1, #45H
MOV @R1, R0
MOV A, R1
MOV A, @R1
MOV @R0, A
```

  - Limitation: R0, R1 are 8-bit registers → register indirect mode can only access on chip RAM

ADDRESSING MODE: REGISTER INDIRECT

- **Why register indirect addressing mode?**
  - Makes it possible to access a group of data through loop
  - E.g. Write a program to copy a block of 10 bytes of data from RAM locations starting at 35H to RAM locations starting at 60H

```
        COUNT EQU 10
        MOV R0, #35H
        MOV R1, #60H
        MOV R3, #COUNT
BACK:   MOV A, @R0        ; (A) = (R0)
        MOV @R1, A        ; (R1) = (A)
        INC R0            ; increment (R0) by 1
        INC R1            ; increment (R1) by 1
        DJNZ R3, BACK
```

ADDRESSING MODE: INDEXED MODE

- **Indexed address mode**
  - Access a group of data stored in ROM using DPTR register (16-bit register)
    - Recall: register indirect mode: access a group of data stored in RAM using R0 and R1
  - Syntax: MOVC A, @A+DPTR ; (A) = (A+DPTR)
    - MOVC: move the data stored in ROM,
    - Recall; MOV can only move data in RAM
  - Example: Assume ROM space starting at 300H contains "ELEG". Write a program to transfer bytes into RAM locations starting at 50H. (Demo)

```
        MOV DPTR, #MYDATA     ; ROM pointer, or MOV DPTR, #300H
        MOV R0, #50H          ; RAM pointer
        MOV R2, #4            ; counter, 4 bytes
BACK:   CLR A
        MOVC A, @A+DPTR       ; move data from ROM to A
        MOV @R0, A            ; move data from A to RAM
        INC DPTR
        INC R0
        DJNZ R2, BACK

        ORG 300H
MYDATA: DB      "ELEG"
        END
```

ADDRESSING MODE: INDEXED MODE

- **MOVC**
  - It has only two possible instructions
    - MOVC A, @A+DPTR
    - MOVC A, @A+PC
  - Any other usage MOVC is not allowed, e.g
    - MOVC A, @DPTR ; invalid
    - MOVC A, @R0+DPTR ; invalid
    - MOVC A, #23H ; invalid
  - Data cannot be moved directly from ROM to RAM, or the other way around
    - Must use A as an intermediate register

ADDRESSING MODE: INDEXED MODE

- **Look up table (LUT)**
  - Use a table to store commonly used numbers
  - E.g. write a program to calculate x^2 for x in the rage in 0 to 9
    - It's computationally expensive to calculate x^2 in real time
    - Use a table to store the value of x^2

```
        ORG 0
        MOV DPTR, #TABLE     ; the ROM location of LUT
        MOV P1, #0FFH        ; set P1 as input
BACK:   MOV A, P1            ; read x from port 1
        MOVC A, @A+DPTR      ; find x^2 from LUT, move it to A
        MOV P2, A            ; send results to P2
        SJMP BACK

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

Index (x)	x^2
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81

OUTLINE

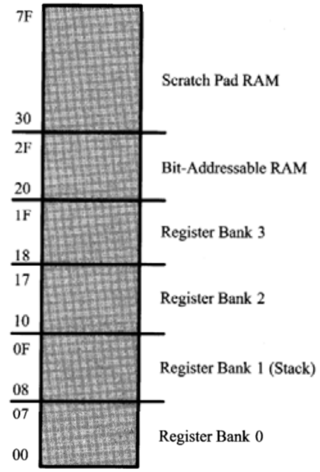
- Immediate and register addressing modes
- Accessing memory using various addressing mode
- **Bit addresses for I/O and RAM**
- Extra 128-byte on-chip RAMs

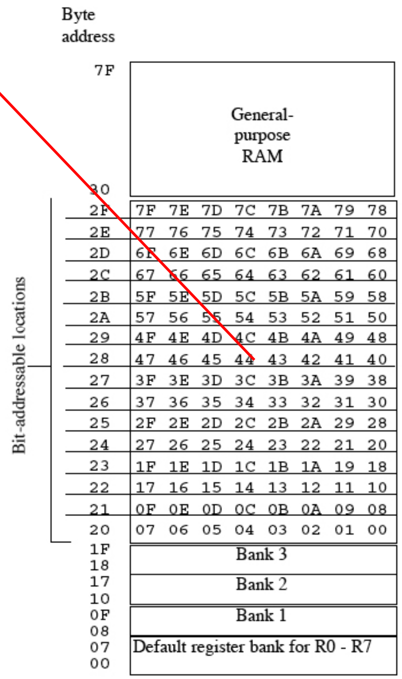
BIT ADDRESS: BIT-ADDRESSABLE RAM

- **Bit-addressable RAM**
  - RAM space 20H – 2FH (16 bytes = 128 bits) is bit-addressable
  - We can access each individual bit in RAM space 20H – 2FH
  - Bit address v.s. byte address
    - Bit address range: 00H – 7FH (128 bits)
    - Byte address range: 20H – 2FH (16 bytes)

    - Bit 00H: bit 0 of byte 20H
    - Bit 01H: bit 1 of byte 20H
    - …
    - Bit 07H: bit 7 of byte 20H
    - Bit 08H: bit 0 of byte 21H
    - ….
    - Bit 78H: bit 0 of byte 2FH
    - …
    - Bit 7FH: bit 7 of byte 2FH

How do we tell if an address is bit address or byte address? (e.g. 08H)





Byte address								
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 ADDRESS: BIT-ADDRESSABLE RAM

- **Bit-addressable RAM**
  - Bit addresses can only be used by bit addressable instructions

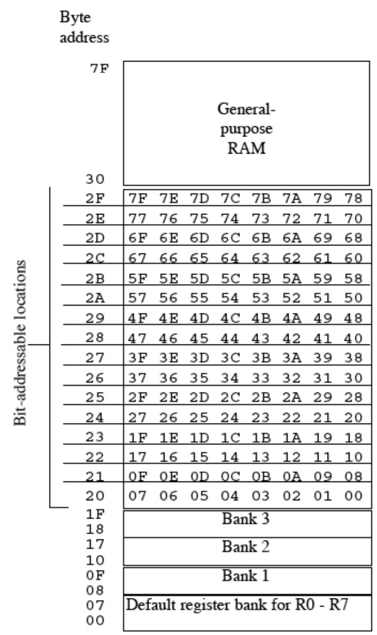
Instruction	Function
SETB bit	Set the bit (bit = 1)
CLR bit	Clear the bit (bit = 0)
CPL bit	Complement the bit (bit = NOT bit)
JB bit,target	Jump to target if bit = 1 (jump if bit)
JNB bit,target	Jump to target if bit = 0 (jump if no bit)
JBC bit,target	Jump to target if bit = 1, clear bit (jump if bit, then clear)
MOV bit, C	**move contents of carry flag to a bit**

    - If an address is used by the above bit instructions, then it's a bit address
    - Address used by all other instructions are byte address.
  - E.g. Find out which byte each of the following bit belongs

    SETB 42H
    CLR 0FH
    CPL 12

  - E.g. save the status of bit P1.7 to bit address 05

    SETB P1.7
    MOV C, P1.7
    MOV 05, C ;MOV 05, P1.7 is illegal



BIT ADDRESS: BIT MEMORY MAP

- **I/O port and some registers are bit addressable**
  - I/O port: P0, P1, P2, P3
    - E.g. P0.2, P1.4
  - Registers: A, B, PSW, IP, IE, ACC, SCON, TCON
    - E.g. PSW.3, A.4, ACC.0
  - Each addressable bit has its unique address
    - E.g. the bit address of P0.2 is 82H,
    - the bit address of PSW.3 is D3H
- **Bit address map**
  - 00H – 7FH (128 bits): bit addressable RAM
  - 80H – 87H (8 bits): P0
  - 88H – 8FH (8 bits): TCON
  - 90H – 9FH (8 bits): P1
  - …
  - D0H – D7H (8 bits): PSW
  - …
  - F0 – F7H (8 bits): B

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 -- -- AC AB AA A9 A8	IE
A0	A7 A6 A5 A4 A3 A2 A1 A0	P2
99	not bit-addressable	SBUF
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 Registers

BIT ADDRESS: DIRECTIVES

- **BIT directive**
  - Assign a name to a bit, improve code readability and code efficiency.
  - E.g.

```
SW BIT P2.3
LED BIT P0.1
MOV C, SW          ; the assembler will replace SW with the address of P2.3,
MOV LED, C         ; the assembler will replace LED with the address of P0.1
```

- **EQU directive**
  - EQU directive can also be used to assign name to bit. The assembler will determine if it's a bit address or byte address based on context
  - E.g.

```
SW       EQU    97H
MYDATA   EQU    0A0H

         MOV    C, SW             ; SW is a bit address
         MOV    MYDATA, #32H      ; MYDATA is a byte address
```

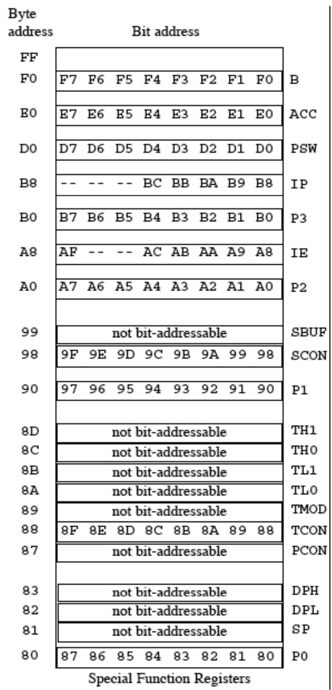
OUTLINE

- Immediate and register addressing modes
- Accessing memory using various addressing mode
- Bit addresses for I/O and RAM
- **Extra 128-byte on-chip RAMs**

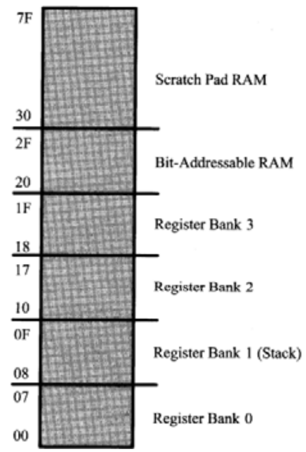
EXTRA RAM

- **Extra 128-byte RAM**
  - 8051 has 128 bytes on-chip RAM (address: 00H – 7FH)
  - 8052 has 256 bytes on-chip RAM
    - DS89C4x0 is 8052 compatible → it has 256 bytes RAM
- **Address map for chips with 256 bytes RAM**
  - The first 128 bytes: 00H – 7FH
  - The extra 128 bytes ("upper memory"): 80H – FFH
    - Problem: the address range 80H – FFH has already been assigned to SFR (e.g. A, B, PSW, DPTR, P0, P1, P2, P3, etc.)
  - Upper memory and SFR use the same address space!
    - Physically they are separate
  - How do we distinguish between upper memory and SFR?
    - To access SFR, we use direct addressing mode or register name
      - E.g. MOV 90H, #55H ; equivalent to MOV P1, #55H
    - To access upper memory, we use indirect addressing mode
      - E.g. MOV R0, 90H

        MOV @R0, #55H ; (90H) = 55H



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 -- -- AC AB AA A9 A8	IE
A0	A7 A6 A5 A4 A3 A2 A1 A0	P2
99	not bit-addressable	SBUF
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



EXTRA RAM

- **Display the contents of upper RAM in Keil**
  - In the memory panel, use I: (demo)
  - C: 80H (display ROM contents)
  - D: 20H (display RAM contents)
  - I: 80H (display upper RAM contents)