Microprocessor
Ch.6 Arithmetic and Logics

OUTLINE

- **Arithmetic instructions**
- **Signed number operations**
- **Logic and compare instructions**
- **Rotate instruction and data serialization**
- **BCD, ASCII**

AROTHMETIC: ADDC

- **ADDC (add with carry)**
  - ADDC A, *source*
  - (A) = (A) + source + CY
  - Destination must be register A!
  - Example: write a program to find the sum of 3CE7H and 3B8DH. Save the lower byte in R6, the higher byte in R7.

```
CLR C
MOV A, #0E7H
ADD A, #8DH       ; the sum of lower byte
MOV R6, A         ; lower byte in R6

MOV A, #3CH
ADDC A, #3BH      ; the sum of higher byte,
                  ; consider the carry from lower byte
MOV R7, A
```

ARITHMETIC: BCD

- **Binary coded decimal (BCD)**
  - Use 4-bit binary codes to represent decimal numbers. Totally 10 codes corresponding to decimal number 0 – 9.
  - Difference from binary number or hex number:
    - BCD contains only 10 codes
    - Any binary number larger than 9 (e.g. 1010) is not a BCD code.

Digit	*BCD*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- **Unpacked BCD**
  - Use a single byte (8 bits) to represent 1 BCD code (4 bits)
  - The high 4 bits are zeros, and low 4 bits are BCD. E.g. (00001000)
  - Convenient to store BCD in 8-bit registers.
  - E.g. unpacked BCD representation of decimal number 17: 0000 0001, 0000 0111
- **Packed BCD**
  - A single byte (8 bits) is used to represent 2 BCD codes.
  - E.g. packed BCD representation of decimal number 17: 0001 0111
  - A binary sequence can be used to represent either BCD or HEX.
    - E.g. 0001 0111 (BCD: 17 decimal) (HEX: 17H)
    - The programmer must know which code (BCD or HEX) is actually used.

ARITHMETIC: BCD ADDTION

- **Addition of BCD numbers**
  - Advantage of packed BCD: easy to use and read
    - E.g. Packed BCD of 39 decimal is 0011 1001
  - Disadvantage: the ADD instruction cannot be used for BCD numbers
    - E.g. 1. find 17+28 by using BCD

      MOV A, #17H

      ADD A, #28H

      - The result is 3F, the high 4-bit is BCD, the low 4-bit is not BCD.
      - To make it a BCD, add 6 to the low 4-bit: 3FH + 06H = 45H, which is the packed BCD for 45 decimal.
    - E.g. 2. find 52+87 by using BCD

      MOV A, #52H

      ADD A, #87H

      - The result is D9, the high 4-bit is BCD, the low 4-bit is not BCD
      - To make it a BCD, add 6 to the high 4 bit: D9H+60H = 139H
  - After summation, if 4-bit is not BCD, add 6 to it to adjust it to BCD.
    - This can be done automatically!

ARITHMETIC: DA

- **DA (decimal adjust for addition)**
  - Convert the sum of 2 BCD numbers to a BCD number
  - E.g. (demo)

```
MOV A, #47H
ADD A, #25H        ; (A) = 6CH
DA A               ; adjust for BCD addition, (A) = 72H
```

  - Operation of DA
    - If lower nibble is not BCD, or if AC = 1, add 0110 to lower nibble
    - If higher nibble is not BCD, or if CY = 1, add 0110 to lower nibble
  - Notes:
    - Can only be used for register A

ARITHMETIC: UNSIGNED NUMBER SUBTRACTION

- **SUBB (subtraction with borrow)**
  - SUBB A, *source* ; (A) = (A) – source – CY
  - Example: find 3FH – 23H

```
CLR C            ; (CY) = 0
MOV A, #3FH
SUBB A, #23H
```

  - For two unsigned numbers A and B, A – B = A + (2's complement of B)
  - Operations of SUBB
    - 1. Find the 2's complement of source
    - 2. find the summation of A and 2's complement of source
    - 3. Invert the carry

ARITHMETIC: UNSIGNED NUMBER SUBTRACTION

- **SUBB (Cont'd)**
  - Example: find 4CH – 6EH

```
CLR C
MOV A, #4CH
SUBB A, #6EH
```

  - If CY = 0 after SUBB, the result is positive;
  - If CY = 1 after SUBB, the result is negative, and reg. A contains the absolute value of the result in the form of 2's complement
  - Find the decimal value of the result in the above example
    - Perform 2's complement over a number twice yields the original number
  - Get 2's complement by using instructions

```
MOV A, #6EH
CPL A          ; 1's complement
INC A
```

ARITHEMETIC: UNSIGNED NUMBER SUBTRACTION

- **Multiple bytes subtraction (demo)**
  - Example: find 2762H – 1296H, store the lower byte in R7, higher byte in R6

```
CLR C
MOV A, #62H
SUBB A, #96H     ; 62H-96H = CCH with CY = 1
                 ; CY = 2 indicate there is a borrow
MOV R7, A
MOV A, #27H
SUBB A, #12H
MOV R6, A
```

  - Note:
    - In SUBB, the destination must be register A

AIRTHEMETIC: UNSIGNED MULTIPLICATION AND DIVISION

- **Multiplication**
  - MUL AB ; A * B, put the higher byte in B and lower byte in A
  - Example

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

- **Division**
  - DIV AB ; A/B, put quotient in A, and remainder in B
  - Example

```
MOV A, #95
MOV B, #10
DIV AB          ; A = 09, B = 05
```

  - If the denominator (B) is 0, OV =1 to indicate there is an error.

AIRTHEMETIC: UNSIGNED MULTIPLICATION AND DIVISION

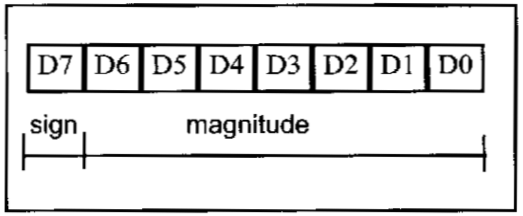
- **Example**
  - Converting a HEX number to decimal
    - Review: HEX to decimal conversion: keep divide hex by 10 and keep the remainder

```
MOV A, #0FDH      ; 253 decimal
MOV B, #10
DIV AB            ; 253/10, (A) = 25, (B) = 3
MOV R7, B         ; save lower digit
MOV B, #10
DIV AB            ; 25/10, (A) = 2, (B) = 5
MOV R6, B         ; save the next digit
                  ; quotient (2) is less than 10
MOV R5, A         ; save the last digit
```

OUTLINE

- Arithmetic instructions
- **Signed number operations**
- Logic and compare instructions
- Rotate instruction and data serialization
- BCD, ASCII

SIGNED NUMBER



- **Signed 8-bit number**
  - MSB represent sign: '1' negative, '0' positive
  - If positive, range is 0 to 127 (00000000 ~ 01111111)
  - If negative, use 2's complement to find absolute value
    - Example: a signed number is represented as 1111 1011, find its decimal value
  - If a number is negative: it's signed number representation = 2's complement of its absolute value
    - Example: find the signed number representation of -1 and -128
  - The range of 8-bit signed number is -128 – 127
    - -128: 1000 0000
    - -127: 1000 0001
    - …
    - -1: 1111 1111
    - 0: 0000 0000
    - 1: 0000 0001
    - …
    - 127: 0111 1111

SIGNED NUMBER: OVERFLOW

- **Overflow might happen during signed number operation**
  - Example

```
MOV A, #+96        ; 60H
MOV R1, #+70       ; 46H
ADD A, R1
```

  - The result is larger than +127 → overflow
  - The CPU will set OV = 1 to indicate overflow.
  - CPU will set OV to 1 in the following conditions
    - There is a carry from D6 to D7 but no carry out of D7 (CY = 0)
    - There is a carry from D7 out but no carry from D6 to D7
  - If there is carry from both D6 to D7 and D7 out, OV = 0

SIGNED NUMBER: OVERFLOW

- **Examples: find the OV flag in the following examples**
  - 1.
    ```
    MOV A, #-128
    MOV R4, #-2
    ADD A, R4
    ```
  - 2.
    ```
    MOV A, #-2
    MOV R1, #-5
    ADD A, R1
    ```
  - 3.
    ```
    MOV A, #+7
    MOV R1, #+18
    ADD A, R1
    ```

OUTLINE

- Arithmetic instructions
- Signed number operations
- **Logic and compare instructions**
- Rotate instruction and data serialization
- BCD, ASCII

LOGIC INSTRUCTIONS

- **ANL (and logic)**
  - ANL *destination, source* ; (dest) = (dest) AND (src)
  - Bit by bit AND operation
- **ORL (or logic)**
  - ORL *destination, source* ; (dest) = (dest) OR (src)
  - Bit by bit OR operation
- **XRL (xor logic)**
  - XRL destination, source ; (dest) = (dest) XOR (src)
  - Bit by bit XOR operation
- **Example**

```
MOV A, #15H
MOV R0, #3CH
ANL A, R0
XOR A, R0
ORL A, R0
```

COMPARE INSTRUCTIONS

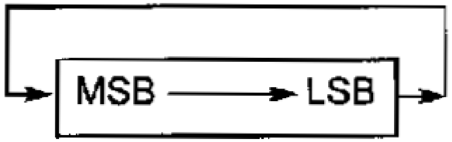
- **CJNE (compare and jump if not equal)**
  - CJNE *destination, source, target*
  - If destination $\neq$ source, jump to target
    - If destination >= source, set CY =0
    - If destination < source, set CY = 1
  - Example: assume P1 is connected to a temperature sensor. Write a program to continuously read the temperature and test it for the value of 75.
    - If T = 75, then A = 75; if T < 75, then R1 = T; if T > 75, then R2 = T
  - Self study: Example 6-27 (p.160)

OUTLINE

- Arithmetic instructions
- Signed number operations
- Logic and compare instructions
- **Rotate instruction and data serialization**
- BCD, ASCII

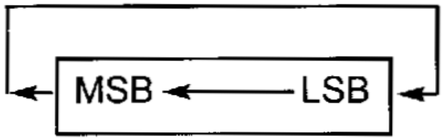
ROTATION

- **RR (rotate the bits to the right)**
  - RR A ; can only be used with register A
  - Cyclically rotate the bits of A to right
  - Example




```
MOV A, #36H
RR A
RR A
```

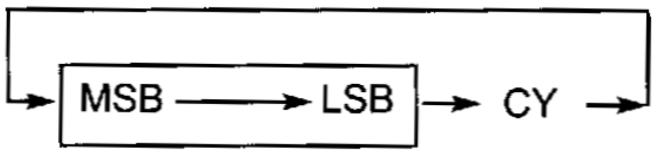
- **RL (rotate the bits to the left)**
  - RL A ; can only be used with register A
  - Cyclically rotate the bits of A to left
  - Example



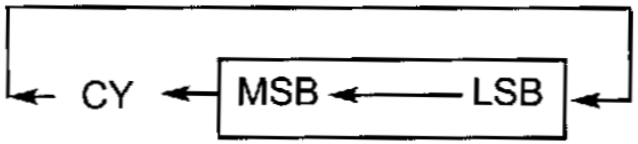

```
MOV A, #2CH
RL A
RL A
```

ROTATION

- **RRC (rotate right through carry)**
  - RRC A ; can only be used with A
  - Rotate right through carry



- **RLC (rotate left through carry)**
  - RLC A ; can only be used with A
  - Rotate left through carry

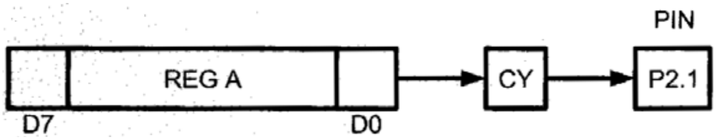


  - Example

```
CLR C
MOV A, #26H
RRC A
SETB C
MOV A, #15H
RRL A
```

ROTATION: SERIALIZING DATA

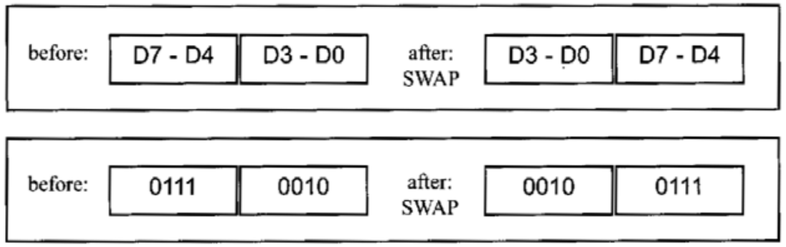
- **Serializing data**
  - Transfer data one bit at a time.
  - Example: write a program to transfer 41H serially via pin 2.1. Put two highs at the start and end of the data. Send the byte LSB first.




```
      MOV  A,#41H
      SETB P2.1       ;high
      SETB P2.1       ;high
      MOV  R5, #8
HERE: RRC  A
      MOV  P2.1,C     ;send the carry bit to P2.1
      DJNZ R5, HERE
      SETB P2.1       ;high
      SETB P2.1       ;high
```

ROTATION: SWAP

- **SWAP**
  - SWAP A   ; can only be used with A
  - Swap the lower nibble with the higher nibble



  - Example

MOV A, #23H
SWAP A

OUTLINE

- Arithmetic instructions
- Signed number operations
- Logic and compare instructions
- Rotate instruction and data serialization
- **BCD, ASCII**

BCD AND ASCII

- **Binary, BCD and ASCII**
  - Some devices use packed BCD to represent number
    - E.g. timer uses packed BCD to keep track of time
  - Some devices use binary number (hex) to represent number
    - E.g. sensor represent temperature in binary format
  - The display device usually accepts ASCII code
    - E.g. In order to display the character in LCD, we need to send the ASCII code to the display device
  - Usually we need to perform conversion between BCD, Binary, and ASCII

Key	ASCII (hex)	BCD (unpacked)
0	30	0000 0000
1	31	0000 0001
2	32	0000 0010
3	33	0000 0011
4	34	0000 0100
5	35	0000 0101
6	36	0000 0110
7	37	0000 0111
8	38	0000 1000
9	39	0000 1001

Unpacked BCD → ASCII: add 30H to unpacked BCD

ASCII → unpacked BCD: Mask out the upper nibble of ASCII

BCD AND ASCII: BCD → ASCII

- **Packed BCD to ASCII conversion**
  - Packed BCD → unpacked BCD → ASCII
  - From unpacked BCD to ASCII: add 30H to unpacked BCD
  - E.g.

Packed BCD	***Unpacked BCD***	***ASCII***
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 &	0011 0010 &
	0000 1001	0011 1001

  - E.g. Assume Reg. A has a packed BCD. Convert it to two ASCII codes and place them in R2 and R6.

```
MOV A, #29H        ; packed BCD
MOV R2, A          ; keep a copy of BCD in R2
ANL A, #0FH        ; low nibble → unpacked BCD
ADD A, #30H        ; unpacked BCD → ASCII
MOV R6, A          ; save the ASCII of low nibble to R6

MOV A, R2          ; get the original BCD
ANL A, #F0H        ; get high nibble
SWAP A             ; unpacked BCD
ADD A, #30H        ; unpacked BCD → ASCII
MOV R2, A
```

BCD AND ASCII: ASCII → BCD

- **Convert ASCII to packed BCD**

Key	*ASCII*	*Unpacked BCD*	*Packed BCD*
4	34	00000100	
7	37	00000111	01000111 or 47H

  - ASCII → unpacked BCD: mask upper nibble with 0
  - Unpacked BCD → packed BCD: combine two lower nibble to 1 byte
  - Example: convert the ASCII code "47" to a packed BCD

```
MOV A, #'4'
ANL A, #0FH      ; mask out upper nibble, unpacked BCD
SWAP A
MOV B, A         ; store results in B

MOV A, #'7'
ANL A, #0FH      ; mask out upper nibble, unpacked BCD

ORL A, B         ; combine two nibbles into one byte
```

BCD AND ASCII: LOOK UP TABLE FOR ASCII

- **Using look up table for ASCII**
  - Commonly used in interfacing with keypad and LCD
  - Example: P0.0, P0.1, P0.2 are connected to 3 switches. Write a program to send the ASCII code ‘0’, ‘1’, … ‘7’ to P2 based on the combination of the 3 switches.

```
            MOV DTPR, #MYDATA
            MOV A, P1               ; read switches
            ANL A, #07H             ; mask all but lower 3 bits
            MOVC A, @A+DPTR         ; A is the index into LUT
            MOV P2, A
            SJMP $                  ; stay here

            ORG 400H
MYDATA:     DB ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’
```

BCD AND ASCII: BINARY → ASCII

- **Binary to ASCII conversion**
  - Many analog-to-digital converter provide output data in binary (hex) format
  - To display the data, we need to convert it to ASCII
  - Two steps: 1. binary → unpacked BCD, 2. unpacked BCD → ASCII
  - Example:

```
            ;----- main program------
            ORG 0
            ACALL BIN_2_DEC
            ACLL DEC_2_ASCII

            ;-----BIN_2_DEC--------
BIN_2_DEC:
            MOV A, #235         ; (A) = 0EBH
            MOB B, #10
            DIV AB              ; (A) = 23, (B) = 5
            MOV R0, B           ; (R0) = 5 = 05H, unpacked BCD
            MOV B, #10
            DIV AB              ; (A) = 2, (B) = 3
            MOV R1, B           ; (R1) = 3 = 03H, unpacked BCD
            MOV R2, A           ; (R2) = 2 = 02H, unpacked BCD
            RET
```

BCD AND ASCII: BINARY → ASCII

- **Binary to ASCII conversion (Cont'd)**

```
                    ; ---------BCD 2 ASCII
DEC_2_ASCII:
                    MOV A, R0           ; (A) = 05H
                    ORL A, #30H         ; BCD → ASCII
                    MOV R0, A           ; (R0) = '5'

                    MOV A, R1           ; (A) = 03H
                    ORL A, #30H         ; BCD → ASCII
                    MOV R1, A           ; (R1) = '3'

                    MOV A, R2           ; (A) = 02H
                    ORL A, #30H         ; BCD → ASCII
                    MOV R2, A           ; (R2) = '2'
```

- **Self-study: checksum byte in ROM (p.170)**