

LESSON 3: ARITHMETIC INSTRUCTIONS

Objectives:

1. Study the operation of arithmetic instructions.

- ADD A,<byte>
- ADDC A,<byte>
- SUBB A,<byte>
- INC A
- INC <byte>
- INC DPTR
- DEC A
- DEC <byte>
- MUL AB
- DIV AB
- DA A

2. Use the Assembler, ASM51, learn Assembler control, and Assembler directive.

The arithmetic instructions are performed with accumulator. The operand can be register R0-R7 or on-chip memory with direct or indirect addressing mode. Most of them execute within one machine cycle, so for 12MHz oscillator, it will be 1 μ S.

ADD A, <byte>

The addressing modes are:

Direct	ADD A,30H
Indirect	ADD A,@R0
Register	ADD A,R7
Immediate	ADD A,#30H

EXERCISE 3-1: Enter the opcode shown below started from address 8100. Single step and answer the question.

ADDR	OPCODE			
8100	7530FE	9 main:	mov 30h,#254	; [30h] = ?
8103	E502	10	mov a,2	; A = ?
8105	2530	11	add a,30h	; A = ? CARRY FLAG = ?
8107	2430	12	add a,#30h	; A = ? CY = ?
8109	7F64	13	mov r7,#100	; R7 = ?
810B	2F	14	add a,r7	; A = ? CY = ?
810C	7831	15	mov r0,#31h	; R0 = ?

810E 27 16 add a,@r1 ; A = ? CY = ?

If the result of addition is larger than 255 or FFH, the carry flag will set. We can use ADDC instruction for higher significant byte adding. The higher significant byte will be added with carry flag.

ADDC A, <byte>

EXERCISE 3-2: Enter code below and single step.

ADDR OPCODE

```
8100 7530FF        11 main: mov 30h,#0ffh
8103 753100        12        mov 31h,#00
8106 753202        13        mov 32h,#02
8109 753300        14        mov 33h,#00
                  15
810C E530         16        mov a,30h     ; A = ?
810E 2532         17        add a,32h     ; A = ? CY = ?
8110 F530         18        mov 30h,a     ; [30H] = ?
8112 E531         19        mov a,31h     ; A = ?
8114 3533         20        addc a,33h    ; [33H] = ? CY = ?
```

1. The operand1 is 00FF and operand2 is 0002. Above program performs 00FF+0002 and saves the result to memory address 30h and 31h. The low address stores low significant byte. Try changing the operator1 and 2, 1EEF+A029, show the result by hand calculation and result that stores at location 30h and 31h.

SUBB A, <byte>

SUBB subtracts the byte variable and the borrow (carry) flag from the Accumulator, leaving result in A. Note, if state of carry flag is not known before doing single or multiple precision subtraction, we must clear carry flag using CLR C beforehand.

EXERCISE 3-3: Find the value after execution for each instruction.

ADDR OPCODE

```
8100 7464        11 main: mov a,#100
8102 9401        12        subb a,#1
```

1. Single step, what is the result in Accumulator.

Now suppose, the carry has set before executing SUBB, see what happen?

ADDR OPCODE

```
8100 D3      11  main: setb c ; suppose state of carry was 1
8101 7464    12      mov a,#100
8103 9401    13      subb a,#1
```

2. What is the result in Accumulator?

Above example shows us, better to clear carry (borrow) flag before the SUBB instruction.

3. Now, let have the operand1 and operand2 stored at location 30H, 31H and 32H, 33H. The same as ADDC exercise, except we will subtract it. Show the result by hand calculation and check it with single stepping.

```
INC A
INC <byte>
INC DPTR
DEC A
DEC <byte>
```

The INC instruction increments byte variable by one. The DEC instruction decrements by one. All of them have 8-bit variable, except INC DPTR, it will increment 16-bit DPTR register by one.

From now on we will learn using ASM51 if the exercise asked for writing small program. We can use the assembler to translate assembly source code into machine code, Intel Hex file. The Intel hex file can then be downloaded into 8051SBC's memory for program testing easily.

Study the source code below,

```
$mod51
$include(mypaulm2.equ)
```

```
    dseg at 30h
```

```
num1: ds 2
num2: ds 2
```

```
    cseg at 8000h
    jmp main
```

```
    org 8100h
```

```
main: mov num1,#0feh
```

```

mov num1+1,#12h
mov num2,#2eh
mov num2+1,#1fh

mov a,num1
add a,num2
mov num1,a

mov a,num1+1
addc a,num2+1
mov num2+1,a

jmp monitor

end

```

The first line, \$mod51 is assembler control. It will tell assembler to use the file name mod51 for EQU symbol. The actual content of mod51 is the definition of location for 8051 registers. Let us see its content.

```

P0  DATA 080H ;PORT 0
SP  DATA 081H ;STACK POINTER
DPL DATA 082H ;DATA POINTER - LOW BYTE
DPH DATA 083H ;DATA POINTER - HIGH BYTE
PCON DATA 087H ;POWER CONTROL
TCON DATA 088H ;TIMER CONTROL
TMOD DATA 089H ;TIMER MODE
TL0  DATA 08AH ;TIMER 0 - LOW BYTE
TL1  DATA 08BH ;TIMER 1 - LOW BYTE
TH0  DATA 08CH ;TIMER 0 - HIGH BYTE
TH1  DATA 08DH ;TIMER 1 - HIGH BYTE
P1   DATA 090H ;PORT 1
SCON DATA 098H ;SERIAL PORT CONTROL
SBUF DATA 099H ;SERIAL PORT BUFFER
P2   DATA 0A0H ;PORT 2
IE   DATA 0A8H ;INTERRUPT ENABLE
P3   DATA 0B0H ;PORT 3
IP   DATA 0B8H ;INTERRUPT PRIORITY
PSW  DATA 0D0H ;PROGRAM STATUS WORD
ACC  DATA 0E0H ;ACCUMULATOR
B    DATA 0F0H ;MULTIPLICATION REGISTER
IT0  BIT 088H ;TCON.0 - EXT. INTERRUPT 0 TYPE
IE0  BIT 089H ;TCON.1 - EXT. INTERRUPT 0 EDGE FLAG
IT1  BIT 08AH ;TCON.2 - EXT. INTERRUPT 1 TYPE
IE1  BIT 08BH ;TCON.3 - EXT. INTERRUPT 1 EDGE FLAG

```

TR0 BIT 08CH ;TCON.4 - TIMER 0 ON/OFF CONTROL
 TF0 BIT 08DH ;TCON.5 - TIMER 0 OVERFLOW FLAG
 TR1 BIT 08EH ;TCON.6 - TIMER 1 ON/OFF CONTROL
 TF1 BIT 08FH ;TCON.7 - TIMER 1 OVERFLOW FLAG
 RI BIT 098H ;SCON.0 - RECEIVE INTERRUPT FLAG
 TI BIT 099H ;SCON.1 - TRANSMIT INTERRUPT FLAG
 RB8 BIT 09AH ;SCON.2 - RECEIVE BIT 8
 TB8 BIT 09BH ;SCON.3 - TRANSMIT BIT 8
 REN BIT 09CH ;SCON.4 - RECEIVE ENABLE
 SM2 BIT 09DH ;SCON.5 - SERIAL MODE CONTROL BIT 2
 SM1 BIT 09EH ;SCON.6 - SERIAL MODE CONTROL BIT 1
 SM0 BIT 09FH ;SCON.7 - SERIAL MODE CONTROL BIT 0
 EX0 BIT 0A8H ;IE.0 - EXTERNAL INTERRUPT 0 ENABLE
 ET0 BIT 0A9H ;IE.1 - TIMER 0 INTERRUPT ENABLE
 EX1 BIT 0AAH ;IE.2 - EXTERNAL INTERRUPT 1 ENABLE
 ET1 BIT 0ABH ;IE.3 - TIMER 1 INTERRUPT ENABLE
 ES BIT 0ACH ;IE.4 - SERIAL PORT INTERRUPT ENABLE
 EA BIT 0AFH ;IE.7 - GLOBAL INTERRUPT ENABLE
 RXD BIT 0B0H ;P3.0 - SERIAL PORT RECEIVE INPUT
 TXD BIT 0B1H ;P3.1 - SERIAL PORT TRANSMIT OUTPUT
 INT0 BIT 0B2H ;P3.2 - EXTERNAL INTERRUPT 0 INPUT
 INT1 BIT 0B3H ;P3.3 - EXTERNAL INTERRUPT 1 INPUT
 T0 BIT 0B4H ;P3.4 - TIMER 0 COUNT INPUT
 T1 BIT 0B5H ;P3.5 - TIMER 1 COUNT INPUT
 WR BIT 0B6H ;P3.6 - WRITE CONTROL FOR EXT. MEMORY
 RD BIT 0B7H ;P3.7 - READ CONTROL FOR EXT. MEMORY
 PX0 BIT 0B8H ;IP.0 - EXTERNAL INTERRUPT 0 PRIORITY
 PT0 BIT 0B9H ;IP.1 - TIMER 0 PRIORITY
 PX1 BIT 0BAH ;IP.2 - EXTERNAL INTERRUPT 1 PRIORITY
 PT1 BIT 0BBH ;IP.3 - TIMER 1 PRIORITY
 PS BIT 0BCH ;IP.4 - SERIAL PORT PRIORITY
 P BIT 0D0H ;PSW.0 - ACCUMULATOR PARITY FLAG
 OV BIT 0D2H ;PSW.2 - OVERFLOW FLAG
 RS0 BIT 0D3H ;PSW.3 - REGISTER BANK SELECT 0
 RS1 BIT 0D4H ;PSW.4 - REGISTER BANK SELECT 1
 F0 BIT 0D5H ;PSW.5 - FLAG 0
 AC BIT 0D6H ;PSW.6 - AUXILIARY CARRY FLAG
 CY BIT 0D7H ;PSW.7 - CARRY FLAG

For example the DATA directive tells assembler that P1 symbol is located at address 090H on chip memory.

```
P1 DATA 090H ;PORT 1
```

Another example is BIT directive. It tells assembler that Carry Flag BIT location is at 0D7H or bit 7 of PSW register.

CY BIT 0D7H ;PSW.7 - CARRY FLAG

With \$mod51, or \$mod52 typed in the source code, we can then use the symbol of register, instead of the putting the hex address directly.

The second assembler control is \$include(file). It tells the assembler to include any text files in the source code. The example shown below includes file name, mypaulm2.equ to the source code.

```
$include(mypaulm2.equ)
```

Let us see what is the content of mypaulm2.equ.

```
phex1 equ 002Eh ;print a single hex digit
cout equ 0030h ;Send Acc to serial port
cin equ 0032h ;Get Acc from serial port
phex equ 0034h ;Print Hex value of Acc
phex16 equ 0036h ;Print Hex value of DPTR
pstr equ 0038h ;Print string pointed to by DPTR,
                ;must be terminated by 0 or a high bit set
                ;pressing ESC will stop the printing
ghex equ 003Ah ;Get Hex input into Acc
                ;Carry set if ESC has been pressed
ghex16 equ 003Ch ;Get Hex input into DPTR
                ;Carry set if ESC has been pressed
esc equ 003Eh ;Check for ESC key
                ;Carry set if ESC has been pressed
upper equ 0040h ;Convert Acc to uppercase
                ;Non-ASCII values are unchanged
autobaud equ 0042h ;Initialize serial port
pcstr equ 0045h ;print string in compressed format (no docs)
newline equ 0048h ;print CR/LF (13 and 10)
lenstr equ 004Ah ;return the length of a string @DPTR (in R0)
pint8u equ 004Dh ;print Acc at an integer, 0 to 255
pint8 equ 0050h ;print Acc at an integer, -128 to 127
pint16u equ 0053h ;print DPTR as an integer, 0 to 65535
smart_wr equ 0056h ;write Acc @DPTR (RAM or flash), C=1 if error
prgm equ 0059h ;write Acc @DPTR to Flash ROM, C=1 if error
erall equ 005Ch ;erase the Flash ROM chip, C=1 if error
find equ 005Fh ;find next prog header in memory
cin_filter equ 0062 ;like cin, but scan for arrow keys, pgup/pgdn
asc2hex equ 0065h ;convert character 0-9,A-F to number 0-15.

monitor equ 1954h ; return path to monitor program see myEXTRA.asm
```

We see that each line tells the assembler what is the value of a given symbol using EQU directive. For example,

```
phex16 equ 0036h ;Print Hex value of DPTR
```

phex16 is equal 0036h, the comment details what is phex16 and how to use it. The monitor program provides such useful subroutines for easy programming. Suppose we want to print the value of DPTR to terminal, we can just load the 16-bit value into DPTR and call phex16.

```
ADDR OP CODE
8100 909000    70  main: mov dptr,#9000H
8103 120036    71          call phex16
8106 021954    72          jmp monitor
```

EXERCISE 3-4: Enter above code and test run with jump command. See what happen on terminal screen?

The area of source code can switch between CODE segment of DATA segment easily with assembler directives, i.e., CSEG and DSEG respectively.

Suppose we want to reserve memory space for variable, we can then use directive DS (Define Storage). The example define num1 and num2 to be 2-byte variable

```
dseg at 30h
```

```
num1: ds 2
num2: ds 2
```

EXERCISE 3-5: Edit the source code below, use ASM51 to convert it to Hex file and List file. Answer questions.

```
$mod51
#include(mypaulm2.equ)
```

```
dseg at 30h
```

```
num1: ds 2      ; num1 = ?
num2: ds 2      ; num2 = ?
```

```
cseg at 8000h
jmp main
```

```
org 8100h
```

```

main:  mov num1,#0feh
       mov num1+1,#12h
       mov num2,#2eh
       mov num2+1,#1fh

       mov a,num1
       add a,num2
       mov num1,a

       mov a,num1+1
       addc a,num2+1
       mov num2+1,a

       jmp monitor

       end

```

1. What is the result of addition that stored in memory location 30H and 31H?

EXERCISE 3-6: Instead of using command ‘I’ to dump internal memory after addition, try with phex16 subroutine to print result on screen. (We can access 16-bit DPTR with 8-bit DPH and 8-bit DHL registers)

MUL AB **DIV AB**

MUL AB multiplies the unsigned 8-bit integers in the Accumulator and register B. The low-order byte of the 16-bit product is left in the Accumulator, and high order byte in B.

DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of quotient; register B receives remainder.

EXERCISE 3-7 Write a program that performs multiplication with MUL instruction. Print the 16-bit result on screen. Let A=255, B = 5, result will be in DPTR = A X B.

DA A

DA A adjusts the 8-bit value in the Accumulator to BCD number. The input value to be added must be BCD number. DA A must be placed after ADD or ADDC only.

EXERCISE 3-8 Write a program that performs addition between two BCD number, print the result on screen. Suppose the BCD number1 is 1924 and BCD number2 is 1892. Both are stored at internal ram address 30H-31H for BCD1 and 32H-33H for BCD2