Principle and Interface Techniques of Microcontroller

--8051 Microcontroller and Embedded Systems Using Assembly and C

LI, Guang (李光) Prof. PhD, DIC, MIET

WANG, You (王酉) PhD, MIET

杭州 • 浙江大学 • 2022

Chapter 6

Arithmetic & Logic Instructions and Programs

Addition of Unsigned Numbers

ADD A, source ;A = A + source

- The instruction ADD is used to add two operands
  - Destination operand is always in register A
  - Source operand can be a register, immediate data, or in memory
  - Memory-to-memory arithmetic operations are never allowed in 8051 Assembly language

Show how the flag register is affected by the following instruction.

```
MOV A,#0F5H     ;A=F5 hex
ADD A,#0BH      ;A=F5+0B=00
```

Solution:

```
    F5H          1111  0101
+   0BH        + 0000  1011
  ------        -----------
   100H          0000  0000
```

CY =1, since there is a carry out from D7

P（PSW.0）=0, because the number of 1s is zero (an even number), PF is set to 0.

AC =1, since there is a carry from D3 to D4

Addition of Individual Bytes

Assume that RAM locations 40 – 44H have the following values. Write a program to find the sum of the values. At the end of the program, register A should contain the low byte and R7 the high byte.

40 = (7D)
41 = (EB)
42 = (C5)
43 = (5B)
44 = (30)

Solution:

```
        MOV R0,#40H       ;load pointer
        MOV R2,#5         ;load counter
        CLR A             ;A=0
        MOV R7,A          ;clear R7
AGAIN:  ADD A,@R0         ;add the byte ptr to by R0
        JNC NEXT          ;if CY=0 don't add carry
        INC R7            ;keep track of carry
NEXT:   INC R0            ;increment pointer
        DJNZ R2,AGAIN     ;repeat until R2 is zero
```

ADDC and Addition of 16-Bit Numbers

- When adding two 16-bit data operands, the propagation of a carry from lower byte to higher byte is concerned

```
          1
         3C   E7
   +     3B   8D
         78   74
```

When the first byte is added (E7+8D=74, CY=1).
The carry is propagated to the higher byte, which result in 3C+ 3B + 1 =78 (all in hex)

Write a program to add two 16-bit numbers. Place the sum in R7 and R6; R6 should have the lower byte.

Solution:

```
        CLR  C              ;make CY=0
        MOV  A, #0E7H       ;load the low byte now A=E7H
        ADD  A, #8DH        ;add the low byte
        MOV  R6, A          ;save the low byte sum in R6
        MOV  A, #3CH        ;load the high byte
        ADDC A, #3BH        ;add with the carry
        MOV  R7, A          ;save the high byte sum
```

DA Instruction

The binary representation of the digits 0 to 9 is called BCD (Binary Coded Decimal)

```
        DA    A          ;decimal adjust for
addition
```

- **The DA instruction is provided to correct the aforementioned problem associated with BCD addition**

The DA instruction will add 6 to the lower nibble or higher nibble if need

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

Example:

```
MOV  A,#47H      ;A=47H first BCD operand
MOV  B,#25H      ;B=25H second BCD operand
ADD  A,B         ;hex(binary) addition(A=6CH)
DA A             ;adjust for BCD addition (A=72H)
```

The “DA” instruction works only on A. In other word, while the source can be an operand of any addressing mode, the destination must be in register A in order for DA to work.

DA works only after an ADD, but not after INC

DA Instruction

Summary of DA instruction

- After an ADD or ADDC instruction
  1. If the lower nibble (4 bits) is greater than 9, or if AC=1, add 0110 to the lower 4 bits
  2. If the upper nibble is greater than 9, or if CY=1, add 0110 to the upper 4 bits

Example:

```
      HEX          BCD
      29           0010 1001
    + 18         + 0001 1000
      41           0100 0001   AC=1
    +  6         +      0110
      47           0100 0111
```

Since AC=1 after the addition, "DA A" will add 6 to the lower nibble.
The final result is in BCD format.

Assume that 5 BCD data items are stored in RAM locations starting at 40H, as shown below. Write a program to find the sum of all the numbers. The result must be in BCD.

40=(71)
41=(11)
42=(65)
43=(59)
44=(37)

Solution:

```
        MOV  R0,#40H      ;Load pointer
        MOV  R2,#5        ;Load counter
        CLR  A            ;A=0
        MOV  R7,A         ;Clear R7
AGAIN:  ADD  A,@R0        ;add the byte pointer to by R0
        DA   A            ;adjust for BCD
        JNC NEXT          ;if CY=0 don't accumulate carry
        INC R7            ;keep track of carries
NEXT:   INC R0            ;increment pointer
        DJNZ R2,AGAIN     ;repeat until R2 is 0
```

Subtraction of Unsigned Numbers

- In many microprocessor there are two different instructions for subtraction:
  SUB and SUBB (subtract with borrow)
  - In the 8051 we have only SUBB
  - The 8051 uses adder circuitry to perform the subtraction

```
SUBB A,source        ;A = A – source – CY
```

- To make SUB out of SUBB, we have to make CY=0 prior to the execution of the instruction
  - Notice that we use the CY flag for the borrow

- SUBB when CY = 0
  1. Take the 2's complement of the subtrahend (source operand)
  2. Add it to the minuend (A)
  3. Invert the carry

```
        CLR     C
        MOV     A,#4C    ;load A with value 4CH
        SUBB    A,#6EH   ;subtract 6E from A
        JNC     NEXT     ;if CY=0 jump to NEXT
        CPL     A   ;if CY=1, take 1's complement
        INC     A   ;and increment to get 2's comp
NEXT:   MOV     R1,A     ;save A in R1
```

Solution:

```
     4C      0100 1100        0100 1100
  -  6E      0110 1110        1001 0010
    -22                      01101 1110
```

① 2's complement

② +

③ Invert carry

CY =1

CY=0, the result is positive;
CY=1, the result is negative and the destination has the 2's complement of the result

Subtraction of Unsigned Numbers

- SUBB when CY = 1
  This instruction is used for multi-byte numbers and will take care of the borrow of the lower operand

A = 62H – 96H – 0 = CCH
CY = 1

```
CLR     C
MOV     A,#62H          ;A=62H
SUBB    A,#96H          ;62H-96H=CCH with CY=1
MOV     R7,A            ;save the result
MOV     A,#27H          ;A=27H
SUBB    A,#12H          ;27H-12H-1=14H
MOV     R6,A            ;save the result
```

Solution:

A = 27H - 12H - 1 = 14H
CY = 0

We have 2762H - 1296H = 14CCH.

Unsigned Multiplication

- The 8051 supports byte by byte multiplication only
  The byte are assumed to be unsigned data

```
MUL   AB     ;AxB, 16-bit result in B, A
```

```
MOV  A, #25H      ;load 25H to reg. A
MOV  B, #65H      ;load 65H to reg. B
MUL  AB           ;25H * 65H = E99 where
                  ;B = OEH and A = 99H
```

Unsigned Multiplication Summary (MUL AB)

Multiplication	Operand1	Operand2	Result
Byte x byte	A	B	B = high byte A = low byte

Unsigned Division

- The 8051 supports byte over byte division only
  The byte are assumed to be unsigned data
  DIV AB ;divide A by B, A/B

```
MOV  A, #95    ;load 95 to reg. A
MOV  B, #10    ;load 10 to reg. B
DIV  AB        ;A = 09(quotient) and
               ;B = 05(remainder)
```

Unsigned Division Summary (DIV AB)

Division	Numerator	Denominator	Quotient	Remainder
Byte / byte	A	B	A	B

CY is always 0
If B ≠ 0, OV = 0
If B = 0, OV = 1 indicates error

Application for DIV

(a) Write a program to get hex data in the range of 00 – FFH from port 1 and convert it to decimal. Save it in R7, R6 and R5.

(b) Assuming that P1 has a value of FDH for data, analyze program.

Solution:

(a)

```
MOV  A, #0FFH
MOV  P1, A        ;make P1 an input port
MOV  A, P1        ;read data from P1
MOV  B, #10       ;B=0A hex
DIV  AB           ;divide by 10
MOV  R7, B        ;save lower digit
MOV  B, #10
DIV  AB           ;divide by 10 once more
MOV  R6, B        ;save the next digit
MOV  R5, A        ;save the last digit
```

(b) To convert a binary (hex) value to decimal, we divide it by 10 repeatedly until the quotient is less than 10. After each division the remainder is saves.

	Q	**R**
FD/0A =	19	3 (low digit)
19/0A =	2	5 (middle digit)
		2 (high digit)

Therefore, we have FDH=253.

Signed 8-bit Operands

- D7 (MSB) is the sign and D0 to D6 are the magnitude of the number

  If D7=0, the operand is positive, and if D7=1, it is negative

- Positive numbers are 0 to +127
- Negative number representation (2's complement)
  1. Write the magnitude of the number in 8-bit binary (no sign)
  2. Invert each bit
  3. Add 1 to it

Overflow Problem

- If the result of an operation on signed numbers is too large for the register.

  An overflow has occurred and the programmer must be noticed.

Examine the following code and analyze the result.

```
MOV A,#+96      ;A=0110 0000 (A=60H)
MOV R1,#+70     ;R1=0100 0110(R1=46H)
ADD A,R1        ;A=1010 0110
                ;A=A6H=-90,INVALID
```

Solution:

	+96	0110 0000	
+	+70	0100 0110	
+	166	1010 0110	and OV =1

According to the CPU, the result is -90, which is wrong.
The CPU sets OV=1 to indicate the overflow

OV Flag

- In 8-bit signed number operations, OV is set to 1 if either occurs:
  1. There is a carry from D6 to D7, but no carry out of D7 (CY=0)
  2. There is a carry from D7 out (CY=1), but no carry from D6 to D7

```
MOV  A, #-128        ;A=1000 0000(A=80H)
MOV R4, #-2          ;R4=1111 1110(R4=FEH)
ADD  A, R4           ;A=0111 1110(A=7EH=
                      +126,INVALID)
```

	-128	1000 0000	
+	-2	1111 1110	
	-130	0111 1110	and OV=1

OV = 1
The result +126 is wrong

OV Flag

MOV A,#-2 ;A=1111 1110(A=FEH)
MOV R1,#-5 ;R1=1111 1011(R1=FBH)
ADD A, R1 ;A=1111 1001(A=F9H=-7, Correct, OV=0)

	-2	1111 1110
+	-5	1111 1011
	-7	1111 1001 and OV=0

OV = 0
The result -7 is correct

MOV A,#+7 ;A=0000 0111(A=07H)
MOV R1,#+18 ;R1=0001 0010(R1=12H)
ADD A,R1 ;A=0001 1001(A=19H=+25, Correct,OV=0)

	7	0000 0111
+	18	0001 0010
	25	0001 1001 and OV=0

OV = 0
The result +25 is correct

OV Flag

- In unsigned number addition, we must monitor the status of CY (carry)
  Use JNC or JC instructions
- In signed number addition, the OV (overflow) flag must be monitored by the programmer
- JB PSW.2 or JNB PSW.2

2's Complement

- To make the 2’s complement of a number

```
CPL   A          ;1’s complement (invert)
ADD   A, #1      ;add 1 to make 2’s comp.
```

Logic and Compare Instructions

ANL destination ,source ;dest = dest AND source

- This instruction will perform a logic AND on the two operands and place the result in the destination
  The destination is normally the accumulator
  The source operand can be a register, in memory, or immediate

Show the results of the following.

```
MOV  A, #35H     ;A = 35H
ANL  A, #0FH     ;A = A AND 0FH
```

35H	0 0 1 1 0 1 0 1
0FH	$\boxed{0\ 0\ 0\ 0}$ 1 1 1 1
05H	0 0 0 0 0 1 0 1

ANL is often used to mask (set to 0) certain bits of an operand

OR

ORL destination, source ;dest = dest OR source

- The destination and source operands are ORed and the result is placed in the destination
  - The destination is normally the accumulator
  - The source operand can be a register, in memory, or immediate

Show the results of the following.

```
MOV  A, #04H        ;A = 04
ORL  A, #68H        ;A = 6C
```

04H	0 0 0 0 0 1 0 0
68H	0 1 1 0 1 0 0 0
6CH	0 1 1 0 1 1 0 0

ORL instruction can be used to set certain bits of an operand to 1

XOR

XRL destination, source ;dest = dest XOR source

- This instruction will perform XOR operation on the two operands and place the result in the destination

The destination is normally the accumulator.
The source operand can be a register, in memory, or immediate.
XOR can be used to see if two registers have the same value.

Show the r

M
X

54H
78H
2CH

XRL A, Rn

XRL A, direct

XRL A, @Ri

XRL A, #data

XRL direct, A

XRL direct, #data

tion can be
gle certain
perand

Complement Accumulator

CPL A ;complements the register A

- This is called 1's complement

```
MOV  A, #55H
CPL  A          ;now A=AAH
                ;0101 0101(55H)
                ;becomes 1010 1010(AAH)
```

- To get the 2's complement, all we have to do is to add 1 to the 1's complement

Compare Instruction

CJNE destination, source, rel. addr.

- The actions of comparing and jumping are combined into a single instruction called CJNE (compare and jump if not equal)
- ✓ The CJNE instruction compares two operands, and jumps if they are not equal
- ✓ The destination operand can be in the accumulator or in one of the Rn registers
- ✓ The source operand can be in a register, in memory, or immediate

  The operands themselves remain unchanged
- ✓ It changes the CY flag to indicate if the destination operand is larger or smaller

```
        CJNE  R5, #80, NOT_EQUAL      ;check R5 for 80
          ...                         ;R5 = 80
NOT_EQUAL:
        JNC NEXT                      ;jump if R5 > 80
          ...                         ;R5 < 80
NEXT: ...
```

Compare	Carry Flag
destination ≥ source	CY = 0
destination < source	CY = 1

CY flag is always checked for cases of greater or less than, but only after it is determined that they are not equal

- Notice in the CJNE instruction that any Rn register can be compared with an immediate value

There is no need for register A to be involved

- The compare instruction is really a subtraction, except that the operands remain unchanged
  - Flags are changed according to the execution of the SUBB instruction

Write a program to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following.

If T = 75 then A = 75
If T < 75 then R1 = T
If T > 75 then R2 = T

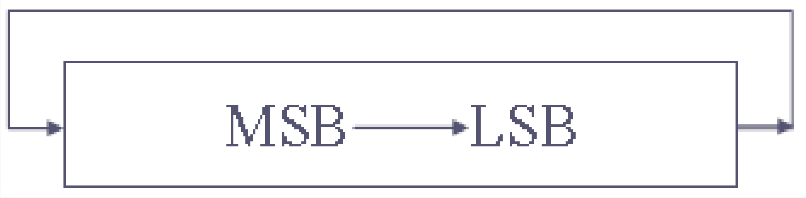
Solution:

```
        MOV   P1,#0FFH     ;make P1 an input port
        MOV   A,P1         ;read P1 port
        CJNE  A,#75,OVER   ;jump if A is not 75
        SJMP  EXIT         ;A=75, exit
OVER:   JNC   NEXT         ;if CY=0 then A>75
        MOV   R1,A         ;CY=1, A<75, save in R1
        SJMP  EXIT         ; and exit
NEXT:   MOV   R2,A         ;A>75, save it in R2
EXIT:   ...
```

Rotate Instruction and Data Serialization

```
RR    A              ;rotate right A
```

- In rotate right
  - ✓ The 8 bits of the accumulator are rotated right one bit, and
  - ✓ Bit D0 exits from the LSB and enters into MSB, D7

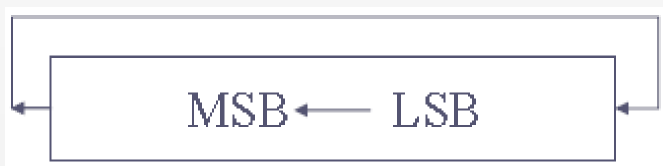



```
MOV  A, #36H     ;A = 0011 0110
RR   A           ;A = 0001 1011
RR   A           ;A = 1000 1101
RR   A           ;A = 1100 0110
RR   A           ;A = 0110 0011
```

Rotate Instruction and Data Serialization

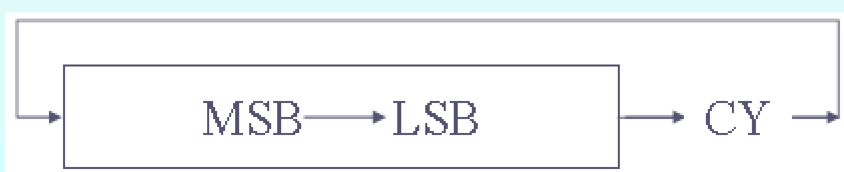
RL A ;rotate left A

- In rotate left
  - The 8 bits of the accumulator are rotated left one bit, and
  - Bit D7 exits from the MSB and enters into LSB, D0



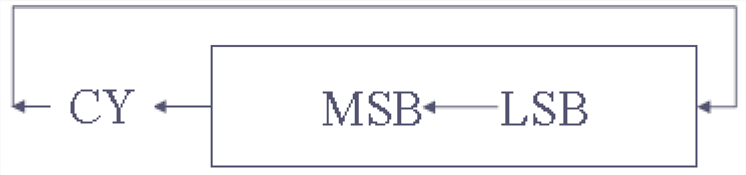
RRC A ;rotate right through carry

- In RRC A
  - Bits are rotated from left to right
  - They exit the LSB to the carry flag, and the carry flag enters the MSB



RLC A ;rotate left through carry

- In RLC A
  - ✓ Bits are shifted from right to left
  - ✓ They exit the MSB and enter the carry flag, and the carry flag enters the LSB



Write a program that finds the number of 1s in a given byte.

```
        MOV  R1, #0
        MOV R7, #8          ;count=08
        MOV A, #97H
AGAIN:  RLC A
        JNC NEXT            ;check for CY
        INC R1              ;if CY=1 add to count
NEXT:   DJNZ R7, AGAIN
```

Serializing Data

- Serializing data is a way of sending a byte of data one bit at a time through a single pin of microcontroller
  - ✓ Using the serial port, discussed in Chapter10
  - ✓ To transfer data one bit at a time and control the sequence of data and spaces in between them

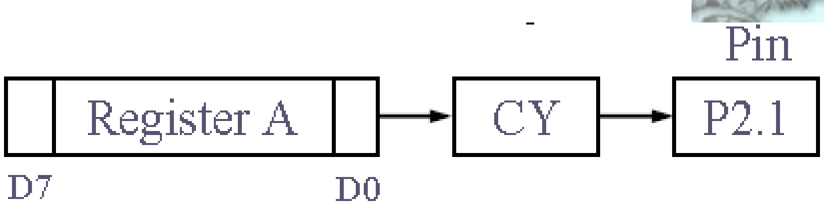
Serializing Data

- Transfer a byte of data serially by
  - ✓ Moving CY to any pin of ports P0 – P3
  - ✓ Using rotate instruction

Write a program to transfer value 41H serially (one bit at a time) via pin P2.1. Put two highs at the start and end of the data. Send the byte LSB first.

Solution:

```
        MOV A,#41H
        SETB P2.1      ;high
        SETB P2.1      ;high
        MOV R5,#8
AGAIN:  RRC A
        MOV P2.1,C     ;send CY to P2.1
        DJNZ R5, AGAIN
        SETB P2.1      ;high
        SETB P2.1      ;high
```



Single-bit Operations with CY

- There are several instructions by which the CY flag can be manipulated directly

Instruction	Function
SETB C	Make CY = 1
CLR C	Clear carry bit (CY = 0)
CPL C	Complement carry bit
MOV b,C	Copy carry status to bit location (CY = b)
MOV C,b	Copy bit location status to carry (b = CY)
JNC target	Jump to target if CY = 0
JC target	Jump to target if CY = 1
ANL C,bit	AND CY with bit and save it on CY
ANL C,/bit	AND CY with inverted bit and save it on CY
ORL C,bit	OR CY with bit and save it on CY
ORL C,/bit	OR CY with inverted bit and save it on CY

Assume that bit P2.2 is used to control an outdoor light and bit P2.5 a light inside a building. Show how to turn on the outside light and turn off the inside one.

Solution:

```
        SETB C          ;CY = 1
        ORL C,P2.2      ;CY = P2.2 ORed w/ CY
        MOV P2.2,C      ;turn it on if not on
        CLR C           ;CY = 0
        ANL C,P2.5      ;CY = P2.5 ANDed w/ CY
        MOV P2.5,C      ;turn it off if not off
```

Write a program that finds the number of 1s in a given byte.

Solution:

```
        MOV  R1,#0          ;R1 keeps number of 1s
        MOV R7,#8           ;counter, rotate 8 times
        MOV  A,#97H         ;find number of 1s in 97H
AGAIN:  RLC  A              ;rotate it thru CY
        JNC  NEXT           ;check CY
        INC  R1             ;if CY=1, inc count
NEXT:   DJNZ  R7,AGAIN      ;go thru 8 times
```

SWAP

SWAP A

- ➢ It swaps the lower nibble and the higher nibble

✓In other words, the lower 4 bits are put into the higher 4 bits and the higher 4 bits are put into the lower 4 bits

- ➢ SWAP works only on the accumulator(A)

before :	D7-D4	D3-D0
after :	D3-D0	D7-D4

BCD and ASCII Application Programs

ASCII code and BCD for digits 0 - 9

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

Packed BCD to ACSII Conversion

- ➢ The DS5000T microcontrollers have a real-time clock (RTC)
- ✓ The RTC provides the time of day (hour, minute, second) and the date (year, month, day) continuously, regardless of whether the power is on or off
- ➢ However this data is provided in packed BCD
- ✓ To be displayed on an LCD or printed by the printer, it must be in ACSII format

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

ACSII to Packed BCD Conversion

- To convert ASCII to packed BCD
  - ✓ It is first converted to unpacked BCD (to get rid of the 3)
  - ✓ Combined to make packed BCD

key	ASCII	Unpacked BCD	Packed BCD
4	34	0000 0100	
7	37	0000 0111	0100 0111 or 47H

```
MOV  A, #'4'          ;A=34H, hex for '4'
MOV  R1, #'7'         ;R1=37H,hex for '7'
ANL  A, #0FH          ;mask upper nibble (A=04)
ANL  R1,#0FH          ;mask upper nibble (R1=07)
SWAP  A               ;A=40H
ORL  A, R1            ;A=47H, packed BCD
```

Assume that register A has packed BCD, write a program to convert packed BCD to two ASCII numbers and place them in R2 and R6.

```
MOV  A, #29H          ;A=29H, packed BCD
MOV  R2, A            ;keep a copy of BCD data
ANL A, #0FH           ;mask the upper nibble (A=09)
ORL  A, #30H          ;make it an ASCII, A=39H('9')
MOV R6, A             ;save it
MOV A, R2             ;A=29H, get the original data
ANL A, #0F0H          ;mask the lower nibble
RR A                  ;rotate right
RR A                  ;rotate right
RR A                  ;rotate right
RR A                  ;rotate right
ORL A, #30H           ;A=32H, ASCII char. '2'
MOV R2, A             ;save ASCII char in R2
```

SWAP A

Using a Lookup Table for ASCII

Assume that the lower three bits of P1 are connected to three switches. Write a program to send the following ASCII characters to P2 based on the status of the switches.

000	‘0’
001	‘1’
010	‘2’
011	‘3’
100	‘4’
101	‘5’
110	‘6’
111	‘7’

Solution:

```
          MOV  DPTR, #MYTABLE
          MOV  A, P1                ;get SW status
          ANL  A,#07H               ;mask all but lower 3
          MOVC A,@A+DPTR            ;get data from table
          MOV  P2,A                 ;display value
          SJMP $                    ;stay here
;------------------
          ORG 400H
MYTABLE   DB ‘0’,‘1’,‘2’,‘3’,‘4’,‘5’,‘6’,‘7’
          END
```

Binary (Hex) to ASCII Conversion

- Many ADC (analog-to-digital converter) chips provide output data in binary (hex)
- ✓ To display the data on an LCD or PC screen, we need to convert it to ASCII
  - Convert 8-bit binary (hex) data to decimal digits, 000 – 255
  - Convert the decimal digits to ASCII digits, 30H – 39H

Checksum Byte in ROM

- ➢ To ensure the integrity of the ROM contents, every system must perform the checksum calculation
- ✓ The process of checksum will detect any corruption of the contents of ROM
- ✓ The checksum process uses what is called a checksum byte

  The checksum byte is an extra byte that is tagged to the end of series of bytes of data

Checksum Byte in ROM

- To calculate the checksum byte of a series of bytes of data
  - ✓ Add the bytes together and drop the carries
  - ✓ Take the 2's complement of the total sum, and it becomes the last byte of the series
- To perform the checksum operation, add all the bytes, including thechecksum byte
  - ✓ The result must be zero
  - ✓ If it is not zero, one or more bytes of data have been changed

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H.(a) Find the checksum byte, (b) perform the checksum operation to ensure data integrity, and (c) if the second byte 62H has been changed to 22H, show how checksum detects the error.

Solution:

(a) Find the checksum byte.

25H + 62H + 3FH + 52H 118H	The checksum is calculated by first adding the bytes. The sum is 118H, and dropping the carry, we get 18H. The checksum byte is the 2's complement of 18H, which is E8H

(b) Perform the checksum operation to ensure data integrity.

25H + 62H + 3FH + 52H + E8H 200H (dropping the carries)	Adding the series of bytes including the checksum byte must result in zero. This indicates that all the bytes are unchanged and no byte is corrupted.

(c) If the second byte 62H has been changed to 22H, show how checksum detects the error.

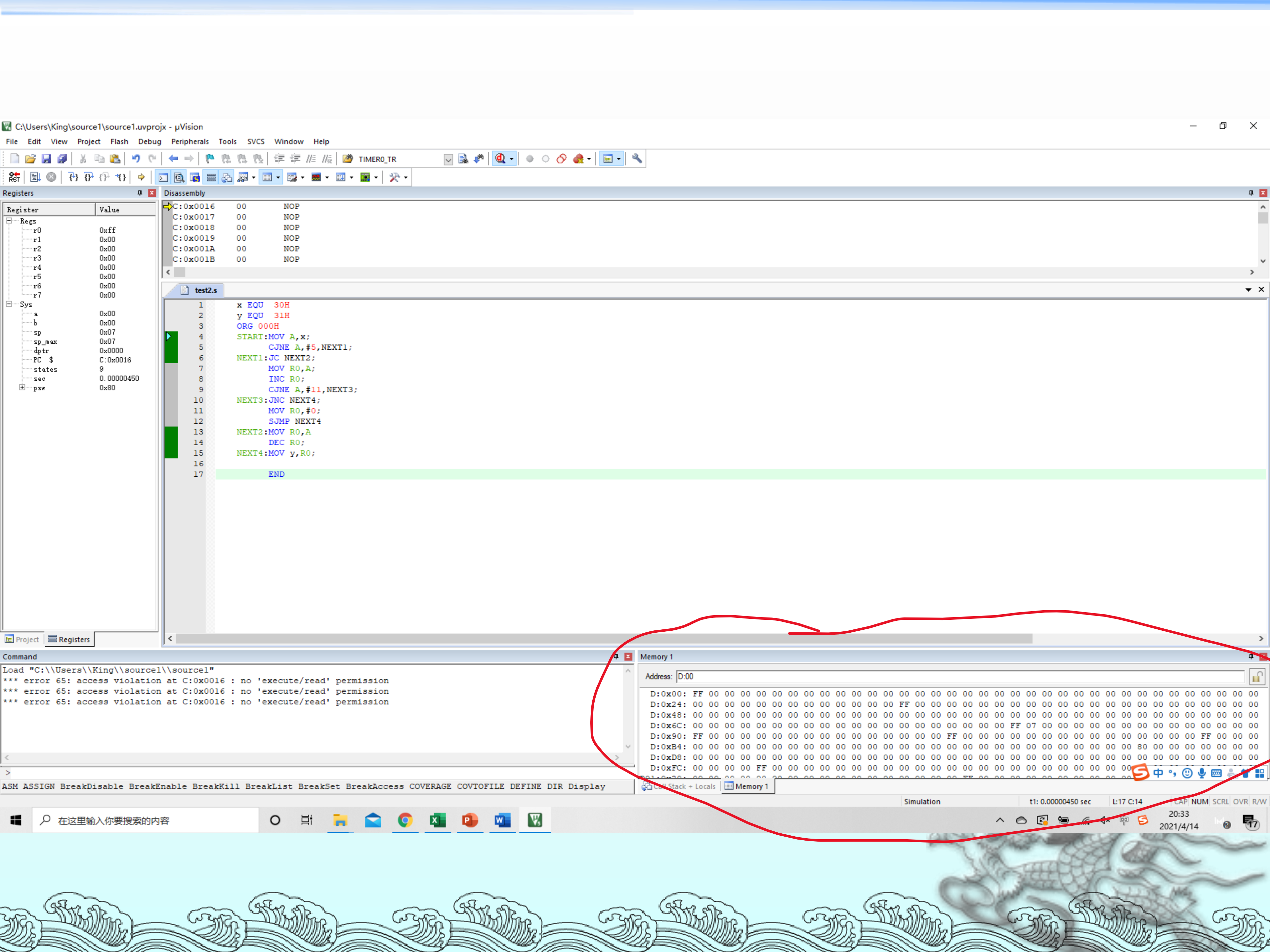
25H + 22H + 3FH + 52H + E8H 1C0H (dropping the carry, we get C0H)	Adding the series of bytes including the checksum byte shows that the result is not zero, which indicates that one or more bytes have been corrupted.

Homework（两题任选一题）

- 1. 在内存RAM 30H- 3EH 中存储着一个数组A[15]，计算其校验和，并将其保存在3FH中。编写汇编指令实现该功能（A[15]={27, 5,32,47,38,235,79,17,187, 58,23,35,211,104,9}; 需要编程给RAM赋值）。

- 2.在内存RAM 30H- 3EH 中存储着一个数组A[15]，找出该数组中的最小值，并将其保存在3FH中。编写汇编指令实现该功能（A[15]={27, 5,32,47,38,235,79, 17,187,58,23,35,211,104,9}; 需要编程给RAM赋值）。

- 需要在Keil中调试程序，并通过截图展示程序运行效果。


C:\Users\King\source1\source1.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
TIMER0_TR
Registers
Register Value
Regs
r0 0xff
r1 0x00
r2 0x00
r3 0x00
r4 0x00
r5 0x00
r6 0x00
r7 0x00
Sys
a 0x00
b 0x00
sp 0x07
sp_max 0x07
dptr 0x0000
PC $ C:0x0016
states 9
sec 0.00000450
psw 0x80
Disassembly
C:0x0016 00 NOP
C:0x0017 00 NOP
C:0x0018 00 NOP
C:0x0019 00 NOP
C:0x001A 00 NOP
C:0x001B 00 NOP
test2.s
x EQU 30H
y EQU 31H
ORG 000H
START:MOV A,x;
CJNE A,#5,NEXT1;
NEXT1:JC NEXT2;
MOV R0,A;
INC R0;
CJNE A,#11,NEXT3;
NEXT3:JNC NEXT4;
MOV R0,#0;
SJMP NEXT4
NEXT2:MOV R0,A
DEC R0;
NEXT4:MOV y,R0;
END
Project Registers
Command
Load "C:\\Users\\King\\source1\\source1"
*** error 65: access violation at C:0x0016 : no 'execute/read' permission
*** error 65: access violation at C:0x0016 : no 'execute/read' permission
*** error 65: access violation at C:0x0016 : no 'execute/read' permission
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display
Memory 1
Address: D:00
D:0x00: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x24: 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x6C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x90: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00 00 00
D:0xD8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0xFC: 00 00 00 00 FF 00 00 00 00 00 00 00
Call Stack + Locals Memory 1
Simulation t1: 0.00000450 sec L:17 C:14 CAP NUM SCRL OVR R/W
在这里输入你要搜索的内容
20:33
2021/4/14

THANK YOU!!