SH6610 instructions introduction

Sino Wealth Electronic (Shanghai) LTD.

Analysis of SH6610 Instructions

The following are SH6610 instructions, categorized and explained according to their respective functions. When you need an instruction for a certain function, you can look up the instruction in its function category. You'd better browse through all the instructions listed here, because even though you can't remember all of them at once, you can still have an impression to remind you of such instruction when needed. Some people don't even know about some very good instructions that he can use, just because he didn't browse through all of them. This ignorance may cause considerable waste of space on a system without a big enough ROM, which is so regrettable. I will explain as simply as possible to improve your learning efficiency. Of course, it is normal to understand or remember only part of the contents after reading once. And you are certain often revert to this book because it is a collection of instructions.

Instructions

Instruction is a series of codes that can be recognized by CPU, and then CPU will operate according to the given instruction.

Operand

Apart from instructions to tell CPU how to operate, the operation object must be designated. The object for CPU to operate with is called Operand. Therefore, a complete instruction must include two items, both instruction and operand.

Format of SH6610 Instructions

Instruction [Operand 1], [Operand 2]

Of the above, items inside [ ] are used according to nature of the instruction. Some instructions require only one operand, while some require two. Instruction and operand shall be separated by a space, and operands shall be separated from each other by a ','.

Execution Time of Instruction

Execution time for SH6610 instructions is one instruction cycle, which is one fourth of the system working frequency.

1-1 Instruction Categories

According to their functions, SH6610 instructions can be classified into four categories:

- Arithmetic Operation Instructions
  I. addition: ADC, ADCM, ADD, ADDM, ADI, ADIM
  II. Subtraction: SBC, SBCM, SUB, SUBM, SBI, SBIM
  III. BCD: DAA, DAS

- Logic Operation Instructions
  EOR, EORM, EORIM, OR, ORM, ORIM, AND, ANDM, ANDIM

- Data Transmission Instructions

  LDA, STA, LDI

- Flow Control Instructions

  BAZ, BC, BA0, BA1, BA2, BA3, CALL, RTNW, RTNI, HALT, STOP, JMP, TJMP

The above are all of the instructions of SH6610 series, which add up to only 40 in number, but never overlook them! A variety of consumer electrical products on the market are created with them, e.g. calculator, remote controller, watch, toy, etc.

1-2 Explanation of Symbols

Before going to our subject, we list the symbols that may appear afterwards so that our readers can understand this book more easily. The symbols are listed as follows:

PC	Program Counter
AC	Accumulator
CY	Carry Flag
Mx	Data Memory
bbb RAM	bank
ST	Stack
TBR	Table Branch Register
X	Program Address
I	Immediate Data

&	Logic AND
|	Logic OR
^	Logic EOR

Now let’s enter the world of instructions of SH6610 series. “Let ‘s go “。

1-3 Instructions for Data Transmission

During the internal operation of the system, data is transmitted rapidly and incessantly between memories or registers. This fast and incessant transmission is the power of system capability. SH6610 system provides several instructions for data transmission, as follows:

Instruction: LDI	Function: to load Immediate Data I to “Accumulator” and “Data Memory”
Format:	LDI Mx, I
Instruction Code:	01111 iiii xxx xxxx
Carry Flag:	Not affected
Operation:	AC , Mx ← I

Explanation

LDI is a very frequently used instruction. It loads immediate data I to accumulator and data memory. However, due to this 4-bit system, the preset range of the immediate data I is 00H ~ 0FH(0 ~ 15), and that of Mx is 00H~7FH.

[Example]

```
LDI     20H, 05H
after execution:
        AC=05H
        Content of data memory $20H=05H
```

Programming Tip

There isn’t any specially defined register for users in SH6610 system, but we can use its powerful data memory as registers in our program designing. In the above example, I have used data memory $20H for a register. However, if they are expressed only by address and without respective names, the design of the program will be very confusing. Here’s a tip for you: you can use the pseudo-instruction EQU to define each data address.

AAA EQU 20H

Name of memory variable

Pseudo-

Data memory address

Therefore, the LDI instruction can also be written like this:

```
LDI     AAA, 05H
```

Instruction: STA	Function:to store the value of Accumulator to Data Memory
Format:	STA Mx, bbb
Instruction Code:	00111 1bbb xxx xxxx
Carry Flag:	Not affected
Operation:	Mx ← AC

Explanation

STA loads the value of accumulator to data memory. When executing this instruction, CPU does the transmission only and the carry flag is not affected.

[Example] Save the value of AC in $21H

```
            :
LDI     20H, 05H      ;AC=05H, $20H=05H
STA     21H, 00H      ;$21H=05H
            :
```

Programming Tip

If operand 2 is the immediate data I, then the program can be written in the following ways:

```
LDI    20H,0AH       ;expressed in hex
LDI    20H,10        ;expressed in decimal
LDI    20H,1010B     ;expressed in binary
```

Instruction: LDA	Function: to load the value of Data Memory to Accumulator
Format:	LDA Mx, bbb
Instruction Code:	00111 0bbb xxx xxxx
Carry Flag:	Not affected
Operation:	AC ← Mx

Explanation

LDA loads the value of data memory to accumulator. This instruction does not affect the carry flag.

[Example] Load the value of $20H to AC

```
        :
LDI     20H,05H     ;$20H=05H,AC=05H
LDI     21H,0FH     ;$21H=0FH,AC=0FH
LDA     20H,0       ;AC=05H
        :
```

Programming Tip

When writing a program, we often use labels as jumping destinations in the program. Label names can be defined by user according to the following rules:

I. A label mustn't begin with number or space.

II. Length of a label mustn't exceed 7 characters. Only the first 7 characters will be recognized for labels exceeding that length.

1-4 Instructions for Arithmetic Operation

SH6610 series provide some frequently used arithmetic (integer) operation instructions, like: Addition, subtraction, BCD adjustment, etc.

Instruction: ADD	Function: to add up the values of Data Memory and Accumulator, and then save the result in the Accumulator
Format:	ADD Mx, bbb
Instruction Code:	00001 0bbb xxx xxxx
Carry Flag:	CY
Operation:	AC ← Mx + AC

Explanation

Instruction ADD adds up the values of data memory and accumulator and saves the result in the accumulator. The ADD operation affects carry flag: when the result of ADD exceeds 0FH, the carry flag is set to 1; otherwise the value of CY is 0. Therefore we can decide whether there's a carry by the value of the carry flag after addition.

[Example] 05H +06H

```
            :
LDI     20H,05H     ;$20H=05H,AC=05H
LDI     21H,06H     ;$21H=06H,AC=06H
LDA     20H,00H     ;AC=05H
ADD     21H,0       ;AC=0BH,CY=0,$21H=06H
                    :
```

[Example] 0BH + 06H

```
            :
LDI     20H,0BH     ;$20H=0BH,AC=0BH
LDI     21H,06H     ;$21H=06H,AC=06H
LDA     20H,00H     ;AC=0BH,CY=0
ADD     21H,0       ;AC=01H,CY=1,$21H=06H
           :
```

Instruction: ADDM	Function: to add up the values of Data Memory and Accumulator, and save the result in both the Accumulator and the Data Memory
Format:	ADDM Mx, bbb
Instruction Code:	00001 1bbb xxx xxxx
Carry Flag:	CY
Operation:	AC , Mx ← Mx + AC

Explanation

Instruction ADDM adds up the values of data memory and accumulator and saves the result in both the accumulator and the data memory. The ADDM operation affects carry flag: when the result of ADDM exceeds 0FH, the carry flag is set to 1; otherwise the value CY is 0. Therefore we can decide whether there's a carry by the value of the carry flag after addition.

[Example] 05H +06H

```
            :
LDI     20H,05H     ;$20H=05H,AC=05H
LDI     21H,06H     ;$21H=06H,AC=06H
LDA     20H,00H     ;AC=05H
ADDM    21H,0       ;AC=0BH,$21H=0BH,CY=0
                      :
```

[Example] 0BH + 06H

```
            :
LDI     20H,0BH     ;$20H=0BH,AC=0BH
LDI     21H,06H     ;$21H=06H,AC=06H
LDA     20H,00H     ;AC=0BH,CY=0
ADDM    21H,0       ;AC=01H,$21H=01H,CY=1
            :
```

Programming Tip

When you are reading the examples, I suggest that you call them into ICE after compiling to watch the change in each of the registers step by step.

Instruction: ADC	Function: to add up the value of Data Memory, Carry Flay and the value of Accumulator, and then save the result in the Accumulator
Format: Instruction Code: Carry Flag: Operation:	ADC Mx, bbb 00000 0bbb xxx xxxx CY AC ← Mx + AC + CY

Explanation

Instruction ADC adds up the value of data memory, carry flag and the value of accumulator, and saves the result in the accumulator. The ADC operation affects the carry flag: when the result of ADC exceeds 0FH, the carry flag is set to 1; otherwise the value of CY is 0. Therefore we can decide whether there's a carry by the value of the carry flag after addition.

[Example] 05H +06H , CY=1

```
            :               ;CY=1
LDI         20H,05H         ;$20H=05H,AC=05H
LDI         21H,06H         ;$21H=06H,AC=06H
LDA         20H,00H         ;AC=05H
ADC         21H,0           ;AC=0CH,$21H=06H,CY=0
            :
```

[Example] 0BH + 06H , CY=0

```
            :               ;CY=0
LDI         20H,0BH         ;$20H=0BH,AC=0BH
LDI         21H,06H         ;$21H=06H,AC=06H
LDA         20H,00H         ;AC=0BH
ADC         21H,0           ;AC=01H,$21H=06H,CY=1
            :
```

Programming Tip

When carry is not considered, you'd better use ADD rather than ADC, in order to avoid extra uncertainty due to the addition of CY value (because CY can be either 1 or 0).

Instruction: ADCM	Function: to add up the value of Data Memory, Carry Flag and the value of Accumulator, and then save the result in both the Accumulator and the Data Memory
Format: Instruction Code: Carry Flag: Operation:	ADCM Mx, bbb 00000 1bbb xxx xxxx CY AC , Mx ← Mx + AC + CY

Explanation

Instruction ADCM adds up the value of data memory, carry flag and the value of accumulator, and saves the result in both the accumulator and the data memory. The ADCM operation affects the carry flag: when the result of ADCM exceeds 0FH, the carry flag is set to1; otherwise the value of CY is 0. Therefore we can decide whether there's a carry by the value of the carry flag after addition.

[Example] 05H +06H , CY=1

```
                  :        ;CY=1
        LDI       20H,05H  ;$20H=05H,AC=05H
        LDI       21H,06H  ;$21H=06H,AC=06H
        LDA       20H,00H  ;AC=05H
        ADCM      21H,0    ;AC=0CH,$21H=0CH,CY=0
                  :
```

[Example] 0BH + 06H , CY=0

```
                  :        ;CY=0
        LDI       20H,0BH  ;$20H=0BH,AC=0BH
        LDI       21H,06H  ;$21H=06H,AC=06H
        LDA       20H,00H  ;AC=0BH,
        ADCM      21H,0    ;AC=01H,$21H=01H,CY=1
                  :
```

Instruction: ADI	Function: to add up the value of Data Memory and Immediate Data I, and then save the result in Accumulator
Format: Instruction Code: Carry Flag: Operation:	ADI Mx, I 01000 iiii xxx xxxx CY AC ← Mx + I

Explanation

Instruction ADI adds up the value of data memory and immediate data I, and saves the result in accumulator. The ADI operation affects carry flag: when the result of ADI exceeds 0FH, the carry flag is set to1; otherwise the value of CY is 0. Therefore we can decide whether there's a carry by the value of the carry flag after addition.

[Example] $20H=05H , I=04H

```
        :
LDI     20H,05H     ;$20H=05H,AC=05H,CY=0
ADI     20H,04H     ;AC=09H,$20H=05H,CY=0
        :
```

[Example] $20H=0AH , I=07H

```
        :
LDI     20H,0AH     ;$20H=0AH,AC=0AH,CY=0
ADI     20H,07H     ;AC=01H,$20H=0AH,CY=1
        :
```

Instruction: ADIM	Function: to add up the value of Data Memory and Immediate Data I, and then save the result in both Accumulator and the Data Memory
Format:	ADIM Mx, I
Instruction Code:	01001 iiii xxx xxxx
Carry Flag:	CY
Operation:	AC , Mx ← Mx + I

Explanation

Instruction ADIM adds up the value of data memory and immediate data I, and saves the result in both accumulator and the data memory. The ADIM operation affects carry flag: when the result of ADIM exceeds 0FH, the carry flag is set to1; otherwise the value of CY is 0. Therefore we can decide by the value of the carry flag after addition whether there is a carry.

[Example] $20H=05H , I=04H

```
:
LDI       20H,05H    ;$20H=05H,AC=05H
ADIM      20H,04H    ;AC=09H,$20H=09H,CY=0
:
```

[Example] $20H=0AH , I=07H

```
:
LDI       20H,0AH    ;$20H=0AH,AC=0AH
```

```
ADIM    20H,07H    ;AC=01H,$20H=01H,CY=1
:
```

Instruction: DAA	Function: to adjust the value of Data Memory to decimal after addition, and then save the result in both Accumulator and the Data Memory
Format:	DAA Mx
Instruction Code:	11001 0110 xxx xxxx
Carry Flag:	CY
Operation:	AC ; Mx ← Decimal , adjust AC for Add

Explanation

Instruction DAA acts by adjusting the value of data memory to decimal after addition and saving the result to both accumulator and the data memory. Its adjusting method is if the value of the data memory is greater than 9 or if CY= 1, then add 6 to the data memory and set the carry flag to 1.

[Example] 06H + 05H , and do DAA adjustment

```
        :
LDI     20H,06H     ;AC=06H,$20H=06H
LDI     21H,05H     ;AC=05H,$21H=05H
LDA     20H,0       ;AC=06H
ADD     21H,0       ;AC=0BH,CY=0
DAA     21H         ;AC=01H,$21H=01H,CY=1
        :
```

Instruction: SUB	Function: to subtract the value of Accumulator from the value of Data Memory, and then save the result in the Accumulator
Format: Instruction Code: Carry Flag: Operation:	SUB Mx, bbb 00011 0bbb xxx xxxx CY AC ← Mx - AC

Explanation

SUB subtracts the value of accumulator from the value of data memory and saves the result in the accumulator. When executing SUB, if the value of data memory is less than the value of accumulator, a "borrow" will take place and CY will be set to 0. On the contrary, if the value of data memory is greater than the value of accumulator, borrow will not happen and CY will be set to 1. Therefore we can decide by the value of CY whether there is a borrow after execution of SUB. Besides, subtraction in the system is done through addition, i.e. when subtracting a number, it is actually adding the number's binary complement.

[Example] 06H - 05H

```
            :
LDI     20H,05H        ;AC=05H,$20H=05H
LDI     21H,06H        ;AC=06H,$21H=06H
LDA     20H,0          ;AC=05H
SUB     21H,0          ;AC=1,CY=1,$21H=06H
            :
```

[Example] 05H - 06H

```
            :
LDI     20H,05H        ;AC=05H,$20H=05H
LDI     21H,06H        ;AC=06H,$21H=06H
LDA     21H,0          ;AC=06H
SUB     20H,0          ;AC=0FH,CY=0,$20H=05H
            :
```

Instruction: SUBM	Function: to subtract the value of Accumulator from the value of Data Memory, and then save the result in both the Accumulator and the Data Memory
Format:	SUBM Mx, bbb
Instruction Code:	00011 1bbb xxx xxxx
Carry Flag:	CY
Operation:	AC , Mx ← Mx - AC

Explanation

System movement of SUBM is almost the same as SUB, it subtracts current value of accumulator from the value of data memory and saves the result in the accumulator as well as in the data memory. When executing SUBM, if the value of data memory is less than the value of accumulator, a "borrow" will take place and CY will be set to 0. On the contrary, if the value of data memory is greater than the value of accumulator, borrow will not happen and CY will be 1. Therefore we can decide by the value of CY whether there is a borrow after execution of SUBM.

[Example] 06H - 05H

```
            :
LDI     20H,05H     ;AC=05H,$20H=05H
LDI     21H,06H     ;AC=06H,$21H=06H
LDA     20H,0       ;AC=05H
SUBM    21H,0       ;AC=01H,CY=1,$21H=01H
            :
```

[Example] 05H - 06H

```
            :
LDI     20H,05H     ;AC=05H,$20H=05H
LDI     21H,06H     ;AC=06H,$21H=06H
LDA     21H,0       ;AC=06H
SUBM    20H,0       ;AC=0FH,CY=0,$20H=0FH
            :
```

Instruction: SBC	Function: to subtract the value of Accumulator from the value of Data Memory, add Carry Flag, and then save the result in the Accumulator
Format:	SBC Mx, bbb
Instruction Code:	00010 0bbb xxx xxxx
Carry Flag:	CY
Operation:	AC ← Mx - AC + CY

Explanation

System movement of SBC is to subtract the value of accumulator from the value of data memory, add the value of carry flag, and then save the result in the accumulator. When executing SBC, if the value of data memory is less than the value of accumulator, a “borrow” will take place and the CY will be set to 0. On the contrary, if the value of data memory is greater than the value of accumulator, borrow will not happen and the CY will be 1. Therefore we can decide by the value of CY whether there is a borrow after execution of SBC.

[Example] CY=0, 6 - 5=?

```
        :               ;CY=0
LDI     20H,05H         ;AC=05H,$20H=05H
LDI     21H,06H         ;AC=06H,$21H=06H
LDA     20H,0           ;AC=05H
SBC     21H,0           ;AC=01H,CY=0,$21H=06H
        :
```

[Example] CY=1, 6 - 5=?

```
        :               ;CY=1
LDI     20H,05H         ;AC=05H,$20H=05H
LDI     21H,06H         ;AC=06H,$21H=06H
LDA     20H,0           ;AC=05H
SBC     21H,0           ;AC=02H,CY=0,$21H=06H
        :
```

Instruction: SBCM	Function: to subtract the value of Accumulator from the value of Data Memory, add Carry Flag, and then save the result in both the Accumulator and the Data Memory
Format:	SBCM Mx , bbb
Instruction Code:	00010 1bbb xxx xxxx
Carry Flag:	CY
Operation:	AC , Mx ← Mx - AC + CY

Explanation

System movement of SBCM is to subtract the value of accumulator from the value of data memory, add the value of carry flag, and then save the result in both the accumulator and data memory. When executing SBCM, if the value of data memory is less than the value of accumulator, a "borrow" will take place and the CY will be set to 0. On the contrary, if the value of data memory is greater than the value of accumulator, borrow will not happen and the CY will be set to 1. Therefore we can decide by the value of CY whether there is a borrow after execution of SBCM.

[Example] CY=0, 6 - 5=?

```
            :           ;CY=0
LDI     20H,05H         ;AC=05H,$20H=05H
LDI     21H,06H         ;AC=06H,$21H=06H
LDA     20H,0           ;AC=05H,CY=0
SBCM    21H,0           ;AC=01H,CY=0,$21H=01H
            :
```

[Example] CY=1, 6 - 5=?

```
            :           ;CY=1
LDI     20H,05H         ;AC=05H,$20H=05H
LDI     21H,06H         ;AC=06H,$21H=06H
LDA                     20H,0       ;AC=05H,CY=1
SBCM                    21H,0       ;AC=02H,CY=0,$21H=02H
            :
```

Instruction: SBI	Function: to subtract the Immediate Data I from the value of Data Memory, and then save the result in Accumulator
Format:	SBI Mx, I
Instruction Code:	01010 iiii xxx xxxx
Carry Flag:	CY
Operation:	AC ←Mx - I

Explanation

System movement of SBI is to subtract immediate data I from the value of data memory, and save the result in accumulator. When executing SBI, if the value of the data memory is less than the immediate data, a "borrow" will take place and CY will be set to 0. On the contrary, if the value of data memory is greater than the immediate data, borrow will not happen and CY will be 1. Therefore we can decide by the value of CY whether there is a borrow after execution of SBI.

```
[ Example ]  $20H=05H , I=04H
                     :
          LDI      20H,05H        ;$20H=05H,AC=05H,CY=0
          SBI      20H,04H        ;AC=01H,$20H=05H,CY=1
                     :
```

```
[ Example ]  $20H=02H , I=07H
                     :
          LDI      20H,02H        ;$20H=02H,AC=02H,CY=0
          SBI      20H,07H        ;AC=0BH,$20H=02H,CY=0
                     :
```

Instruction: SBIM	Function: to subtract Immediate Data I from the value of Data Memory, and then save the result in both Accumulator and the Data Memory
Format:	SBIM Mx, I
Instruction Code:	01011 iiii xxx xxxx
Carry Flag:	CY
Operation:	AC , Mx ← Mx - I

Explanation

System movement of SBIM is to subtract immediate data I from the value of data memory, and save the result in both accumulator and the data memory. When executing SBIM, if the value of data memory is less than the immediate data, a "borrow" will take place and CY will be set to 0. On the contrary, if the value of data memory is greater than the immediate data, borrow will not happen and CY will be 1. Therefore we can decide by the value of CY whether there is a borrow after execution of SBIM.

```
[ Example ]  $20H=05H , I=04H
                     :
          LDI      20H,05H        ;$20H=05H,AC=05H
          SBIM     20H,04H        ;AC=01H,$20H=01H,CY=1
                     :
```

[Example] $20H=0 2H , I=07H

```
                :
        LDI     20H,02H     ;$20H=02H,AC=02H
        SBIM    20H,07H     ;AC=0BH,$20H=0BH,CY=0
                :
```

Instruction: DAS	Function: to adjust the value of Data Memory to decimal after subtraction, and save the result in Accumulator and the Data Memory
Format:	DAS Mx
Instruction Code:	11001 1010 xxx xxxx
Carry Flag:	CY
Operation:	AC ; Mx ← Decimal , adjust AC for Sub

Explanation

Instruction DAS acts by adjusting the value of data memory to decimal after subtraction and saving the result to both accumulator and the data memory. Its adjusting method is if the value of data memory is greater than 9 or if CY=0, then add 0AH to the data memory and set CY to 0.

[Example] 05H - 06H , and do DAS adjustment

```
                :
        LDI     20H,06H     ;AC=06H,$20H=06H
        LDI     21H,05H     ;AC=05H,$21H=05H
        LDA     20H,0       ; AC=06H
        SUB     21H,0       ;AC=0FH,CY=0
D         AS    21H         ;AC=09H,$21H=09H,CY=0
                :
```

1-5 Instructions for Logic Operation

Logic instructions are essential to system structu re. SH6610 se ries MCU provid e some common logic instructions. Now I'm going to explain to you one by one in most details, and assist my explanation with simpl e examples, so that you can quickly unde rstand action theory of each instruction.

Instruction: AND	Function: to do logic AND operation with the values of Data Memory and Accumulator, and then save the result in the Accumulator
Format:	AND Mx, bbb
Instruction Code:	00110 1bbb xxx xxxx
Carry Flag:	Not affected
Operation:	AC ← Mx & AC

Explanation

In AND operation, the resul t will be 1(true) only if both of the two operands are 1(true). Its logic table is as follows:

Logic operation table for AND

a	**b**	**a AND b**
0	0	0
0	1	0
1	0	0
1	1	1

However, in real inst ruction the logic operand has 4 bit s rather than 1 bit. The instruction A ND i s to AND the valu es of dat a me mory with accumulator, a nd the re sult i s saved in the accumulator.

[Example] 06H & 05H

```
        :
        LDI     20H,0110B    ;AC=06H,$20H=06H
        LDI     21H,0101B    ;AC=05H,$21H=05H
        AND     20H,0        ;AC=0100B,$20H=06H
        :
```

Instruction: ANDM	Function: to do logic AND operation with the values of Data Memory and Accumulator, and then save the result in both the Accumulator and the Data Memory
Format:	ANDM Mx, bbb
Instruction Code:	00110 1bbb xxx xxxx
Carry Flag:	Not affected
Operation:	AC , Mx ← Mx & AC

Explanation

System movement of the instruction ANDM is almost the same as AND, but saving the operation result in data memory as well as in accumulator.

[Example] 01 10B & 0101B

```
              :
LDI    20H,0110B    ;AC=0110B,$20H=0110B
LDI    21H,0101B    ;AC=0101B,$21H=0101B
ANDM   20H,0        ;AC=0100B,$20H=0100B
              :
```

Instruction: ANDIM	Function: to do logic AND operation with the value of Data Memory and Immediate Data I, and then save the result in both Accumulator and the Data Memory
Format:	ANDIM Mx, I
Instruction Code:	01110 iiii xxx xxxx
Carry Flag:	Not affected
Operation:	AC , Mx ← Mx & I

Explanation

System movement of the instruction ANDIM is to change operand 2 (accumulator) of instruction AND to immediate data I. This instruction is in immediate mode, so the address of data memory can only be set to bank 0($000 H ~ $07FH). The operation result is saved in both accumulator and the data memory.

[Example] $20H=01 10B , I=0011B

```
              :
LDI    20H,0110B    ;AC=06H,$20H=0110B
ANDIM  20H,0011B    ;AC=0010B,$20H=0010B
              :
```

Programming Tip

ANDIM itself has a special function "MASK". When we need to set a certain bit to 0, we can clear this bit to 0 with ANDIM like this:

[Example] Clear bit 2 of $20H to 0

ANDIM 20H, 1011B

After execution: $20H=x0xxB

Instruction: OR	Function: to do logic OR operation with the values of Data Memory and Accumulator, and then save the result in the Accumulator
Format: Instruction Code: Carry Flag: Operation:	OR Mx, bbb 00101 0bbb xxx xxxx Not affected AC ← Mx | AC

Explanation

In OR operation, the result will be 1(true) if either one of the two operands is 1(true). Its logic table is as follows:

Logic operation table for OR

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

However, in real instruction the logic operand also has 4bits rather than 1 bit. The instruction OR is to OR the values of data memory with accumulator, and the result is saved in the accumulator.

```
[ Example ]  0001B  | 0100B
                  :
      LDI      20H,0001B     ;$20H=0001B,AC=0001B
      LDI      21H,0100B     ;$21H=0100B,AC=0100B
      OR       20H,0         ;$20H=0001B,AC=0101B
                  :
```

Instruction: ORM	Function: to do logic OR operation with the values of Data Memory and Accumulator, and then save the result in both the Accumulator and the Data Memory
Format: Instruction Code: Carry Flag: Operation:	ORM Mx, bbb 00101 1bbb xxx xxxx Not affected AC , Mx← Mx | AC

Explanation

System movement of the instruction ORM is almost the same as OR, but saving the operation result in data memory as well as in accumulator.

```
[ Example ]  0001B   | 0100B
                     :
            LDI      20H,0001B     ;$20H=0001B,AC=0001B
            LDI      21H,0100B     ;$21H=0100B,AC=0100B
            ORM      20H,0         ;$20H=0101B,AC=0101B
                     :
```

Programming Tip

In program designing, if a certain bit of a variable needs to be set to 1 and the other bits must not be affected, then this can be done by the instruction OR. Because any bit that has done OR operation with 0 can keep its original value, while those with 1 will have the value of 1.

Instruction: ORIM	Function: to do logic OR operation with the value of Data Memory and Immediate Data I, and then save the result in both Accumulator and the Data Memory
Format:	ORIM Mx, I
Instruction Code:	01101 iiii xxx xxxx
Carry Flag:	Not affected
Operation:	AC , Mx ← Mx | I

Explanation

System movement of the instruction ORIM is to do logic OR operation with the value of data memory and Immediate Data I, and save the result in both accumulator and the data memory. This instruction is also in immediate mode.

[Example] Set bit 3 of the value of $20H to 1

```
ORIM     20H,  1000B
After execution: $20H=1xxxB
```

Instruction: EOR	Function: to do logic Exclusive OR operation with the values of Data Memory and Accumulator, and then save the result in the Accumulator
Format: Instruction Code: Carry Flag: Operation:	EOR Mx, bbb 00100 0bbb xxx xxxx Not affected AC ← Mx ^ AC

Explanation

System movement of EOR is to do logic Exclu sive OR operation with the value s of dat a memory and accumulator, and save the result in the accumulator. EOR is usually referred to as Exclusive OR, becau se the operati on resu lt will be 1 only if the two opera nds have different values; otherwise the result will be 0. The logic table for EOR is as follows:

Logic operation table for EOR

a	**b**	**a EOR b**
0	0	0
0	1	1
1	0	1
1	1	0

[Example] 0011B ^ 0101B

```
        :
LDI     20H,0011B     ;$20H=0011B,AC=0011B
LDI     21H,0101B     ;$21H=0101B,AC=0101B
EOR     20H,0         ;$20H=0011B,AC=0110B
        :
```

Instruction: EORM	Function: to do logic Exclusive OR operation with the values of Data Memory and Accumulator, and then save the result in both the Accumulator and the Data Memory
Format:	EORM Mx, bbb
Instruction Code:	00100 1bbb xxx xxxx
Carry Flag:	Not affected
Operation:	AC , Mx ← Mx ^ AC

Explanation

System movement of the instructio n EORM is almost the sa me as EOR, i.e. doing EOR action with the values of data memory and accumulator, but saving the operation result in the data memory as well as in the accumulator.

[Example] 0011B ^ 0101B

```
            :
LDI     20H,0011B       ;$20H=0011B,AC=0011B
LDI     21H,0101B       ;$21H=0101B,AC=0101B
EORM    20H,0           ;$20H=0110B,AC=0110B
            :
```

Instruction: EORIM	Function: to do logic EOR operation with the value of Data Memory and Immediate Data I, and then save the result in both the Accumulator and the Data Memory
Format:	EORIM Mx, I
Instruction Code:	01100 iiii xxx xxxx
Carry Flag:	Not affected
Operation:	AC , Mx ← Mx ^ I

Explanation

Operand 2 of the instructi on EORIM should be immediate dat a. This instruction is to do logic EOR op eration with the value of d ata memory and immedi ate data I, and to save th e result in both the accumulator and the data memory.

[Example] $20H=0 011B, I=0101B

```
            :
LDI     20H,0011B       ;$20H=0011B,AC=0011B
EORIM   20H,0101B       ;$20H=0110B,AC=0110B
            :
```

Programming Tip

After reading all of the logic instructions, you may wonder why there haven't the 'NOT' instruction (inverse)? If there haven't such an instruction in SH6610 series, what can I do? Don't worry, Programming Tip is going to tell you how to use other instructions to perform the NOT function. The operation of NOT is to change 0 → 1 or 1 → 0 on each bit. Here, the EOR instruction can help us to get the inversed value by doing EOR operation with immediate data (0FH) and the variable that wants to be done NOT. You'll understand clearly after reading the following example.

[Example] Do the operation NOT with the value of $20H (1100B)

```
EORIM    20H,0FH
```

Execution result: $20H=0011B,AC=0011B

1-6 Instructions for Flow Control

Instruction: JMP	Function: to jump to a designated address to execute program
Format:	JMP X
Instruction Code:	1110p xxxx xxx xxxx
Carry Flag:	Not affected
Operation:	PC ← X(Include p)

Explanation

Instruction JMP jumps to a designated address to execute program. However, addressing capability of SH6610 series CPU is limited to 4K words (0000H~0FFFH), so the jumping range of JMP can only reach 4K(0FFFH). Address beyond 4K shall be reached by switching banks. There's detailed explanation in later chapters for how to switch banks. The JMP instruction is similar to the GOTO instruction in BASIC program.

[Example] PC=40H , Jump to 0E00H

```
JMP    0E00H
```

Execution result: PC=0E00H

[Example] Jump to LABLE

```
        JMP     LOOP        ;jump to loop (PC ← 0340h)
                :
        ORG  0340H
        LOOP :  NOP
                            NOP
                :
```

Instruction: BAZ	Function: if AC equals 0, then go to a designated address to execute program; otherwise continue to execute the next line
Format:	BAZ X
Instruction Code:	10010 xxxx xxx xxxx
Carry Flag:	Not affected
Operation:	PC ←X , if AC=0

Explanation

If the value of AC is 0, then after executing the BAZ instruction, PC will go to the designated address X to execute program, the range of X being from $000H to $7FF H. It continues to execute the next line if the value of AC is 1

[Example] if ($20H=$20H-1)=00H then goto INC21H

```
            LDI     20H,0FH     ;$20H=0FH
DEC20 H:    SBIM    20H,01H     ;AC,$20H ← $20H -1
            BAZ     INC21H      ;ifAC=0 jump to INC21H
            JMP     DEC20H      ;else jump to DEC20H
                    :
 INC21H     ADIM    21H,01H     ;$21H+1
                    :
```

Instruction: BA0	Function: if bit 0 of AC is 1, then go to a designated address to execute program; otherwise continue to execute the next line
Format:	BA0 X
Instruction Code:	10100 xxxx xxx xxxx
Carry Flag:	Not affected
Operation:	PC ← X , if AC(0)=1

Explanation

If bit 0 of AC is 1, then after executing BA0 instruction, PC will go to the designated address X to execute program, the range of X being from $000H to $7FFH or from $0800H to $0FFFH. It continues to execute the next line if bit 0 of AC is 0.

[Example] if $20H(bit 0)=1 then goto INC21H

```
                :
          LDI   20H,0FH    ;$20H=0FH,AC=0FH
DEC20H:   BIM   20H,01H    ;$20H,AC←$20H -1
          BA0   INC21H     ;if AC(bit0)=1,jump to INC21H
          JMP   DEC20H     ;else jump to DEC20H
                :
INC21H:   ADIM  21H,01H    ;$21H,AC←$21H+1
```

Instruction: BA1	Function: if bit 1 of AC is 1, then go to designated address to execute program; otherwise continue to execute the next line
Format:	BA1 X
Instruction Code:	10101 xxxx xxx xxxx
Carry Flag:	Not affected
Operation:	PC ← X , if AC(1)=1

Explanation

If bit 1 of AC is 1, then after executing BA1 instruction, PC will go to the designated address X to execute program, the range of X being from $000H to $7FFH or from $0800H to $0FFFH. It continues to execute the next line if bit 1 of AC is 0.

[Example] if $20H(bit 1)=1 then goto INC21H

```
                :
          LDI   20H,0FH    ;$20H=0FH,AC=0FH
DEC20 H:        SBIM       20H,01H      ;$20H,AC←$20H -1
          BA1   INC21H     ;ifAC(bit1)=1,jump to INC21H
                JMP        DEC20H       ;else jump to DEC20H
                :
INC21 H:  ADIM  21H,01H    ;$21H,AC←$21H+1
                :
```

Instruction: BA2	Function: if bit 2 of AC is 1, then go to designated address to execute program; otherwise continue to execute the next line
Format:	BA2 X
Instruction Code:	10110 xxxx xxx xxxx
Carry Flag:	Not affected
Operation:	PC ←X , if AC(2)=1

Explanation

If bit 2 of AC is 1, then after executing BA2 instruction, PC will go to the designated address X to execute program, the range of X being from $000H to $7FFH or from $0800H or $0FFFH. It continues to execute the next line if bit 1 of AC is 0.

[Example] if $20H(bit 2)=1 then goto INC21H

```
                :
        LDI     20H,0FH     ;$20H=0FH,AC=0FH
DEC20H: SBIM    20H,01H     ;$20H,AC←$20H -1
        BA2     INC21H      ;if AC(bit2)=1,jump to INC21H
        JMP     DEC20H      ;else jump to DEC20H
                :
INC21H: ADIM    21H,01H     ;$21H,AC←$21H+1
                :
```

Instruction: BA3	Function: if bit 3 of AC is 1, then go to designated address to execute program; otherwise continue to execute the next line
Format:	BA3 X
Instruction Code:	10111 xxxx xxx xxxx
Carry Flag:	Not affected
Operation:	PC ←X , if AC(3)=1

Explanation

If bit 3 of AC is 1, then after executing BA3 instruction, PC will go to the designated address X to execute program, the range of X being from $000H to $7FFH or from $0800H to $0FFFH. It continues to execute the next line if bit 1 of AC is 0.

[Example] if $20H(bit 3)=1 then goto INC21H

```
                    :
            LDI     20H,0FH     ;$20H=0FH,AC=0FH
DEC20H      SBIM    20H,01H     ;$20H,AC←$20H -1
            BA3     INC21H      ;if AC(bit3)=1,jump to INC21H
            JMP     DEC20H      ;else jump to DEC20H
                    :
INC21H:     ADIM    21H,01H     ;$21H,AC←$21H+1
                                :
```

Instruction: BC	Function: if CY is 1, then go to designated address to execute program; otherwise continue to execute the next line
Format:	BC X
Instruction Code:	10011 xxxx xxx xxxx
Carry Flag:	CY
Operation:	PC ← X , if CY=1

Explanation

If CY is 1, then after executing instruction BC, PC will go to the designated address X to execute program, the range of X being from $000 H to $7FFH. It continues to execute the next line if CY is not 1. The instruction BC is often used after addition or subtraction to decide whether there is a carry or borrow. You should especially note that for addition, CY is set to 1 when there is a carry, while for subtraction CY is set to1 when there isn't any borrow. Therefore you should be careful when dealing with program flows.

[Example] if CY=1 then goto INC20H

```
                    :
            LDI     20H,0FH     ;$20H=0FH,AC=0FH,CY=0
INC20H :    SBIM    20H,01H     ;$20H,AC←$20H -1
            BC      INC20H      ;if CY=1,jump to INC20H
                    :
```

Instruction: TJMP	Function:Unconditionally go to the address composed by (PC11~PC8), TBR and AC value to execute program
Format:	TJMP
Instruction Code:	11110 1111 111 1111
Carry Flag:	Not affected
Operation:	

Explanation

The destination address for Instruction TJMP is composed by PC's bit 8~bit 11, TBR and AC value (please refer to instruction RTNW for program example).

Example: PC=300H, TBR=01H, AC=02H, then the rule for composing a destination address is as follows:

address=PC(bit8~bit 11) TBR AC

If:

PC =300H

TBR=01H

AC=02H

Then the destination address is: 3 1 2 H

Instruction: CALL	Function: to call a subprogram
Format:	CALL X
Instruction Code:	11000 xxxx xxx xxxx
Carry Flag:	Not affected
Operation:	ST← CY;PC , PC ← X (not include p)

Explanation

Instruction CALL is used to call a subprogram. First it saves the values of CY and PC+1 to stack for returning to the calling program, then goes to the designated address X ($0000H ~ $07FFH or $0800H ~ $0FFFH) to execute program. Instructions RTNW or RTNI can be used to return to the calling program. When using CALL to call a subprogram, you should especially note how many layers of stack have already been used, because SH6610 series only provide 4-layer stacks. If more than 4 layers are used, serious error will be occurred when returning to the calling program!

Instruction: RTNW	Function: to return to the calling program, H→TBR, L→AC
Format:	RTNW H, L
Instruction Code:	11010 000h hhh 1111
Carry Flag:	Not affected
Operation:	PC ← ST , TBR ← hhhh , AC ← 1111

Explanation

RTNW is an instruction to get data from stack to PC for returning to the calling program, and at the same time put the value of H into TBR and the value of L into AC. This instruction is often used to get stationary data.

[Example] To get data from ROM address 302H

```
TBR EQU           OEH
TEMP     EQU      20H
                  :
                  :
001A     LDI      TBR, 00H   ;put index value (high nibble) 0 into TBR.
001B     LDI      TEMP, 02   ;put index value (low nibble) 2 into AC
001C     CALL     300H       ;call subprogram.
001D              :
                  :
                  :
         ORG      300H
0300     TJMP                ; get destination address $0302H according to
(PC11~PC8),TBR,AC
0301     RTNW     00H,01H
0302     RTNW     00H,02H    ;return to main program, H→TBR,L→AC
0303     RTNW     04H,05H
0304     RTNW     09H,08H
0305              :
```

Instruction: RTNI	Function: to return from interrupt or subprogram
Format:	RTNI
Instruction Code:	11010 1000 000 0000
Carry Flag:	CY
Operation:	CY;PC ←ST

Explanation

Instruction RTNI is mainly used for retu rning from interrupt or sub program. It fills CY and PC with values of st ack (CY an d returni ng add ress) wh en returning. What's the difference between RTNI and RTNW? We can find that whe n returning by RTNW, only the returning address in the st ack is fetche d into PC, but CY value is not fetched. And RTNW fetches another two valu es (H→TBR, L→AC), which RTNI does not do. Therefore you can choose from the two instructions according to your needs.

[Example] To exchange two numbers

```
000E        1 TBR          EQU      0EH
0020        2 REGX         EQU      20H
0021        3 REGY         EQU      21H
0022        4 TEMP         EQU      22H
            5 ;********************
0005 780E   12 RESET :     LDI      TEMP,00H    ;set TEMP=00h
0006 7920   13             LDI      REGX,02H    ;set RegX=02h
0007 7A21   14             LDI      REGY,04H    ;set RegY=04h
0008 C00A   15             CALL     SWAPXY      ; ;call subprogram
            17 ;*********************
000A 3820   18 SWAPXY      LDA      REGX,00H    ;AC=02H
000B 3C22   19             STA      TEMP,00H    ;TEMP=02H
000C 3821   20             LDA      REGY,00H    ;AC=04H
000D 3C20   21             STA      REGX,00H    ;REGX=04H
000E 3822   22             LDA      TEMP,00H    ;AC=02H
000F 3C21   23             STA      REGY,00H    ;REG2=02H
0010 D400   24             RTNI                 ;return to main program
```

Instruction: HALT	Function: CPU to be halt from working
Format:	HALT
Instruction Code:	11011 0000 000 0000
Carry Flag:	Not affected
Operation:	No

Explanation

After executing inst ruction HA LT, CPU will be ha lt while it s surrounding circuit (counter , oscillation circuit) continues working. The instruct ion HA LT is usually use d to stop CP U temporarily in order to save power. In HALT mode, when any of the system interrupts occurs, CPU will be released from HALT mode and continue to work.

[Example] HALT program, to enable PORT B interrupt to wake up the program

```
IEX     EQU   00H          ;interrupt enable register
IRQ     EQU   01H          ;interrupt require flag
PORTB   EQU   09H          ;i/o port b
              :
        LDI   PORTB,0FH    ;set port b = " high '
        LDI   IEX,0001B        ;enable port interrupt
        LDI   IRQ,00H          ;clear interrupt require flag
        HALT                   ;system cpu halt
        NOP
              :
```

Instruction: STOP	Function: to stop the whole chip (including oscillation circuit)
Format:	STOP
Instruction Code:	11011 1000 000 0000
Carry Flag:	Not affected
Operation:	No

Explanation

Executing instruction STOP will stop the whole chip from working, incl uding oscillation circuit. Only PORT interrupt and external interrupt can relea se CPU from S TOP mode, so you must en able an inte rrupt befo re e ntering ST OP mode , otherwise the system can't be waked up from STOP mode.

[Example] STOP program, to enable PORT B interrupt to wake up the program

```
IEX     EQU   00H            ;interrupt enable register
IRQ     EQU   01H            ;interrupt require flag
PORTB   EQU   09H            ;i/o port b
              :
        LDI   PORTB,0FH      ;set port b = " high '
        LDI   IEX,0001B      ;enable port interrupt
        LDI   IRQ,00H        ;clear interrupt require flag
        STOP                 ;all system is " stop "
        NOP
              :
```

Instruction: NOP	Function: to do nothing
Format:	NOP
Instruction Code:	1111 1111 111 1111
Carry Flag:	Not affected
Operation:	No

Explanation

Instruction NOP means doing nothing in its instruction cycle and it is often used for time delay. Because it does nothing when executing, you don't worry if it will affect any current status.