5 The Program Status Word

Every microcontroller contains flags that may be used for testing the outcome of an instruction's execution. For example, the carry flag may be used to test the outcome of an 8-bit addition to see if the result is greater than 255.

Some microcontrollers use a special bit to indicate whether the contents of the accumulator is zero or not (the PIC microcontroller, for example). This flag is usually called the zero or Z flag and conditional jump† instructions that test its value can be used to branch (jump to another location in code memory) if the accumulator is zero or if the accumulator is not zero (if Z is set, the accumulator contains zero, if Z is clear the accumulator contains a number other than zero).

The 8051 does not have such a bit. To test the status of the accumulator the instructions `JZ rel` (jump if `(A) = 0`) and `JNZ rel` (jump if `(A) <> 0`) are used.

However, the 8051 contains a number of flags, in the special function register called the Program Status Word (PSW). These flags can be tested by conditional jumps. Before we go into the functions of these flags, it would first be useful to understand how positive and negative numbers are stored in binary.

Signed Numbers

Memory locations and registers in the 8051 are, for the most part, eight bits wide. With eight bits, there are 256 combinations (2^8), as listed (partially) below.

```
0000 0000
0000 0001
0000 0010
0000 0011
   ...
1111 1101
1111 1110
1111 1111
```

If this list of binary numbers represents positive decimal numbers, then the range is 0 to 255, which of course is 256 different numbers.

However, this same list can also represent both positive and negative numbers, the range being –128 to 127.

Note that this still consists of 256 combinations: –1 to –128 make 128 numbers and 0 to 127 make another 128 numbers.

Let's see how negative numbers are stored.

† Unconditional jumps result in program execution resuming at a different location in memory. Conditional jumps test some condition, then jump if the condition is true, or continue with the next instruction if it is false. We will deal with these in more detail in the next chapter.

One's Compliment

To get the one's compliment of a number, each bit is inverted. Three examples are given in the table opposite:

8-bit Number	One's Compliment
1101 0001	0010 1110
1111 1111	0000 0000
0100 0110	1011 1001

Two's Compliment

To get the two's compliment of a number, one is added to the one's compliment. The three examples from the table above are converted into two's compliment here.

8-bit Number	Two's Compliment
1101 0001	0010 1111
1111 1111	0000 0001
0100 0110	1011 1010

To change the two's compliment of a number back to its original value, simply get the two's compliment again. Try it with the examples above.

The Sign Bit

A signed number is a number which can be either positive or negative while an unsigned number can only be positive. With signed numbers, the MSB (most significant bit – in this case, bit 7) is used to determine whether or not the number is positive or negative. If the MSB is zero then the number is positive while if the MSB is one the number is negative.

Therefore, a positive number is stored unchanged. For example, the signed number 86 (in decimal) is stored the same as the unsigned equivalent (0101 0110 is binary for 86).

However, a negative number is stored as the two's compliment of its absolute value. For example, –86 is stored as the two's compliment of 86 (1010 1010 is the 2's compliment of 0101 0110).

Converting Signed Binary Numbers to Decimal

Since an MSB of zero means a positive number and an MSB of one means a negative number, you may be thinking it is necessary for the system to test the MSB in order to determine the sign of a number. This is not so. Converting signed numbers from binary to decimal is exactly the same as converting unsigned numbers from binary to decimal, except for one small difference – the MSB is negative.

For example, let's take the binary number 1101 1100.

If this is an unsigned number, then converting it to decimal gives:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	1	1	0	0
128	64	0	16	8	4	0	0

Adding the values in the bottom row gives 220.

However, if it is a signed number, the conversion is the same, except that the MSB is negative.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	1	1	0	0
-128	64	0	16	8	4	0	0

Adding the values in the bottom row gives –36.

The PSW

Now we will investigate the function of the PSW flags, starting with one of the most commonly used flags, the carry.

Bit	Symbol	Address	Description
PSW.7	CY	D7	Carry Flag
PSW.6	AC	D6	Auxiliary Carry Flag
PSW.5	F0	D5	Flag 0
PSW.4	RS1	D4	Register Bank Select 1
PSW.3	RS0	D3	Register Bank Select 0
PSW.2	OV	D2	Overflow Flag
PSW.1	--	D1	Reserved
PSW.0	P	D0	Even Parity Flag

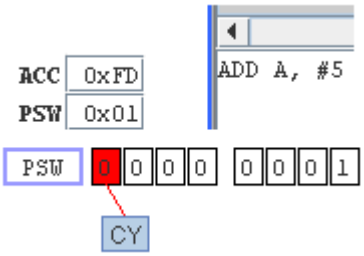
The Program Status Word (PSW)

Carry Flag

The carry flag has two functions.

1. Firstly, it is used as the carry-out in 8-bit addition/subtraction. For example, if the accumulator contains FDH and we add five to its contents, it will then contain two and the carry flag will be set. It is also set if a subtraction causes a borrow into bit 7. In other words, if a number is subtracted from a number smaller than it, the carry flag will be set. For example, if A contains 3DH and R3 contains 4BH, the instruction `SUBB A, R3` will result in the carry bit being set (4BH is greater than 3DH).
2. The carry flag is also used during Boolean operations. For example, we could AND the contents of bit 73H with the carry flag, the result being placed in the carry flag – `ANL C, 73H` (the bit at address 73H is logically anded with the carry, the result placed in the carry).

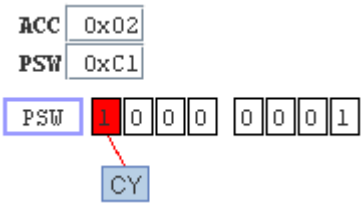
Below are screenshots of the EdSim51 simulator, illustrating these two examples of the carry in action.



<- The contents of ACC is FDH (253 in decimal), and the carry (bit 7 of the PSW) is zero, prior to execution of the instruction that adds five to the accumulator.

After adding five, ACC contains two and the carry bit is set, indicating the result of the addition (253 + 5) is greater than 255.

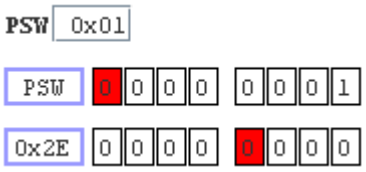
->



<- In this example, the carry is one and bit 73H is zero (if you look at the memory map, you will see bit 73H is bit 3 of byte location 2EH, as highlighted opposite.

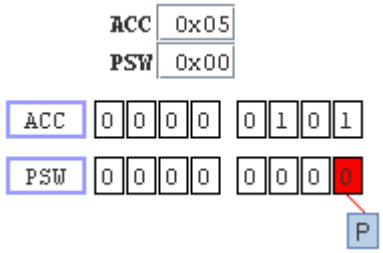
After anding the carry with bit 73H, it can be seen that the result (anding one with zero results in zero) is placed in the carry.

->



Parity Bit

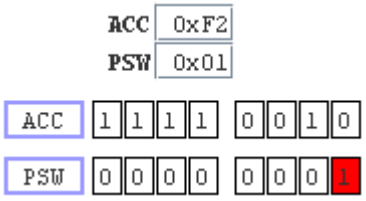
The parity bit is automatically set or cleared every machine cycle‡ to ensure even parity with the accumulator. The number of ones in the accumulator plus the parity bit is always even. In other words, if the number of ones in the accumulator is odd then the parity bit is set to make the overall number of bits even. If the number of ones in the accumulator is even then the parity bit is cleared to make the overall number of bits even.



<- For example, if the accumulator holds the number five (see opposite), then it has an even number of ones. Therefore the parity bit is cleared.

If the accumulator holds the number F2H, it has an odd number of ones. Therefore the parity bit is set to make the overall number of ones even.

->

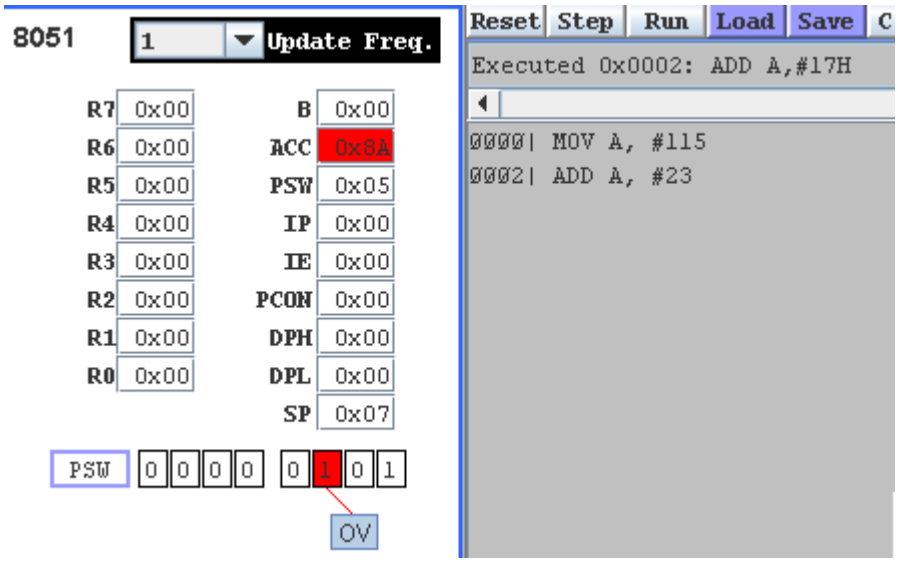


As we shall see later in this book, the parity bit is most often used for detecting errors in transmitted data.

Overflow Flag

The overflow flag is bit 2 of the PSW. This flag is set after an addition or subtraction operation if the result in the accumulator is outside the signed 8-bit range (–128 to 127). In other words, if the addition or subtraction of two numbers results in a number less than –128 or greater than 127, the OV flag is set.

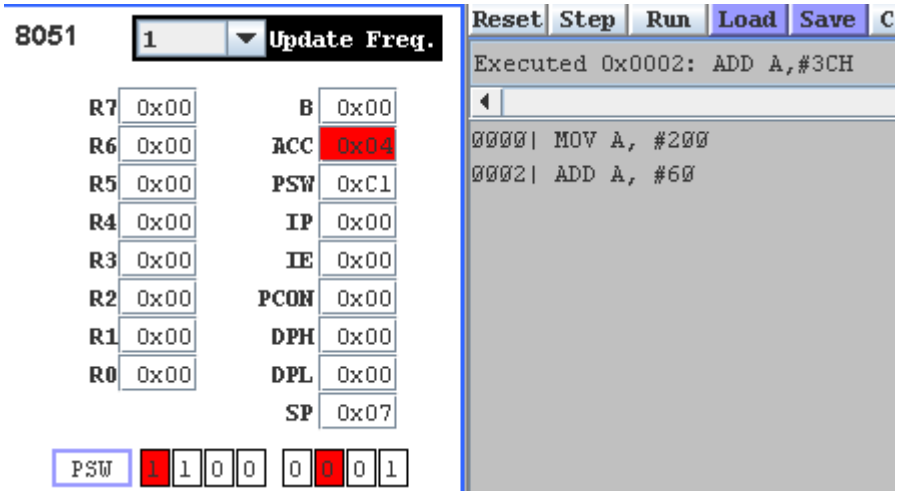
When signed numbers are added or subtracted, software can check this flag to see if the result is in the range –128 to 127. For example: 115 + 23 = 138 (73H + 17H = 8AH). If these numbers are being treated as signed numbers then 8AH is (as a signed number) –118 in decimal. Obviously, 115 + 23 is not equal to –118. The problem lies with the fact that the correct answer (138) is too big to be represented by an 8-bit signed number. Therefore, the OV flag is set to alert the program that the result is out of range.



‡ The 8051 (like all microcontrollers), carries out a number of operations, such as reading from ROM and updating the parity bit, during what is known as the machine cycle. Most one-byte instructions take one machine cycle to execute, whereas two-byte instructions take two machine cycles, as ROM needs to be accessed twice.

You may wonder what happens if the sum of two numbers is outside the range of an unsigned number. For example: 200 + 60 = 260. The result is a 9-bit number and the carry flag is set. However, the result is also greater than 127 (the 8-bit signed number maximum) so you might expect the OV flag to be set also. But, if you test this in the EdSim51 simulator you will notice OV is not set. Why?

The answer is quite simple: the sum in HEX is: C8H + 3CH. Regardless of whether we are dealing with signed or unsigned numbers, 3CH is equal to 60 in decimal. However, C8H as an unsigned number is 200 in decimal, but as a signed number it's –56 in decimal. When deciding the value of the OV flag, only the case of signed numbers is taken into account. So, this equates to –56 + 60 = 4.



If you run this code in the simulator you will see that the accumulator contains 4, the carry is set to indicate that, if this is unsigned arithmetic, the answer is greater than 255, but OV is clear because if this is signed arithmetic the answer is in the range –128 to 127.

Auxiliary Carry Flag (AC)

The auxiliary carry flag is set or cleared after an add instruction (`ADD A, operand` or `ADDC A, operand`) only. The condition that results in AC being set is:

If a carry was generated out of bit 3 into bit 4 of the accumulator.

For example, 8 + 9 = 17. In binary, this is 0000 1000 + 0000 1001 = 0001 0001. Notice that, in both numbers, bit 3 is one. Therefore, adding them together results in a carry of one into bit 4.

To understand the purpose of this flag, we first need to look at binary coded decimal (BCD) and why it is useful.

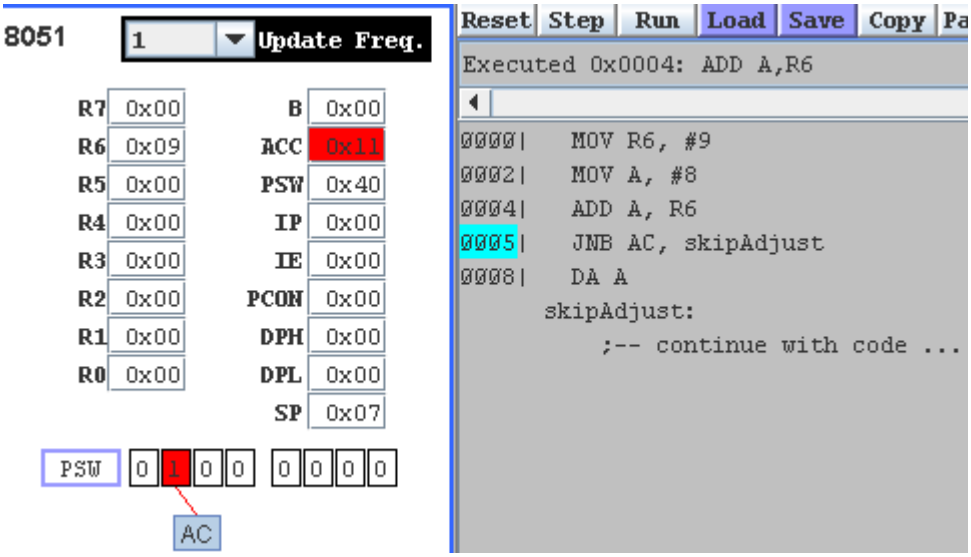
```
0000 0000    0  |  0001 0000    10
0000 0001    1  |  0001 0001    11
0000 0010    2  |  0001 0010    12
0000 0011    3  |  ...
0000 0100    4  |  0001 1001    19
0000 0101    5  |  0010 0000    20
0000 0110    6  |  0010 0001    21
0000 0111    7  |  0010 0010    22
0000 1000    8  |  ...  and so on
0000 1001    9  |
```

In 8-bit BCD, the byte is split into two 4-bit nibbles. Each nibble has a range of zero to nine. This can be useful is we wish to output the BCD number to a display. For example, the number 21 in decimal is 15 in HEX. If we were to output this to a display, where the lower nibble represents the units and the upper nibble represents the tens, the number 15 would appear on the display, instead of the actual number 21. We could use the `DA A` (decimal adjust A) instruction that would change the value in the accumulator from 15H to 21H. Then, if this was output to the display, the upper nibble would display a two, while the lower nibble would display a one – the number 21 is displayed.

The AC flag may be tested after an addition to see if there was a carry from the lower nibble to the higher nibble. If so,

this means the BCD addition resulted in an overflow, and the instruction `DA A` can be used to change the HEX code in A back to BCD.

For example, if we add 9 to 8 in the accumulator, as shown below:



The above code adds 8 to 9, leaving 17 (11H) in the accumulator. As explained on the previous page, this means there was a carry from the lower nibble to the higher nibble. Therefore AC will be set and the following instruction (`DA A`) will change A from 11H to 17H.

Register Bank Select Bits

Bits 3 and 4 of the PSW are used for selecting the register bank. Since there are four register banks, two bits are required for selecting a bank, as detailed below.

PSW.4 (RS1)	PSW.3 (RS0)	Register Bank	Address of Register Bank
0	0	0	00H to 07H
0	1	1	08H to 0FH
1	0	2	10H to 17H
1	1	3	18H to 1FH

For example, if we wished to activate register bank 3, we would use the following instructions:

```
SETB RS1    ; set register select bit 1
SETB RS0    ; set register select bit 0
```

If we then moved the contents of R4 to the accumulator (`MOV A, R4`) we would be moving the data from location 1CH to accumulator. However, the beginning programmer seldom, if ever, bothers with moving the register bank.

This is one sample chapter from the soon to be published *EdSim51's Beginner's Guide to the 8051* by James Rogers. The book is expected to be available by early March, 2009. Check **www.edsim51.com** for latest news.