flags and conditional jumps

by Jeremy Gordon - jg@jgnet.co.uk

This file is intended for those interested in 32 bit assembler programming, in particular for Windows.

The "flags" are each one bit of memory contained within the processor itself. Since each flag is only one bit it is either 1 or 0 ("set" or "clear") at any one time. There are six flags which are used to indicate the result of certain instructions. Some instructions such as CMP, TEST and BT only alter some of these flags and do nothing else. Other instructions carry out other operations but also alter some of the flags. Some instructions don't alter the flags at all. When looking at each mnemonic the **mnemonic manual** will tell you which flags are changed by which instructions (not yet available).

A common use for the flags is to divert execution to a particular part of code using the conditional jump instructions. These instructions will jump or will not jump depending on the state of one or more of the flags. Only five of the flags can be used in this way - zero, sign, carry, overflow and parity. The sixth flag (auxiliary carry) and seventh flag (direction flag) are read by other instructions. Here is more information about the five flags which can be used by the conditional jump instructions:-

Z (zero flag)

Set if the result is zero. If after an arithmetic instruction the number left in the register or memory area which is the subject of the instruction is zero, then this flag is set. Often you just need to do a simple comparison of two values without changing them. In that case you can use the instruction CMP. CMP does a pretend SUB without actually changing the values given to it as operands. For example:-

```
CMP EAX,33h       ;set zero flag if eax=33h, but don't change eax
SUB EAX,33h       ;set zero flag if eax=33h (eax now 33h less)
CMP EAX,EDX       ;set zero flag if eax=edx
CMP EAX,[VALUE]   ;set zero flag if eax=the number in VALUE
```

The zero flag can also be used to show the result of a count down or up, for example

```
DEC EAX           ;set zero flag if eax is zero after instruction clear if not
INC EAX           ;set zero flag if eax is zero after instruction clear if not
```

The zero flag can also be used to control the repeating of the string instructions ie. LODS, STOS and MOVS

```
REPZ              ;repeat the string instruction while zero flag is set
REPNZ             ;repeat the string instruction while zero flag is not set
```

A return from an API in Windows often indicates that the API has failed. So you will often need to check for this event. When testing registers you can use these alternatives:-

```
CMP EAX,0         ;see if the number in eax is zero (zero flag set if so)
OR EAX,EAX        ;this does exactly the same but uses 2 opcodes instead of 3
TEST EAX,EAX      ;again this is the same and uses only 2 opcodes
```

The 16-bit and 8-bit versions of the instructions test only the first 16 or 8 bits of the register or memory area respectively, for example

```
CMP W[DAVID],0    ;see if the first 16 bits of memory labelled DAVID are zero
CMP B[SUE],0      ;see if the first 8 bits of memory labelled SUE are zero
OR DX,DX          ;see if the dx register is zero (16-bit register)
TEST DH,DH        ;see if the dh register is zero (8-bit register)
SUB B[VALUE],2    ;reduce the 8 bits at VALUE by 2 (zero flag set if now zero)
DEC SI            ;zero flag set when si is zero (16-bit register)
INC B[COUNT]      ;zero flag set when 8 bits at COUNT are zero
```

Since the flags are very useful when returning from a routine to show whether the routine was successful or not you will sometimes need to set them expressly. In order to set the zero flag you can use:-

```
CMP EAX,EAX       ;set the zero flag (no change to eax)
SUB EAX,EAX       ;set the zero flag (making eax=0 too)
CMP EAX,EDX       ;when they must be different clear the zero flag
OR EAX,EAX        ;when eax cannot be zero clear the zero flag
TEST EAX,EAX      ;same effect as OR EAX,EAX
```

When used with TEST, the zero flag will be set if the bit being tested is zero.

```
MOV ECX,1         ;give ecx the value 1
TEST ECX,1        ;the zero flag is not set
CMP ECX,1         ;the zero flag is set
MOV EDX,0
TEST EDX,1        ;the zero flag is set
CMP EDX,1         ;the zero flag is not set
MOV EBX,-1
TEST EBX,-1       ;test all 32 bits - zero flag not set
CMP EBX,-1        ;see if ebx is -1 - zero flag set
```

The zero flag is mainly used with the JZ and JNZ conditional jump instructions for example:-

```
JZ >L10           ;jump forward to L10 if the zero flag is set (to 1)
```

```
JNZ L1           ;jump back to L1 if the zero flag is clear
```
The zero flag is also used in the JA (jump if above), JB (jump if below) and similar conditional jump instructions.

It can also be used within loops using special instructions or on its own, for example:-

```
L1:
                 ;other code here
CMP EDX,EAX
LOOPZ L1         ;decrement ecx, continue to loop until ecx=0
                 ;or until edx=eax (when zero flag will be set)
;*******
L1:
                 ;other code here
CMP EDX,EAX
LOOPNZ L1        ;decrement ecx, continue to loop until ecx=0
                 ;or until edx does not=eax (when zero flag will be clear)
;*******
L1:
                 ;other code here
CMP EDX,EAX
JZ >L10          ;jump out of loop when edx=eax (zero flag set)
LOOP L1          ;decrement ecx, continue to loop until ecx=0
L10:
;*******
L1:
                 ;other code here
CMP EDX,EAX
JNZ L1           ;continue to loop until edx=eax (zero flag set)
```

S (sign flag)

Set if the most significant bit (the *leftmost* bit) of the result is 1. The position of this bit depends on the data size. In a byte the most significant bit is bit 7 (8th bit of bits 0 to 7); in a word it is bit 15 (16th bit of bits 0 to 15) and in a dword it is bit 31 (32nd bit of bits 0 to 31). So this bit will be set if the result of the instruction is 80h or higher (for a byte), 8000h or higher (for a word) or 80000000h or higher (for a dword). Note that in signed numbers the most significant bit indicates whether the number is negative or not.
The sign flag is altered by INC and DEC whereas the carry flag is not, so testing the sign flag is often useful in loops, for example

```
L0:
;
DEC ECX          ;reduce ecx by one
JNS L0           ;loop back to L0 if ecx is not yet -1
```
It can also be conveniently used in multi-action functions for example:-
```
MULTI_ACTION:    ;on entry al holds the action to take
DEC AL           ;see if al held zero
JS >L0           ;yes
DEC AL           ;see if al held one
JS >L1           ;yes
DEC AL           ;see if al held two
JS >L2           ;yes
DEC AL           ;see if al held three
JS >L3           ;yes
DEC AL           ;see if al held four
JS >L4           ;yes
```
Using the sign flag is a convenient way to see if the high bit of a register is set or cleared. A number of instructions set the flag without altering the register for example:-
```
OR EDX,EDX       ;set the sign flag if the high bit of edx is set
CMP EDX,EDX      ;   - ditto -
TEST EDX,EDX     ;   - ditto -
OR CL,CL         ;set the sign flag if the high bit of cl is set
CMP CL,CL        ;   - ditto -
TEST CL,CL       ;   - ditto -
```
When checking areas of memory though, you can only address the memory once per instruction so you need to use CMP, for example:-
```
CMP B[DATA44],0       ;set the sign flag if the 8th bit of DATA44 is set
CMP W[DATA44],0       ;set the sign flag if the 16th bit of DATA44 is set
CMP D[DATA44],0       ;set the sign flag if the 32nd bit of DATA44 is set
CMP B[DATA44+7],0     ;set the sign flag if the 64th bit of DATA44 is set
CMP B[DATA44+9],0     ;set the sign flag if the 80th bit of DATA44 is set
```
Note that the position of the high bit in the area of memory known as DATA44 which is used in these instructions depends on the data size used in the instruction. This is because data in memory areas is stored in reverse-byte order, the least significant byte first and the most significant last. See understand reverse storage. The instruction CMP B[DATA44+7],0 looks at the 8th byte which holds the 64th bit. This is the sign but for a 64 bit data size.

The sign flag is mainly used with the JS and JNS conditional jump instructions for example:-

```
JS >L10          ;jump forward to L10 if the sign flag is set (to 1)
JNS L1           ;jump back to L1 if the sign flag is clear
```

The sign flag is also used in the JG (jump if greater-than), JNG (jump if not greater-than) and similar conditional jump instructions.

C (carry flag)

Set if the result of the instruction has gone beyond the limit of the data size (ie. a "carry" has occurred). For example suppose in an 8 bit instruction the value of 1 is added to 255. This can't make 256 since 255 is the data limit for a byte. So the result will be 0, but the carry flag will be set. Similarly suppose in an instruction the value of 4 is subtracted from 2. Again this causes the carry flag to be set because the result has gone below zero which is the lower limit of the data size. The carry flag therefore indicates that an overflow has occurred when using unsigned numbers. See overflow flag for finding oveflows when signed numbers are being used.
Unlike other flags there are instructions which are designed to manipulate the carry flag directly:-

```
STC              ;set carry flag
CLC              ;clear carry flag
CMC              ;complement carry flag
```

Since these instructions are so simple the carry flag is very useful to report the result of a function to its caller, for example:-

```
CALCULATE2:
;
CMP EBX,ESI
JZ >.fail        ;jump to fail if ebx=esi
CMP EAX,ESI
JZ >.success     ;jump to success if eax=esi
.fail
STC              ;set carry to show fail
RET
.success
CLC              ;clear carry to show success
RET
;
CALCULATE1:
CALL CALCULATE2
JC >L40          ;jump forward to L40 if not successful
```

Note that INC and DEC do not alter the carry flag. Nor do the loop instructions. This is useful if you have a loop which needs to report its result using the carry flag for example:-

```
.loop
;
CMP ESI,EDI      ;see if esi is below edi (set carry if so)
DEC ECX          ;see if any more loops to do
JNZ .loop        ;yes
RET              ;return with the result of cmp esi,edi in the carry flag
```

There are some instructions which always clear the carry flag. This is useful to know to avoid the need for CLC if you want to clear the carry flag. These instructions are AND, OR and TEST.
Some instructions respond to the carry flag as input or give their output in the carry flag. See the mnemomics concerned.

The carry flag is mainly used with the JC and JNC conditional jump instructions for example:-

```
JC >L10          ;jump forward to L10 if the carry flag is set
JNC L1           ;jump back to L1 if the carry flag is clear
```

The carry flag is also used in the JA (jump if above), JB (jump if below) and similar conditional jump instructions.

O (overflow flag)

To understand the overflow flag you need to understand about signed numbers. The overflow flag is used to indicate an overflow when using signed numbers. The carry flag cannot be used for this. A simple example suffices to prove this:-

```
MOV AL,0FEh      ;give al 254 decimal (unsigned) or -2 (signed)
ADD AL,4h        ;add 4 - al now holds 2h
```

here the carry flag will be set because the unsigned result of 258 was too large for the data size limit of 255. But regarded as a signed calculation, there was no overflow. The value 2 now in al is the correct result of -2 +4. The overflow flag is clear.
Here is another example where there **is** an overflow in a signed calculation:-

```
MOV AL,7Fh       ;give al 127 decimal
ADD AL,4h        ;add 4 - al now holds 83h
```

here the carry flag is clear because the unsigned result of 131 is within the data size limit of 255. But regarded as a signed calculation, there was an overflow because if al holds 83h this is the signed number -125 decimal which is the wrong result. The correct result of 131 is outside the signed range of -127 to +128.
So in these type of arithmetic operations the processor sets the overflow flag if the sign bit changes but there is no "carry". This is independent of the carry flag as can be seen from:-

```
MOV AL,7Fh       ;give al 127 decimal
INC AL           ;cause overflow (carry unaffected by INC)
;
MOV AL,80h       ;give al -128 decimal
DEC AL           ;cause overflow (carry unaffected by DEC)
```

In the shift instructions *only for single shift operations* does the overflow flag give a valid indication whether the signed result is too large for the data size. For example:-

```
MOV AL,80h       ;give al -128
SHL AL,1         ;x2 cause overflow
MOV AL,0FEh      ;give al -2
SHL AL,1         ;x2 result -4 no overflow
MOV AL,80h       ;give al -128
SHL AL,2         ;x4 no overflow indicated
MOV AL,0FEh      ;give al -2
SAR AL,1         ;/2 result -1 no overflow
```

The SAR instruction is a special signed shift right instruction which maintains the correct sign in the result. It does this by shifting all the bits *except* the high bit. Because a single SAR shift is effectively a divide by +2 it can never overflow. But SHL can, and in single shift operations the overflow flag is set appropriately. To achieve this, the processor tests to see if the sign bit is the same as the carry flag and clears the overflow flag if it is. Because of this test, it is possible to identify another use for the overflow flag as follows (*note these tests change the contents of the register*):-

```
SHL AL,1
JNO >L1          ;jump if the highest two bits of al were the same
SHL AL,1
JO >L1           ;jump if the highest two bits of al were different
SHL EAX,1
JNO >L1          ;jump if the highest two bits of eax were the same
SHL EAX,1
JO >L1           ;jump if the highest two bits of eax were different
```

The rotate instructions work in the same way. Since the ROR instruction shifts all bits to the right replacing the highest bit with the lowest bit, this provides a way to compare the highest bit and the lowest bit of data. For example (*note these tests change the contents of the register*):-

```
ROR AL,1
JNO >L1          ;jump if the lowest and highest bits of al were the same
ROR AL,1
JO >L1           ;jump if the lowest and highest bits of al were different
ROR EAX,1
JNO >L1          ;jump if the lowest and highest bits of eax were the same
ROR EAX,1
JO >L1           ;jump if the lowest and highest bits of eax were different
```

The special signed multiply instruction IMUL sets the overflow flag if the signed result is too large for the data size.

The overflow flag is mainly used with the JO and JNO conditional jump instructions for example:-

```
JO >L10          ;jump forward to L10 if the overflow flag is set
JNO L1           ;jump back to L1 if the overflow flag is clear
```

P (parity flag)

The parity flag indicates whether there are an even or odd number of bit set in the data. The parity flag is set if the number of bits set is even and cleared if it is odd. In serial communications, the parity bit is used as an unsophisticated error check. After each byte is sent, the transmitter sends a*parity bit* which tells the receiver whether the byte just sent should have been even or odd parity. This might miss a single corrupted byte but does usually detect a series of corrupted ones. When used in this way a byte may be less than 8 bits: 7 bits plus a parity bit is often used for serial transmissions.

The parity flag is mainly used with the JP and JNP conditional jump instructions for example:-

```
JP >L10          ;jump forward to L10 if the parity flag is set
JNP L1           ;jump back to L1 if the parity flag is clear
```

The auxiliary carry flag is used in Binary Coded Decimal (BCD) arithmetic. Unlike the other flags this flag is not used to divert execution depending on whether or not it is set. Instead it is set by one BCD instruction and then read by the next BCD instruction. For more information see **understand bcd arithmetic**. (Not yet available).

Conditional jump instructions - unsigned

Instruction	Alternative	Action
JZ	JE	Jump if zero flag set (jump if equal)
JNZ	JNE	Jump if zero flag not set (jump if not equal)
JC	JB or JNAE	Jump if carry flag set (jump if "below" or not above or equal)
JNC	JNB or JAE	Jump if carry flag not set (jump if not "below" or above or equal)
JA	JNBE	Jump if neither carry flag or zero flag set (ie. jump if "above" or if not "below" or equal). See note for JNA below.

JNA	JBE	Jump if either carry flag or zero flag set (ie. jump if "not above" or if "below" or equal). This is useful to check after a subtraction that a register remains above 1. For example, you might use this in protective coding as follows (where EDI is at the end of a string in BUFFER):-
		`MOV EDX,ADDR BUFFER`
		`SUB EDI,EDX ;get length of string in EDI`
		`JNA >.error ;length found to be negative or zero`

JP		Jump if parity flag set
JNP		Jump if parity flag not set
JECXZ		Jump if ECX=0
JCXZ		Jump if CX=0

Conditional jump instructions - signed

Instruction	Alternative	Action
JS		Jump if sign flag set (jump if signed number is negative)
JNS		Jump if sign flag not set (jump if signed number is positive)
JO		Jump if overflow flag set
JNO		Jump if overflow flag not set
JL	JNGE	Jump if sign and overflow flags are different (jump if "less" or "not greater or equal"). This special test is necessary for numbers regarded as signed because JA (jump if above) and JB (jump if below) do not work with such numbers. For example if AL is 1 then CMP AL,-1 will not cause JB to jump since the carry flag is not set. But JL will jump because (regarded as signed) -1 is *less than* 1.
JNL	JGE	Jump if sign and overflow flags are the same (jump if "not less" or "greater or equal"). See note above for JL.
JLE	JNG	Jump if sign and overflow flags are different and zero flag is set (jump if "less or equal" or "not greater")
JLNE	JG	Jump if sign and overflow flags are the same and zero flag is clear (jump if "not less or equal" or "greater")

Copyright © Jeremy Gordon 2002-2003