The Intel 80386, part 3: Flags and condition codes

devblogs.microsoft.com/oldnewthing/20190122-00

January 23, 2019

Raymond Chen

The flags register contains a bunch of stuff, but here are the flags easily accessible in the debugger:

Flag	Clear/Set	Meaning	Notes
OF	nv/ov	Overflow	
DF	up/dn	Direction	Must be *up* at function boundaries
SF	pl/ng	Sign	
IF	ei/di	Interrupts	Set if interrupts are enabled
ZF	nz/zr	Zero	
AF	na/ac	Auxiliary carry	Not used by C code
PF	pe/po	Parity	Not used by C code
CF	nc/cy	Carry	

We'll learn about the direction flag when we get to string operations. The important detail for now is that the direction flag must be clear (*up*) at function boundaries.

Instructions for manipulating the interrupt flag are privileged, so you won't see user-mode code messing with it. I wouldn't normally have mentioned it, but the Windows disassembler displays the state of the interrupt flags in the register output, so I included it here just so you can see what it means (and then promptly forget about it).

The auxiliary carry is used to indicate whether a carry occurred between bits 3 and 4. It is used by the binary coded decimal instructions.

The parity is used to indicate whether the number of set bits in the least significant 8 bits of the result is odd or even.

The *Clear/Set* column denotes how the Windows disassembler represents flags in the register output:

```
eax=00000000 ebx=00000000 ecx=9f490000 edx=00000000 esi=7f19e000 edi=00000000
eip=77a93dad esp=0048f844 ebp=0048f870 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
```

The *efl* represents the value of the 32-bit flags register, and selected bits are parsed out and rendered as mnemonics on the line above.

Various combinations of conditions can be expressed with condition codes. Note that many conditions have multiple mnemonics. The first one listed is the one the disassembler uses.

Code	Meaning	Condition	Notes
`E`	Equal	`ZF`	
`Z`	Zero		
`NE`	Not equal	`!ZF`	
`NZ`	Not zero		
`A`	Above	`!CF && !ZF`	Unsigned greater than
`NBE`	Not below or equal		
`AE`	Above or equal	`!CF`	Unsigned greater than or equal
`NB`	Not below		
`NC`	No carry		No unsigned overflow
`B`	Below	`CF`	Unsigned less than
`NAE`	Not above or equal		
`C`	Carry set		Unsigned overflow
`BE`	Below or equal	`CF || ZF`	Unsigned less than or equal
`NA`	Not above		
`G`	Greater	`!(SF ^ OF) && !ZF`	Signed greater than
`NLE`	Not less than or equal		
`GE`	Greater than or equal	`!(SF ^ OF)`	Signed greater than or equal
`NL`	Not less than		

`L`	Less than	`(SF ^ OF)`	Signed less than
`NGE`	Not greater than or equal		
`LE`	Less than or equal	`(SF ^ OF) || ZF`	Signed less than or equal
`NG`	Not greater than		
`S`	Sign	`SF`	Negative
`NS`	No sign	`!SF`	Positive or zero
`O`	Overflow	`OF`	Signed overflow
`NO`	No overflow	`!OF`	No signed overflow
`P`	Parity	`PF`	Even number of bits set
`PE`	Parity even		
`NP`	No parity	`!PF`	Odd number of bits set
`PO`	Parity odd		

The overflow and parity conditions are not normally used by C code. Note also that many flags are not testable via condition codes. (Poor auxiliary carry flag. Nobody loves you.)

There are a few instructions for directly manipulating selected flags:

```
STC         ; set carry
CLC         ; clear carry
CMC         ; complement (toggle) carry

STD         ; set direction (go down)
CLD         ; clear direction (go up)
```

Controlling the interrupt flag is a privileged instruction, so you won't see it in user-mode code. There are no instructions for directly manipulating the other flags, but you can manipulate them indirectly by performing an arithmetic operation with a known effect on flags. For example, you can force *ZF* to be set by performing a calculation whose result is known to be zero, such as `XOR EAX, EAX`.

Okay, that was extremely boring, but it had to be done. Next time, we'll start doing arithmetic.

Raymond Chen

Follow