WIKIPEDIA

FLAGS register

The **FLAGS** register is the status register that contains the current state of a CPU. The size and meanings of the flag bits are architecture dependent. It usually reflects the result of arithmetic operations as well as information about restrictions placed on the CPU operation at the current time. Some of those restrictions may include preventing some interrupts from triggering, prohibition of execution of a class of "privileged" instructions. Additional status flags may bypass memory mapping and define what action the CPU should take on arithmetic overflow.

The carry, parity, adjust, zero and sign flags are included in many architectures. The adjust flag used to be called auxiliary carry bit in 8080 and half-carry bit in the Zilog Z80 architecture.

In i386 architecture the register is 16 bits wide. Its successors, the **EFLAGS** and **RFLAGS** registers, are 32 bits and 64 bits wide, respectively. The wider registers retain compatibility with their smaller predecessors.

Contents

FLAGS

Intel x86 FLAGS register[1]						
Bit #	**Mask**	**Abbreviation**	**Description**	**Category**	**=1**	**=0**
FLAGS						
0	0x0001	CF	Carry flag	Status	CY(Carry)	NC(No Carry)
1	0x0002		Reserved, always 1 in **EFLAGS** [2][3]			
2	0x0004	PF	Parity flag	Status	PE(Parity Even)	PO(Parity Odd)
3	0x0008		Reserved[3]			
4	0x0010	AF	Adjust flag	Status	AC(Auxiliary Carry)	NA(No Auxiliary Carry)
5	0x0020		Reserved[3]			
6	0x0040	ZF	Zero flag	Status	ZR(Zero)	NZ(Not Zero)
7	0x0080	SF	Sign flag	Status	NG(Negative)	PL(Positive)
8	0x0100	TF	Trap flag (single step)	Control		
9	0x0200	IF	Interrupt enable flag	Control	EI(Enable Interrupt)	DI(Disable Interrupt)
10	0x0400	DF	Direction flag	Control	DN(Down)	UP(Up)
11	0x0800	OF	Overflow flag	Status	OV(Overflow)	NV(Not Overflow)
12-13	0x3000	IOPL	I/O privilege level (286+ only), always 1 on 8086 and 186	System		
14	0x4000	NT	Nested task flag (286+ only), always 1 on 8086 and 186	System		
15	0x8000		Reserved, always 1 on 8086 and 186, always 0 on later models			
EFLAGS						
16	0x0001 0000	RF	Resume flag (386+ only)	System		
17	0x0002 0000	VM	Virtual 8086 mode flag (386+ only)	System		
18	0x0004 0000	AC	Alignment check (486SX+ only)	System		
19	0x0008 0000	VIF	Virtual interrupt flag (Pentium+)	System		

20	0x0010 0000	VIP	Virtual interrupt pending (Pentium+)	System
21	0x0020 0000	ID	Able to use CPUID instruction (Pentium+)	System
22-31	0xFFC0 0000		Reserved	System
RFLAGS				
32-63	0xFFFF FFFF… …0000 0000		Reserved	

Note: The mask column in the table is the AND bitmask (as hexadecimal value) to query the flag(s) within FLAGS register value.

Usage

All FLAGS registers contain the condition codes, flag bits that let the results of one machine-language instruction affect another instruction. Arithmetic and logical instructions set some or all of the flags, and conditional jump instructions take variable action based on the value of certain flags. For example, `jz` (Jump if Zero), `jc` (Jump if Carry), and `jo` (Jump if Overflow) depend on specific flags. Other conditional jumps test combinations of several flags.

FLAGS registers can be moved from or to the stack. This is part of the job of saving and restoring CPU context, against a routine such as an interrupt service routine whose changes to registers should not be seen by the calling code. Here are the relevant instructions:

- The PUSHF and POPF instructions transfer the 16-bit FLAGS register.
- PUSHFD/POPFD (introduced with the i386 architecture) transfer the 32-bit double register EFLAGS.
- PUSHFQ/POPFQ (introduced with the x64 architecture) transfer the 64-bit quadword register RFLAGS.

In 64-bit mode, PUSHF/POPF and PUSHFQ/POPFQ are available but PUSHFD/POPFD are not.[4]:4−349,4−432

The lower 8 bits of the FLAGS register is also open to direct load/store manipulation by SAHF and LAHF (load/store AH into flags).

Example

The ability to push and pop FLAGS registers lets a program manipulate information in the FLAGS in ways for which machine-language instructions do not exist. For example, the `cld` and `std` instructions clear and set the direction flag (DF), respectively; but there is no instruction to complement DF. This can be achieved with the following assembly code:

```
pushf          ; Use the stack to transfer the FLAGS
pop   ax       ; ...into the AX register
push  ax       ; and copy them back onto the stack for storage
xor   ax, 400h ; Toggle (complement) DF only; other bits are unchanged
```

```
push  ax        ; Use the stack again to move the modified value
popf            ; ...into the FLAGS register
; Insert here the code that required the DF flag to be complemented
popf           ; Restore the original value of the FLAGS
```

By manipulating the FLAGS register, a program can determine the model of the installed processor. For example, the alignment flag can only be changed on the 486 and above. If the program tries to modify this flag and senses that the modification did not persist, the processor is earlier than the 486.

Starting with the Intel Pentium, the CPUID instruction reports the processor model. However, the above method remains useful to distinguish between earlier models.

See also

- Bit field
- Control register
- CPU flag (x86)
- Program status word
- Status register
- x86 assembly language
- x86 instruction listings

References

1. *Intel 64 and IA-32 Architectures Software Developer's Manual* (http://download.intel.com/products/processor/manual/253665.pdf#page=93) (PDF). **1**. May 2012. pp. 3–21.
2. *Intel 64 and IA-32 Architectures Software Developer's Manual* (https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf#page=78) (PDF). **1**. Dec 2016. p. 78.
3. "Silicon reverse engineering: The 8085's undocumented flags" (http://www.righto.com/2013/02/looking-at-silicon-to-understanding.html). *www.righto.com*. Retrieved 2018-10-21.
4. *Intel 64 and IA-32 Architectures Software Developer's Manual* (http://download.intel.com/products/processor/manual/253667.pdf#page=351) (PDF). **2B**. May 2012.