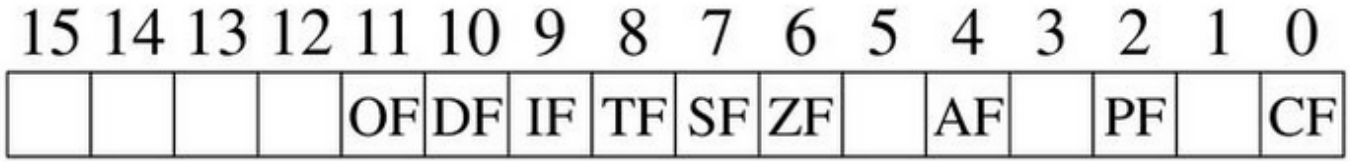
Языки программирования (Asm)

Занятие №5. Флаги. Команды для работы с данными.

МИРЭА - РТУ, БК №252

Москва 2020

Регистр флагов



Регистр флагов отличается от всех остальных регистров тем, что процессор работает с его отдельными битами. Значения битов данного регистра влияют на выполнение ассемблерных инструкций.
Все флаги устанавливаются процессором автоматически в процессе выполнения ассемблерных инструкций, однако существуют также специальные команды для установки отдельных флагов в необходимое состояние. Например, команда STC устанавливает флаг переноса (CF) в значение 1, а CLC сбрасывает в 0.
Некоторые биты регистра флагов зарезервированы и не используются. К ним, например, относятся 1,3,5 и 15 биты 16-битного регистра FLAGS.

Регистр флагов

Здесь мы не будем рассматривать все имеющиеся флаги. Из всех флагов нам понадобятся 8 флагов, находящихся в 16-битном регистре FLAGS:

1. CF Carry Flag Флаг переноса
2. PF Parity Flag Флаг чётности
3. OF Overflow Flag Флаг переполнения
4. SF Sign Flag Флаг знака
5. ZF Zero Flag Флаг нуля
6. AF Auxiliary Carry Flag Вспомогательный флаг переноса
7. IF Interrupt Enable Flag Флаг разрешения прерываний
8. DF Direction Flag Флаг направления

AF / IF / DF

Auxiliary Carry Flag Вспомогательный флаг переноса (или флаг полупереноса). Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый. Этот флаг используется автоматически командами двоично-десятичной коррекции.
Interrupt Enable Flag Флаг разрешения прерываний. Если сбросить этот флаг в 0, то процессор перестанет обрабатывать прерывания от внешних устройств. Обычно его сбрасывают на короткое время для выполнения критических участков программы.
Direction Flag Флаг направления. Контролирует поведение команд обработки строк. Если установлен в 1, то строки обрабатываются в сторону уменьшения адресов, если сброшен в 0, то наоборот.

Команды PUSHF / POPF

Для сохранения регистра флагов на стеке существует команда PUSHF. Она предназначена для работы с 16-битным регистром флагов. В 32-битном режиме вместо нее используется команда PUSHFD, которая соответственно заносит на вершину стека значение регистра EFLAGS. Для 64-битного режима существует аналогичная команда PUSHFQ, которая заносит в стек значение регистра RFLAGS

```
1 pushfd ; Для x86
2 pushfq ; Для x64
```

Для того чтобы получить значение регистра флагов с вершины стека используются команды POPF, POPFD, POPFQ

```
1 popfd ; Для x86
2 popfq ; Для x64
```

Команды для работы с данными

Помимо известной нам команды MOV и команд для работы со стеком, к командам перемещения данных относятся следующие

1. XCHG меняет между собой значения двух своих операндов
2. CMOVcc перемещение с условием (аналогично Jcc.)
3. LEA - load effective address
4. XLATB
5. CBW — преобразование байта в слово,
6. CWD — преобразование слова в двойное слово
7. CWDE — преобразование слова в двойное слово.
8. CDQ — преобразование двойного слова в учетверённое слово.

XCHG

Команда XCHG меняет местами значения своих операндов. Она может использоваться в трех случаях:
XCHG регистр, регистр
XCHG память, регистр
XCHG регистр, память
но не может использоваться для одновременного доступа к двум операндам, находящимся в памяти.

```
1 xchg al, bl
2 xchg bl, [x]
```

Также важно помнить о том, что операнды команды XCHG обязательно должны быть одного размера. Так, например, строка *xchg al, bx* вызовет ошибку, т.к. длина AL = 8 битам, а BX = 16 битам.

CMOVcc

Команда CMOVcc (перемещение с условием) имеет те же модификации[1], что и команды условного перехода. Она выполняется только при определенном состоянии флагов.

1. CMOVA (Если CF=0 и ZF=0)
2. CMOVAE (Если CF=0)
3. CMOVB (Если CF=1)
4. CMOVBE (Если CF=1 и ZF=1)
5. CMOVC (Если CF=1)
6. CMOVE (Если ZF=1)
7. CMOVG (Если ZF=0 и SF=OF)
8. CMOVGE (Если SF=OF)

[1] Полный список можно найти на
https://www.jaist.ac.jp/iscenter-new/mpc/altix/altixdata/opt/intel/vtune/doc/users_guide/mergedProjects/analyzer_ec/mergedProjects/reference_olh/mergedProjects/instructions/instruct32_hh/vc35.htm

LEA

LEA регистр, память

Команда LEA вычисляет адрес второго операнда и заносит в первый операнд, который должен обязательно быть регистром. Для примера можно загрузить в ECX адрес третьей буквы строки "Hello World". Обычно, команда LEA встречается в более сложных вычислениях с адресами. Например, при работе с массивами.

```
1 mov eax, 4
2 mov ebx, 1
3 lea ecx, [hello+2]
4 mov edx, 4
5 int 0x80
```

Приведение к другому размеру

CBW — преобразование байта в слово. Команда копирует старший разряд регистра AL во все разряды регистра AH.
CWDE — преобразование слова в двойное слово. Команда копирует старший бит регистра AX в разряды старших двух байт регистра EAX.
CWD — преобразование слова в двойное слово. Команда копирует значение старшего бита регистра AX во все биты регистра DX.
CDQ — преобразование двойного слова в учетверённое слово. Команда копирует знаковый бит регистра EAX во все биты регистра EDX.

```
1 mov al, 0b10000000
2 cbw
3 ; После CBW регистр AX = 0b1111111110000000
```

```
1 mov ah, 0b10010100
2 cwd
3 ; После CWD регистр DX = 0b1111111111111111
```

XLATB

Команда XLATB - (transLATe Byte from table) выполняет преобразование байта.
Используется для замены байта в регистре AL байтом из последовательности (таблицы) байтов в памяти.
Адрес последовательности байтов, из которой будет осуществляться выборка байта для замены в регистре AL, должен быть предварительно загружен в пару DS:EBX. (для Linux - просто EBX)
Команду XLATB можно использовать для выполнения перекодировок символов. Для формирования адреса таблицы в регистрах DS:EBX можно использовать команду LEA или оператор ассемблера OFFSET в команде MOV.

```
1 mov     ebx, subst1 ; Адрес условной подстановки
2 mov     al, [x] ; Исходный байт в AL
3 sub     al, '0' ; '0' == 0x30 == 48
4 xlatb
```