В синтаксисе Intel (например, компиляторы TASM, MASM, NASM) многие команды имеют вид: ***ком*** *назначение, источник*
т.е. результат сохраняется в **первый** операнд.
При этом размеры операндов должны быть **строго** равны (за исключением некоторых специальных команд).
Недопустимо, чтобы **оба** операнда были ячейками **памяти**.

Арифметические:

Составил: ***Третьяков Андрей*** *(МФТИ)*

Действие	Команда	Аналог в C	Примечание
Пересылка (*aka* Копирование, Присвоение)	**mov *куда*, *откуда***	**куда = откуда**	Лишь один операнд может быть сегментным регистром, при этом 2-й должен быть регистром общего назначения либо памятью. Значение второго операнда и регистра флагов не изменяется. *Примеры:* mov eax, ebx mov eax, [b] mov al, bl mov rax, rbx mov eax, 5 mov ax, bx mov [a], ebx mov dword [a], 5
Загрузка адреса	**lea оп1, оп2**	**оп1 = &оп2**	Копирование в **оп1** не значения, а ***адреса*** **оп2**. Пример: **lea eax, [ebx+ecx*8+10]** <=> **eax = ebx+ecx*8+10**
Обмен	**xchg оп1, оп2**	**swap(оп1, оп2)**	
Сложение	**add оп1, оп2**	**оп1 += оп2**	Меняет флаги
Вычитание	**sub оп1, оп2**	**оп1 –= оп2**	Меняет флаги
Сравнение	**cmp оп1, оп2**	реализация условий (**==**, **!=**, **>**, **<** и т.д.)	То же самое, что и **sub**, только первый операнд (**оп1**) **не изменяется**. Только выставляются флаги.
Инкремент	**inc оп**	**оп++**	Меняет флаги, кроме CF
Декремент	**dec оп**	**оп--**	Меняет флаги, кроме CF
Смена знака	**neg оп**	**оп = –оп**	Меняет флаги
Умножение (беззнаковое)	**mul bl** **mul bx** **mul ebx**	**ax = al * bl** **dx:ax = ax * bx** **edx:eax = eax * ebx**	**Первый операнд фиксирован** (регистр **al**, **ax** или **eax**, в зависимости от размера второго операнда). **Второй операнд** не может быть константой. Может менять флаги. Куда помещается **результат** - тоже зависит от размера 2-го операнда.
Умножение (со знаком)	**imul bl** **imul bx** **imul ebx** **imul eax, ebx** **imul eax, ebx, 10**	**ax = al * bl** **dx:ax = ax * bx** **edx:eax = eax * ebx** **eax *= ebx** **eax = ebx * 10**	Для формы с 1 операндом всё то же самое, что и для **mul**. Для формы с 3 операндами - последний оп. должен быть константой. Может менять флаги.
Деление (целочисленное, беззнаковое)	**div bl** **div bx** **div ebx**	**al = ax / bl** **ax = dx:ax / bx** **eax = edx:eax / ebx**	Делимое **фиксировано** (см. в соответствии с **mul**). Может менять флаги. **ah = ax % bl** **div** частное помещает в **al**, а остаток в **ah** **dx = dx:ax % bx** младший байт делимого - в **ax**, старший в **dx** **edx = edx:eax / ebx** аналогично, только с 32-битными регистрами
Деление со знаком	**idiv оп2**	Всё то же самое, что и для **div**. Остаток имеет знак делимого.	

Поразрядные и другие:

Действие	Команда	Аналог в C	Примечание
Поразрядное И	**and оп1, оп2**	**оп1 &= оп2**	Меняет флаги (CF = OF = 0)
	test оп1, оп2	**оп1 & оп2**	То же самое, что и **and**, только первый операнд (**оп1**) **не изменяется**. Только выставляются флаги. Используется для анализа бит операндов.
Поразрядное ИЛИ	**or оп1, оп2**	**оп1 |= оп2**	Меняет флаги (CF = OF = 0)
Поразрядный XOR	**xor оп1, оп2**	**оп1 ^= оп2**	Меняет флаги (CF = OF = 0)
Инвертирование	**not оп**	**оп = !оп**	Не меняет флаги
Сдвиг влево	**shl оп1, оп2**	**оп1 <<= оп2**	**оп1 *= pow(2, оп2)** (unsigned **оп1**) Второй операнд может быть только константой либо регистром **cl** (размеры операндов не обязаны быть равны). Меняет флаги CF и OF. Выдвигаемый бит попадает в CF.
	sal оп1, оп2		
Сдвиг вправо	**shr оп1, оп2**	**оп1 >>= оп2**	**оп1 /= pow(2, оп2)** (unsigned **оп1**) Всё то же самое, что и для **shl**.
Ариф.сдвиг вправо	**sar оп1, оп2**	**оп1 >>= оп2**	**оп1 /= pow(2, оп2)** (signed **оп1**) Всё то же самое, что и для **shl**. Слева вдвигается значение старшего бита операнда, а не 0.
Циклич.сдвиг влево	**rol оп1, оп2**		Всё то же самое, что и для **shl**
Циклич.сдвиг вправо	**ror оп1, оп2**		Всё то же самое, что и для **shl**
Циклический сдвиг влево через CF	**rcl оп1, оп2**		Выдвигаемый бит попадает в CF, а справа вдвигается старое значение CF. Остальное аналогично **rol**.
Циклический сдвиг вправо через CF	**rcr оп1, оп2**		Выдвигаемый бит попадает в CF, а слева вдвигается старое значение CF. Остальное аналогично **ror**.
Расширение **al** до **ax**	**cbw**	Используются, в осн., для подготовки делимого перед командой **div**/**idiv**	Копирует старший бит регистра **al** во все биты регистра **ah**
Расширение **ax** до **dx:ax**	**cwd**		Копирует старший бит регистра **ax** во все биты регистра **dx**
Расширение **ax** до **eax**	**cwde**		Копирует старший бит регистра **ax** во все биты старшей половины **eax**
Расширение **eax** до **edx:eax**	**cdq**		Копирует старший бит регистра **eax** во все биты регистра **edx**
Пересылка со знак. расширением	**movsx оп1, оп2**	**оп1 = оп2**	Размер **оп1** должен быть больше размера **оп2**. Копирует старший бит **оп2** во все биты старшей половины **оп1**.
Пересылка с беззнак. расширением	**movzx оп1, оп2**	**оп1 = оп2**	Размер **оп1** должен быть больше размера **оп2**. Старшая половина **оп1** заполняется нулями.
Пустая команда	**nop**		Не делает ничего, только тратит место в программе и процессорное время

Переходы и стек: *Составил:* ***Третьяков Андрей*** *(МФТИ)*

Действие	**Команда**	**Аналог в C**	**Примечание**
Безусловный переход	**jmp метка**	**goto метка;**	Используется при организации циклов и сложных условий
Условный переход	**jcc метка**	if (!условие) {…} метка:	Реализация циклов и условий. Условие **cc** берётся из регистра флагов.
	jcxz метка	if (cx != 0) {...} метка:	В качестве метки может использоваться относительный адрес, регистр
	jecxz метка	if (ecx != 0) {...} метка:	или ячейка памяти, содержащие абсолютный адрес перехода.
Установка байта по усл.	**setcc оп**		То же самое, что **jcc**, только вместо перехода изменяет **оп** на 01 или 00
Простейший цикл	**loop метка**	**метка: do {… } while (--ecx);**	Автоматически изменяет регистр ecx. Не меняет флагов.
	loopz метка		То же самое, что и **loop**, только ещё учитывается флаг ZF
	loope метка		Абсолютно идентично **loopz**
	loopnz метка		Аналогично, только используется обратное значение ZF
	loopne метка		Абсолютно идентично **loopnz**
Положить в стек	**push оп**		
Сохранить *все* регистры в стеке	**pusha**		Регистры кладутся в стек в следующем порядке: ax,cx,dx,bx,sp,bp,si,di
	pushad		eax,ecx,edx,ebx,esp,ebp,esi,edi
Сохранить в стеке регистр флагов	**pushf**		Сохраняется регистр Flags (16 бит)
	pushfd		Сохраняется регистр Eflags (32 бит)
Достать из стека	**pop оп**		
Восстановить *все* регистры из стека	**popa**		Регистры забираются из стека в порядке, обратном **pusha**
	popad		Регистры забираются из стека в порядке, обратном **pushad**
Восстановить из стека регистр флагов	**popf**		Восстанавливается регистр Flags (16 бит)
	popfd		Восстанавливается регистр Eflags (32 бит). Флаги VM и RF не изменяются.
Вызов процедуры	**call метка**	**func();**	В стек сохраняется адрес команды, следующей за **call**.
Пролог функции	**enter оп1, 0**		**push ebp; mov ebp, esp; sub esp, оп1**
Эпилог функции	**leave**		**mov esp, ebp; pop ebp**
Возврат из процедуры	**ret**		Восстанавливает из стека содержимое регистра eip.
	ret оп		То же самое, только ещё и очищает из стека **оп** байт (дополнительно к адресу возврата).
Вызов прерывания	**int оп**		Программный вызов прерывания с номером **оп**. Управление передаётся на обработчик прерывания (функцию), адрес которого хранится в ячейке таблицы прерываний под индексом **оп**.

Операции с битами:

Действие	**Команда**	**Примечание**
Проверка бита	**bt оп, индекс**	Извлекает из **оп** бит с **индексом** и устанавливает его в флаг CF. **оп** и **индекс** не изменяются.
Проверка бита с инверси	**btc оп, индекс**	То же, что и **bt**, только в операнде **оп** указанный бит инвертируется. Старое значение бита - в CF.
Проверка бита со сбросо	**btr оп, индекс**	То же, что и **btc**, только указанный бит обнуляется
Проверка бита с установ	**bts оп, индекс**	То же, что и **btc**, только указанный бит устанавливается в 1
Инвертирование CF	**cmc**	Флаг переноса (выхода за пределы разрядной сетки)
Установка CF в 1	**stc**	Флаг переноса (выхода за пределы разрядной сетки) устанавливается в 1
Установка DF в 1	**std**	Флаг направления (в цепочечных командах; регистры esi и edi будут декрементироваться)
Установка IF в 1	**sti**	Флаг прерываний (маскируемых; прерывания разрешены)
Сброс CF	**clc**	Флаг переноса (выхода за пределы разрядной сетки) устанавливается в 0
Сброс DF	**cld**	Флаг направления (в цепочечных командах; регистры esi и edi будут инкрементироваться)
Сброс IF	**cli**	Флаг прерываний (маскируемых; прерывания запрещены)
Копирование Flags в ah	**lahf**	Копирует только младший байт регистра Flags в ah (т.е. CF, PF, AF, ZF, SF)
Копирование ah в Flags	**sahf**	Копирует ah в младший байт регистра Flags (т.е. CF, PF, AF, ZF, SF), остальные байты не измен.
Bit Scan Forward	**bsf оп1, оп2**	В **оп1** записывается индекс самого младшего бита операнда **оп2**, установленного в 1. Меняет флаг ZF.
Bit Scan Reverse	**bsr оп1, оп2**	В **оп1** записывается индекс самого старшего бита операнда **оп2**, установленного в 1. Меняет флаг ZF.