**1 РЕГИСТРЫ ОБЩЕГО НАЗНАЧЕНИЯ (32/ 16/ 8)**

* **EAX/ AX/ *AH/AL*** (*accumulator register*) – аккумулятор; переменная: x = [x]
* **EBX/ BX/ *BH/BL*** (*base register*) – регистр базы; массив x[4], x[ebx], [x + 4], [x][4], [x + ebx]
* **ECX/ CX/ *CH/CL*** (*counter register*) – счётчик; адрес ячейки в регистре: [ebx] или [ebx+4\*ecx]
* **EDX/ DX/ *DH/DL*** (*data register*) – регистр данных; **DUP** – констр. повторения (массив). k db 10 dup (0,1)
* **ESI/ SI** (*source index register*) – индекс источника; ? – неинициализи. **TYPE** – размер переменной
* **EDI/ DI** (*destination index register*) – индекс приёмника (получателя); db – 1 byte, dw – 2 byte
* **ESP/ SP** (*stack pointer register*) – регистр указателя стека; dd – 4byte, dq – 8byte, dt – 10byte
* **EBP/ BP** (*base pointer register*) – регистр указателя базы кадра стека.

**2 УКАЗАТЕЛЬ КОМАНД**

* **EIP** (*указатель команд*) содержит смещение следующей подлежащей выполнению команды.

**3 РЕГИСТР ФЛАГОВ**

**ФЛАГИ СОСТОЯНИЯ (**отражают результат выполнения **ADD, SUB, MUL, DIV)**

* **CF** (*флаг переноса*) устанавливается при переносе из старшего значащего бита/заёме в старший

значащий бит и показывает наличие переполнения в беззнаковой целочисленной арифметике. Также используется в длинной арифметике сложения с переносом (**ADC**) и вычитания с заёмом (**SBB**). Изменяется напрямую с помощью инструкций **STC**, **CLC**, **CMC**.

* **PF** (*флаг чётности*) устанавливается, если младший значащий байт результата содержит чётное число единичных битов. Инструкции сравнения (**FCOM**, **FCOMP** и т. п.)
* **AF** (*вспомогательный флаг переноса*) устанавливается при переносе из бита 3-го результата/заёме в 3-ий бит результата. Этот флаг ориентирован на использование в двоично-десятичной арифметике.
* **ZF** (*флаг нуля*) устанавливается, если результат равен нулю.
* **SF** (*флаг знака*) равен значению старшего значащего бита результата, который является знаковым битом в знаковой арифметике (1 – отрицательное число).
* **OF** (*флаг переполнения*) устанавливается, если целочисленный результат слишком длинный для размещения в целевом операнде (регистре или ячейке памяти). Наличие переполнения в знаковой целочисленной арифметике.

**УПРАВЛЯЮЩИЙ ФЛАГ**

* **DF** (*флаг направления*) управляет строковыми инструкциями (**MOVS, CMPS, SCAS, LODS и STOS**) – установка флага заставляет уменьшать адреса (обрабатывать строки от старших адресов к младшим), обнуление заставляет увеличивать адреса. Инструкции **STD** и **CLD** соответственно устанавливают и сбрасывают флаг DF.

**СИСТЕМНЫЕ ФЛАГИ И ПОЛЕ IOPL** (управляют операционной средой, нет в прикладных прогах)

* **IF** (*разрешения прерываний*) – обнуление запрещает отвечать на маскируемые запросы на прерывание.
* **TF** (*трассировки*) – установка этого флага разрешает пошаговый режим отладки.
* Поле **IOPL** показывает уровень приоритета ввода-вывода исполняемой программы или задачи: чтобы программа или задача могла выполнять инструкции ввода-вывода или менять флаг IF, её текущий уровень приоритета (**CPL**) должен быть ≤ **IOPL**.
* **NT** (*вложенности задач*) – устанавливается, когда текущая задача «вложена» в другую, прерванную задачу. **NT** проверяется инструкцией **IRET** для определения типа возврата: меж/внутри – задачного.
* **RF** (*флаг возобновления*) используется для маскирования ошибок отладки.
* **VM** – установка флага в защищённом режиме вызывает переключение в режим виртуального 8086.
* **AC** (*проверки выравнивания*) – включает контроль выравнивания операндов при обращениях к памяти.
* **VIF** – виртуальная копия флага IF; используется совместно с флагом VIP.
* **VIP** – устанавливается для указания наличия отложенного прерывания.
* **ID** – возможность программно изменить этот флаг в регистре флагов указывает на поддержку инструкции **CPUID**.

**4 СЕГМЕНТНЫЕ РЕГИСТРЫ**

16-битные регистры могли адресовать только 64 Кб оперативной памяти, что недостаточно для программы. Поэтому память программе выделялась в виде сегментов, с размером 64 Кб. При 32-битной архитектуре необходимость в этом отпала, и про сегментные регистры можно забыть. **CS, DS, SS, ES, FS и GS**. Адреса перехода сегментировались по **CS**, обращения к данным сегментировались по **DS**, а обращения к стеку – **SS**. 32-битные регистры позволяют адресовать 4 Гб памяти, что уже достаточно для любой программы.

**1. MOV** – команда **пересылки.** Значение второго операнда записывается в первый операнд. Операнды должны иметь одинаковый размер. Команда не меняет флаги.

**mov eax, ebx; mov eax, 0ffffh; mov x, 0; xor** **eax,** **eax**; Тут быстрое обнуление **eax**.

**2. MOVSX** <операнд1>, <операнд2> - знаковое расширение, старшие биты со знаковым битом. оп1 > оп2.

**3. MOVZX** <операнд1>, <операнд2> - беззнаковое расширение – старшие биты нулём. оп1 > оп2.

**4. XCHG –** команда **перестановки** двух величин. Каждый из операндов может быть регистром или ячейкой памяти. Переставить содержимое двух регистров можно, а двух ячеек памяти – нет. Флаги не меняет.

**xchg** **<операнд1>, <операнд2>**

**5. PTR** – оператор **указания** типа. ~~mov [ebx], 0;~~ ERR-0 может иметь любой размер. mov **byte ptr [ebx], 0;** 1 байт

**6. ADD** – команда **сложения**. Складывает операнды и результат на место 1-го операнда. Меняет флаги AF, CF, OF, PF, SF и ZF.

**add** **eax, b; add esi, eax; add eax, x[ebx] //** массив**; add ebx, type x;**

**7. SUB – вычитает** из первого операнда второй, результат на место 1-го операнда. Меняет флаги AF, CF, OF, PF, SF, ZF.

**sub eax, ebx; sub esp, 104h; sub esp, 16;** // резерв 16 байт.

**8. INC** и **DEC** – команды **инкремента** и **декремента**. Меняют флаги AF, OF, PF, SF и ZF. Меньше места.

**inc eax; dec ecx;**

**9. NEG** – **изменение** знака. Команда меняет флаги AF, CF, OF, PF, SF и ZF.

**mov bl, -128;** -> **neg bl;** //BL = -128, OF = 1

**10. MUL** – беззнаковое **умножение**. Операнд – первый сомножитель. Второй сомножитель явно не указан. Если операнд MUL имеет размер 1 байт, то 2-ой сомножитель берётся из регистра AL, результат в регистр AX. Если операнд MUL имеет размер 2 байта, то 2-ой сомножитель берётся из регистра AX, результат в регистровую пару DX:AX. Если операнд MUL имеет размер 4 байта, то 2-ой сомножитель берётся из регистра EAX, а результат помещается в регистровую пару EDX:EAX. Меняет флаги CF и OF.

**11. IMUL** – знаковое **умножение** имеет несколько вариантов. Первый соответствует команде MUL. Второй в IMUL указывается регистр, содержащий один из сомножителей. В этот регистр запишется результат. Третий в IMUL указывается и результат, и оба сомножителя. Результат только в регистр (1 операнд), а второй сомножит. только операнд. Первый - регистр или ячейка памяти. Четвёртый в IMUL указываются оба сомножителя. Первый – регистр, второй – регистр или ячейка памяти. Результат в первым операнд. Флаги как MUL**.**

**imul ebx; imul ebx, 6; imul eax, ebx, 100000; imul eax, x;**

**12. DIV** – **беззнаковое** **деление**. В командах только один операнд – делитель, либо регистр, либо ячейка памяти.

**13. IDIV** – **знаковое** **деление**. Если операнд размера 1 байт, то делимое из AX, частное в AL и остаток – в AH. Если операнд размера 2 байта, то делимое из DX:AX, частное в AX и остаток – в DX. Если делитель размера 4 байта, то делимое из регистровой пары EDX:EAX, частное в EAX и остаток – в EDX.

При делении, если делитель имеет размер 2 или 4 байта, то нужно устанавливать значение и регистра DX/EDX. Если же делитель имеет размер 1 байт, то можно просто записать делимое в регистр AX.

**14. CBW** – Знаковое расширение AL до AX

**15. CWD** – Знаковое расширение AX до DX:AX

**16. CWDE** – Знаковое расширение AX до EAX

**17. CDQ** – Знаковое расширение EAX до EDX:EAX

**18. JMP** – безусловный **переход**. **jmp** L **...** L: mov eax, x – прямой переход. **jmp** ebx – косвенный переход.

**19. CMP** – команда **сравнения**. Установка и сброс флагов. Эквивалент SUB без занесения результата.

**20.** **TEST** – логическое AND без записи результата. Меняет флаги ZF, SF, PF. Проверка равенства нулю.

**21.** **JXX** <метка> – условный переход. Таблица команд перехода (22) после команд сравнения и несравнения и ECX/CX

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| мнемо | название  Переход если: | условие перехода после CMP op1, op2 | значения флагов |  | мнемо | условие |  | мнемо | ECX/CX |
| **JE** | равно | op1 = op2 | ZF = 1 | **JZ** | ZF = 1 | **JCXZ** | переход  если  CX = 0 |
| **JNE** | не равно | op1 ≠ op2 | ZF = 0 | **JNZ** | ZF = 0 |
| **JL/JNGE** | меньше | op1 < op2 | SF ≠ OF | **JS** | SF = 1 |
| **JLE/JNG** | меньше или равно | op1 <= op2 | SF≠OF или ZF=1 | **JNS** | SF = 0 |
| **JG/JNLE** | больше | op1 > op2 | SF=OF и ZF=0 | **JC** | CF = 1 |
| **JGE/JNL** | больше или равно | op1 ≥ op2 | SF = OF | **JNC** | CF = 0 | **JECXZ** | переход  если  ECX = 0 |
| **JB/JNAE** | ниже | op1 < op2 | CF = 0 | **JO** | OF = 1 |
| **JBE/JNA** | ниже или равно | op1 ≤ op2 | CF=1 или ZF=1 | **JNO** | OF = 0 |
| **JA/JNBE** | выше | op1 > op2 | CF=0 и ZF=0 | **JP** | PF = 1 |
| **JAE/JNB** | выше или равно | op1 ≥ op2 | CF = 0 | **JNP** | PF = 0 |

**22. LOOP –** цикл с постусловием. Вычитает из счетчика ECX - 1. Если ECX > 0 продолжить цикл.

**23. LOOPE / LOOPZ** – цикл с постусловием. Досрочный выход если ZF = 0.

**24. LOOPNE / LOOPNZ** – цикл с постусловием. Досрочный выход если ZF = 1.

**25.** **LEA** – **загрузка** **адреса** переменных в памяти или параметров или начало массива в регистр эффективного адреса – адрес переменной. Флаги не меняет.

x dd 100 dup (0); **lea** **ebx**, **x.** Помещает только адрес, не получает доступа к содержимому x.

**26.** **SHL** – Логический **сдвиг влево**. Последний ушедший бит сохраняется во флаге CF. **УМНОЖЕНИЕ**

**27.** **SHR** – Логический **сдвиг вправо**. Последний ушедший бит сохраняется во флаге CF. **ДЕЛЕНИЕ**

**28.** **SAL** – Арифметический **сдвиг влево**. Последний ушедший бит сохраняется во флаге CF. **УМНОЖЕНИЕ**

**29.** **SAR** – Арифметический **сдвиг вправо**. Последний ушедший бит сохраняется во флаге CF. **ДЕЛЕНИЕ**

**30.** **ROL** – Циклический **сдвиг влево**. Последний ушедший бит сохраняется во флаге CF.

**31.** **ROR** – Циклический **сдвиг вправо**. Последний ушедший бит сохраняется во флаге CF.

**32**. **SHLD** – Расширенный сдвиг влево. «Освободившиеся» биты операнда1 заполняются старшими битами операнда2.

**33**. **SHRD** – Расширенный сдвиг **вправо**. «Освободившиеся» биты операнда1 заполняются младшими битами операнда2.

**34.** **PUSH** – **кладёт в стек** параметр для процедуры. Идет перед вызовом функции. Стек используется для передачи параметров и хранения локальных данных процедур. Операнд кладётся на вершину стека, а значение ESP уменьшается на размер операнда.

**35.** **POP** – **взять** данные **из стека**.

**36.** **PUSHA** – сохраняет в стеке содержимое регистров AX, CX, DX, BX, SP, BP, SI, DI.

**37.** **PUSHAD** – сохраняет в стеке содержимое регистров EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI.

**38.** **POPA POPAD** – восстанавливают из стека значения (E)DI, (E)SI, (E)BP, (E)SP, (E)BX, (E)DX, (E)CX, (E)AX.

**39.** **PUSHF** **/** **PUSHFD** – сохраняют в стеке младшие 16 бит / все 32 бита регистра флагов.

**40.** **POPF / POPFD** – восстанавливают в стеке младшие 16 бит / все 32 бита регистра флагов.

**41.** **CALL** – записывает адрес следующей за ней команды встек и переходит на первую команду указанной процедуры.

**42**. **RET** – считывает из вершины стека адрес и выполняет переход по нему. Возврат из процедуры.