

1. Регистры общего назначения.

Регистры общего назначения процессора (РОН) 8086. Регистры общего назначения

- группа регистров, доступная для чтения/записи основными командами.

Предназначены для временного хранения данных, записи параметров машинных команд, арифметической обработки и т.д. Существует всего 4 РОН: AX, BX, CX, DX.

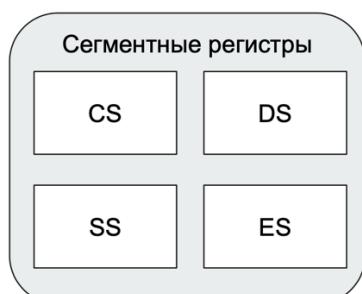
Каждый содержит в себе 16 бит и делится на 2 части по 8 бит - старшую (high, H) и младшую (low, L). Обращаться можно как к регистру целиком, так и к его половинам по отдельности.

AX		аккумулятор - умножение, деление, обмен с устройствами ввода/вывода (команды ввода и вывода);
AH	AL	
BX		базовый регистр в вычислениях адреса, часто указывает на начальный адрес (называемый базой) структуры в памяти;
BH	BL	
CX		счетчик циклов, определяет количество повторов некоторой операции;
CH	CL	
DX		определение адреса ввода/вывода, так же может содержать данные, передаваемые для обработки в подпрограммы.
DH	DL	

2. Сегментные регистры. Адресация в реальном режиме. Понятие сегментной части адреса и смещения.

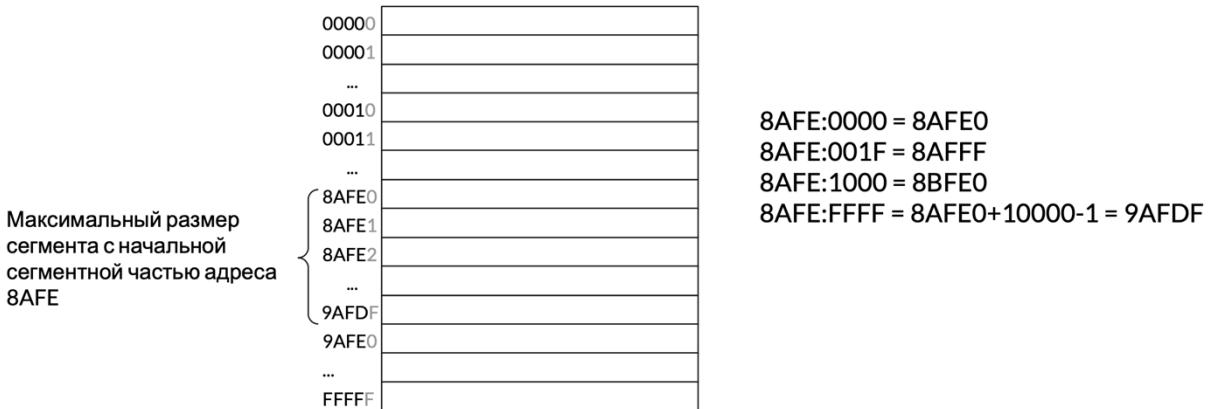
Реальный режим работы - режим совместимости современных процессоров с 8086. Доступен 1 Мб памяти (2²⁰ байт), то есть разрядность шины адреса - 20 разрядов. Физический адрес получается сложением адреса начала сегмента (на основе сегментного регистра) и смещения.

Сегментный регистр хранит в себе старшие 16 разрядов (из 20) адреса начала сегмента. 4 младших разряда в адресе начала сегмента всегда нулевые. Говорят, что сегментный регистр содержит в себе номер параграфа начала сегмента. Вычисление физического адреса выполняется процессором аппаратно.



- Сегмент кода - регистр CS. Командой MOV изменить невозможно, меняется автоматически по мере выполнения команд.
- Сегмент данных. Основной регистр - DS, при необходимости дополнительных сегментов данных задействуются ES, FS, GS.

Сегментная модель памяти 8086



Сегмент стека - регистр SS

3. Регистры работы со стеком.

Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний

- SP - указатель на вершину стека
- BP – base pointer: Используется в подпрограмме для сохранения "начального" значения SP, Адресация параметров, Адресация локальных переменных

В x86 стек "растёт вниз", в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента

- PUSH <источник> - поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- POP <приёмник> - считать данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- PUSHA - поместить в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- POPA - загрузить регистры из стека (SP игнорируется)

CALL - вызов процедуры, RET - возврат из процедуры

CALL <операнд>

- Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Передаёт управление на значение аргумента.

RET/RETN/RETF <число>

- Загружает из стека адрес возврата, увеличивает SP
- Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

4. Структура программы. Сегменты.

Как и на других языках программирования, программа на ассемблере может состоять из нескольких файлов - модулей. При компиляции (трансляции) каждый модуль превращается в объектный файл, далее при компоновке объектные файлы соединяются в единый исполняемый модуль. Модули обычно состоят из описания сегментов будущей программы с помощью директивы SEGMENT.

Пример:

```
имя SEGMENT [READONLY] выравнивание тип разряд 'класс'  
...  
имя ENDS
```

Параметры:

- Выравнивание - расположение начала сегмента с адреса, кратного какому-либо значению. Варианты: BYTE, WORD (2 байта), DWORD (4 байта), PARA (16 байт, по умолчанию), PAGE (256 байт).
- Тип: PUBLIC (сегменты с одним именем объединяются в один); STACK (для стека); COMMON (сегменты будут "наложены" друг на друга по одним и тем же адресам памяти); AT <начало> - расположение по фиксированному физическому адресу, параметр - сегментная часть этого адреса; PRIVATE - вариант по умолчанию.
- Класс - метка, позволяющая объединить сегменты (расположить в памяти друг за другом).

Структура программы на ассемблере

(Зубков С. В. Assembler для DOS, Windows, Unix, глава 3)

- Модули (файлы исходного кода)
 - Сегменты (описание блоков памяти)
 - команды процессора;
 - инструкции описания структур данных, выделения памяти для переменных и констант;
 - макроопределения.

Полный формат строки:

метка команда/директива операнды ; комментарий

5. Прерывание 21h. Примеры ввода-вывода.

int <номер> - вызов (генерация **программного** прерывания)

21h - прерывание DOS, предоставляет прикладным программам около 70 различных функций (ввод, вывод, работа с файлами, завершение программы и т.д.)

Номер функции прерыванию 21h передаётся через регистр AH. Параметры для каждой функции передаются собственным способом, он описан в документации. Там же описан способ возврата результата из функции в программу.

- Аналог системного вызова в современных ОС
- Используется наподобие вызова подпрограммы
- Номер функции передаётся через AH

Прерывание DOS - вывод на экран в текстовом режиме

Функция	Назначение	Вход	Выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом \$	-

Прерывание DOS - ввод с клавиатуры

Функция	Назначение	Вход	Выход
01	Считать символ из stdin с эхом	-	AL - ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL = FF	AL - ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL - ASCII-код символа
08	Считать символ без эха	-	AL - ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры	-	AL=0, если клавиша не была нажата, и FF, если была
0Ch	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	
4Ch	Завершить программу	AL = код завершения	-

4Ch - "особая" функция. При её вызове управление в программу не вернётся, память, занимаемая программой, будет очищена, и управление вернётся вызвавшей программе.

6. Стек. Назначение, примеры использования.

- LIFO/FIFO (last in, first out) - последним пришёл, первым ушёл
- Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний
- SP - указатель на вершину стека
- В x86 стек "растёт вниз", в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента
- PUSH <источник> - поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- POP <приёмник> - считать данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- PUSHA - поместить в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- POPA - загрузить регистры из стека (SP игнорируется)

CALL - вызов процедуры, RET - возврат из процедуры

CALL <операнд>

- Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Передаёт управление на значение аргумента.

RET/RETN/RETF <число>

- Загружает из стека адрес возврата, увеличивает SP
- Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

7. Регистр флагов.

Регистр FLAGS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CF	-	PF	-	AF	-	ZF	SF	TF	IF	DF	OF	IOPL	NT	-	

- CF (carry flag) - флаг переноса
- PF (parity flag) - флаг чётности
- AF (auxiliary carry flag) - вспомогательный флаг переноса
- ZF (zero flag) - флаг нуля
- SF (sign flag) - флаг знака
- TF (trap flag) - флаг трассировки
- IF (interrupt enable flag) - флаг разрешения прерываний
- DF (direction flag) - флаг направления
- OF (overflow flag) - флаг переполнения
- IOPL (I/O privilege flag) - уровень приоритета ввода-вывода
- NT (nested task) - флаг вложенности задач

8. Команды условной и безусловной передачи управления.

Команда безусловной передачи управления JMP

JMP <операнд>

- Передаёт управление в другую точку программы (на другой адрес памяти), не сохраняя какой-либо информации для возврата.
- Операнд - непосредственный адрес (вставленный в машинный код), адрес в регистре или адрес в переменной.

Команды условных переходов J..

(Зубков С. В. Assembler для DOS, Windows, Unix, глава 2)

- Переход типа short или near
- Обычно используются в паре с CMP
- Термины “выше” и “ниже” - при сравнении беззнаковых чисел
- Термины “больше” и “меньше” - при сравнении чисел со знаком

Виды переходов (передачи управления)

Короткий (short) - в пределах адресов -128..+127 от текущего значения IP (1 байт).

Ближний (near) - в пределах того же сегмента (2 байта).

Дальний (far) - на произвольный адрес (4 байта).

Виды условных переходов (часть 1)

Команда	Описание	Состояние флагов для выполнения перехода
JO	Есть переполнение	OF = 1
JNO	Нет переполнение	OF = 0
JS	Есть знак	SF = 1
JNS	Нет знака	SF = 0
JE, JZ	Если равно/если ноль	ZF = 1
JNE, JNZ	Не равно/не ноль	ZF = 0
JP/JPE	Есть чётность / чётное	PF = 1
JNP/JPO	Нет чётности / нечётное	PF = 0
JCXZ	CX = 0	-

Виды условных переходов (часть 2)

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JB JNAE JC	Если ниже Если не выше и не равно Если перенос	CF = 1	нет
JNB JAE JNC	Если не ниже Если выше или равно Если нет переноса	CF = 0	нет
JBE JNA	Если ниже или равно Если не выше	CF = 1 или ZF = 1	нет
JA JNBE	Если выше Если не ниже и не равно	CF = 0 и ZF = 0	нет

9. Организация многомодульных программ.

Директивы глобальных объявлений

PUBLIC идентификатор

Описывает идентификатор, как доступный из других модулей.

EXTRN определение [, определение] .

Указывает, что идентификатор определен в другом модуле. Определение описывает идентификатор и имеет следующий формат:

имя : тип

"Имя" - это идентификатор, который определен в другом модуле. "Тип" должен соответствовать типу идентификатора, указанному при его определении, и может быть следующим: NEAR, FAR, PROC, BYTE, WORD, DWORD, DATAPTR, CODEPTR, FWORD, PWORD, QWORD, TBYTE, ABS или именем структуры.

10. Подпрограммы. Объявление, вызов.

CALL - вызов процедуры,
RET - возврат из процедуры

CALL <операнд>

- Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Передаёт управление на значение аргумента.

RET/RETN/RETF <число>

- Загружает из стека адрес возврата, увеличивает SP
- Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

Name proc <переход>

ret

Name endp

11. Арифметические команды.

Целочисленная арифметика (основные команды)

- ADD <приёмник>, <источник> - выполняет арифметическое сложение приёмника и источника. Сумма помещается в приёмник, источник не изменяется.
- SUB <приёмник>, <источник> - арифметическое вычитание источника из приёмника.
- MUL <источник> - беззнаковое умножение. Умножаются источник и AL/AH, в зависимости от размера источника. Результат помещается в AH либо DX:AX.
- DIV <источник> - целочисленное беззнаковое деление. Делится AL/AH на источник. Результат помещается в AL/AH, остаток - в AH/DX.
- INC <приёмник> - инкремент на 1
- DEC <приёмник> - декремент на 1

Двоичная арифметика. ADD, ADC, SUB, SBB

ADD, SUB не делают различий между знаковыми и беззнаковыми числами.

ADC <приёмник>, <источник> - сложение с переносом. Складывает приёмник, источник и флаг CF.

SBB <приёмник>, <источник> - вычитание с заемом. Вычитает из приёмника источник и дополнительно - флаг CF.

```
add ax, cx  
adc dx, bx
```

```
sub ax, cx  
sbb dx, bx
```

Арифметические флаги - CF, OF, SF, ZF, AF, PF



IMUL, MUL, IDIV, DIV

Умножение чисел со знаком:

IMUL <источник>

IMUL <приёмник>, <источник>

IMUL <приёмник>, <источник1>, <источник2>

Целочисленное деление со знаком:

IDIV <источник>

Результат округляется в сторону нуля, знак остатка совпадает со знаком делимого.

INC, DEC

INC <приёмник>

DEC <приёмник>

Увеличивает/уменьшает приёмник на 1.

В отличие от ADD, не изменяет CF.

OF, SF, ZF, AF, PF устанавливаются в соответствии с результатом.

NEG - изменение знака

NEG <приёмник>

Переводит число в дополнительный код.

12. Команды побитовых операций.

Побитовая арифметика (основные команды)

- AND <приёмник>, <источник> - побитовое “И”. AND al, 00001111b
- OR <приёмник>, <источник> - побитовое “ИЛИ”. OR al, 00001111b
- XOR <приёмник>, <источник> - побитовое исключающее “ИЛИ”. XOR AX, AX
- NOT <приёмник> - инверсия

Логический, арифметический, циклический сдвиг. **SAR, SAL, SHR, SHL, ROR, ROL, RCR, RCL**

- SAL тождественна SHL
- SHR зануляет старший бит, SAR - сохраняет (знак)
- ROR, ROL - циклический сдвиг вправо/влево
- RCR, RCL - циклический сдвиг через CF

Операции над битами и байтами **BT, BTR, BTS, BTC, BSF, BSR, SETcc**

- BT <база>, <смещение> - считать в CF значение бита из битовой строки
- BTS <база>, <смещение> - установить бит в 1
- BTR <база>, <смещение> - сбросить бит в 0
- BTC <база>, <смещение> - инвертировать бит
- BSF <приёмник>, <источник> - прямой поиск бита (от младшего разряда)
- BSR <приёмник>, <источник> - обратный поиск бита (от старшего разряда)
- SETcc <приёмник> - выставляет приёмник (1 байт) в 1 или 0 в зависимости от условия, аналогично Jcc

13. Команды работы со строками.

Строковые операции: копирование, сравнение, сканирование, чтение, запись

Строка-источник - DS:SI, строка-приёмник - ES:DI.

За один раз обрабатывается один байт (слово).

- MOVS/MOVSB/MOVSW <приёмник>, <источник> - копирование
- CMPS/CMPSB/CMPSW <приёмник>, <источник> - сравнение
- SCAS/SCASB/SCASW <приёмник> - сканирование (сравнение с AL/AX)
- LODS/LODSB/LODSW <источник> - чтение (в AL/AX)
- STOS/STOSB/STOSW <приёмник> - запись (из AL/AX)

Префиксы: REP/REPE/REPZ/REPNE/REPNZ

Прерывание DOS - ввод с клавиатуры

Функция	Назначение	Вход	Выход
01	Считать символ из stdin с эхом	-	AL - ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL = FF	AL - ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL - ASCII-код символа
08	Считать символ без эха	-	AL - ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры	-	AL=0, если клавиша не была нажата, и FF, если была
0Ch	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	

Прерывание DOS - вывод на экран в текстовом режиме

Функция	Назначение	Вход	Выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом \$	-

14. Прерывания. Обработка прерываний.

Прерывания

- Прерывание - особая ситуация, когда выполнение текущей программы приостанавливается и управление передаётся программе-обработчику возникшего прерывания.
- Виды прерываний:
 - аппаратные (асинхронные) - события от внешних устройств;
 - внутренние (синхронные) - события в самом процессоре, например, деление на ноль;
 - программные - вызванные командой int.

Внешние прерывания, в зависимости от возможности запрета, делятся на:

- **маскируемые** – прерывания, которые можно запрещать установкой соответствующего флага;
- **немаскируемые** (англ. Non-maskable interrupt, NMI) – обрабатываются всегда, независимо от запретов на другие прерывания

Таблица векторов прерываний в реальном режиме работы процессора

- Вектор прерывания – номер, который идентифицирует соответствующий обработчик прерываний. Векторы прерываний объединяются в таблицу векторов прерываний, содержащую адреса обработчиков прерываний.
- Располагается в самом начале памяти, начиная с адреса 0.
- Доступно 256 прерываний.
- Каждый вектор занимает 4 байта - полный адрес.
- Размер всей таблицы - 1 Кб.

Срабатывание прерывания

- Сохранение в текущий стек регистра флагов и полного адреса возврата (адреса следующей команды) - 6 байт
- Передача управления по адресу обработчика из таблицы векторов
- *Настройка стека?*
- *Повторная входимость (реентерабельность), необходимость запрета прерываний?*

IRET - возврат из прерывания

- Используется для выхода из обработчика прерывания
- Восстанавливает FLAGS, CS:IP
- При необходимости выставить значение флага обработчик меняет его значение непосредственно в стеке

Перехват прерывания

- Сохранение адреса старого обработчика
- Изменение вектора на "свой" адрес
- Вызов старого обработчика до/после отработки своего кода
- При деактивации - восстановление адреса старого обработчика

Установка обработчика прерывания в DOS

- int 21h
 - AH=35h, AL= номер прерывания - возвращает в ES:BX адрес обработчика (в BX 0000:[AL*4], а в ES - 0000:[AL*4+2].)
 - AH=25h, AL=номер прерывания, DS:DX - адрес обработчика

Некоторые прерывания

- 0 - деление на 0
- 1 - прерывание отладчика, вызывается после каждой команды при флаге TF
- 3 - "отладочное", int 3 занимает 1 байт
- 4 - переполнение при команде INTO (команда проверки переполнения)
- 5 - при невыполнении условия в команде BOUND (команда контроля индексов массива)
- 6 - недопустимая (несуществующая) инструкция
- 7 - отсутствует FPU
- 8 - таймер
- 9 - клавиатура
- 10h - прерывание BIOS

Резидентные программы

- Резидентная программа - та, которая остаётся в памяти после возврата управления DOS
- Завершение через функцию 31h прерывания 21h / прерывание 27h
- DOS не является многозадачной операционной системой
- Резиденты - частичная реализация многозадачности
- Резидентная программа должна быть составлена так, чтобы минимизировать используемую память

Резидентная программа для MS DOS представляет собой фрагмент кода, постоянно находящийся в оперативной памяти компьютера и вызываемый при возникновении определённых условий. Далее будет показано как написать резидентную программу на ассемблере, постоянно находящуюся в памяти и вызываемую при возникновении в системе прерываний. Сначала рассмотрим определения и основные типы прерываний для процессоров x86.

Прерывание для процессоров x86 представляет собой некоторое событие в системе, нуждающееся в определённой обработке. При возникновении прерывания, за исключением одного случая, выполнение текущей программы прерывается и происходит обработка прерывания. После обработки прерывания продолжается выполнение прерванной программы.

Завершение с сохранением в памяти

- int 27h
 - DX = адрес первого байта за резидентным участком программы (смещение от PSP)
- int 21h, ah=31h
 - AL - код завершения
 - DX - объём памяти, оставляемой резидентной, в параграфах

15. Работа с портами ввода-вывода.

Порты ввода-вывода

- Порты ввода-вывода - механизм взаимодействия программы, выполняемой процессором, с устройствами компьютера.
- IN - команда чтения данных из порта ввода
- OUT - команда записи в порт вывода
- Пример:

```
IN al, 61h  
OR al, 3  
OUT 61h, al
```