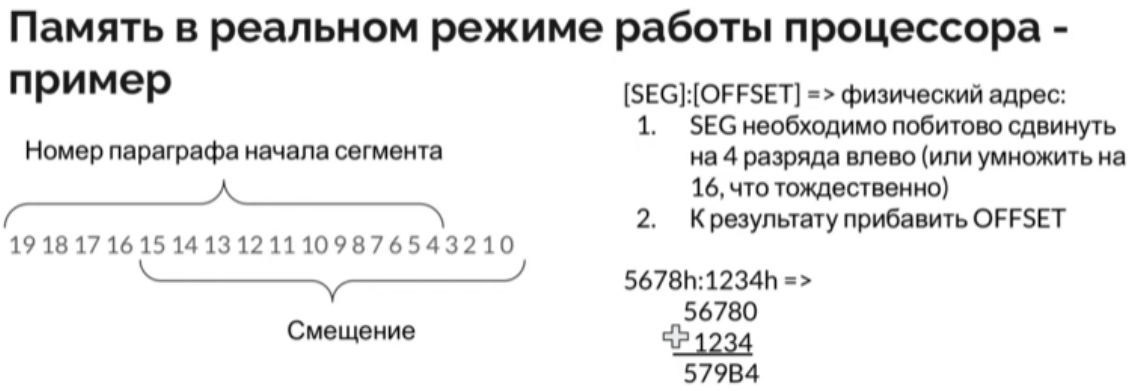
Машинно-зависимые языки программирования

Лекция 2.

Принцип формирования 20-разрядного физического адреса из 16-разрядного сегментного адреса (старшие разряды) и 16-разрядного смещения (младшие разряды):



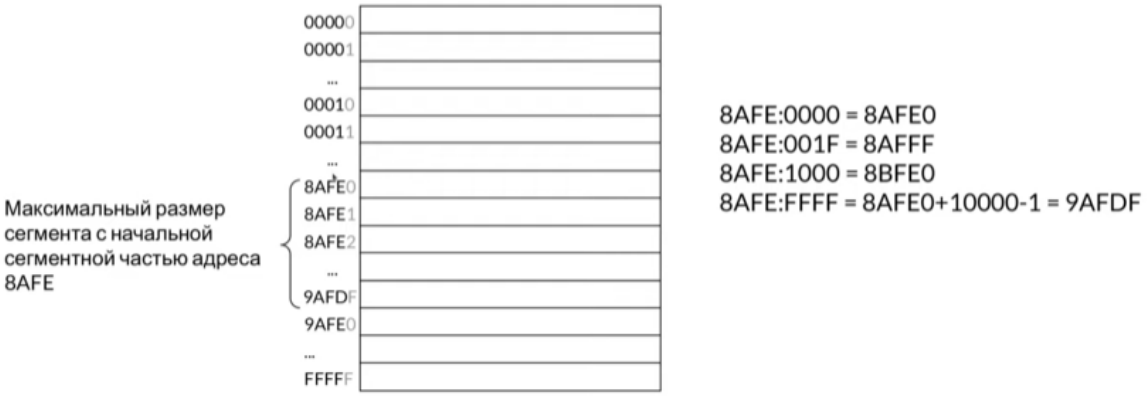
То есть берется адрес, сдвигается на 4 разряда, прибавляется смещение.

Пример:
Допустим, что у нас в компьютере ровно 1 Мб памяти. Тогда ячейки будут пронумерованы с 0 до FFFFF. Блоки памяти по 16 байт будут называться параграфами. Номер параграфа – старшие четыре цифры из 5 (или старшие 16 из 20 двоичных разрядов).

Сегмент может начинаться только с адреса, кратного параграфу, где последний 16-ричный разряд равен 0 (то есть с начала какого-то параграфа, например, 00000 или 00010 и тд). ?

К сегментной части адреса далее добавляется какое-то смещение, которое позволяет уже адресовать некоторый конкретный байт памяти и получить полный физический адрес ячейки в оперативной памяти.

Предположим, некоторый сегмент начинается с адреса 8AFE0 (соответственно, сегментная часть адреса здесь это 8AFE).



Максимальное смещение, которое нам доступно – это 2^16. Поэтому последней доступной ячейкой этого сегмента будет 8AFE0 + 2^16 - 1 = 9AFDF.

На практике размер сегмента зависит от данных, которые мы в него записываем. Может выделяться ровно 64 Кб, может меньше.

Сегменты

Программа может состоять из сегментов трех типов:

- Сегмент кода (регистр CS) // обязательный
- Сегмент данных (основной регистр – DS, для дополнительных сегментов – ES, FS, GS)
- Сегмент стека (регистр SS)

Команда организации цикла LOOP

LOOP метка – уменьшается регистр CX на 1 и выполняет переход на метку, если CX не равен 0. Метка не может быть дальше -128…127 байт от команды.

Структура программы на ассемблере

Основное, из чего состоит программа – это файлы с исходным кодом. Такие файлы в ассемблере называются модулями.

Основное, что находится в каждом модуле – это описание сегментов (то есть описание блоков памяти). Внутри сегментов находятся:

- Команды процессора
- Инструкции описания структур данных, выделения памяти для переменных и констант

- макроопределения

Полный формат строки (все части необязательны):
```
метка           команда/директива           операнды           ; комментарий
```

Метки

Записываются по-разному, в зависимости от того, где он применяются.

В коде

- Имя метки обязательно отделяется двоеточием
- Имя может состоять из латинских букв, цифр (не могут идти в начале), знаков подчеркивания и еще нескольких спец символов
- Обычно используются в командах передачи управления
- Пример:
```
mov cx, 5
label1:
        add ax, bx
        loop label1  ; цикл выполняется 5 раз
```

В данных

- Используются для объявления переменных или констант
- label – ключевое слово, для определения переменной
- Возможные типы:
    - BYTE – 1 байт
    - WORD – 2 байта
    - DWORD – 4 байта
    - FWORD – 6 байт
    - QWORD – 8 байт
    - TBYTE – 10 байт

      // И два доп типа для хранения меток в коде, адресов команд
    - NEAR – метка ближнего перехода
    - FAR – метка дальнего перехода
- Пример:
```
метка label тип
```

- Использование в макросах
- Пример:
```
метка EQU выражение   (EQU можно заменить на =)
```

Директивы

Директива – инструкция ассемблеру, влияющая на процесс компиляции и не являющаяся командой процессора. Обычно не оставляет следов в формируемом машинном коде.

Псевдокоманда – директива ассемблера, которая приводит к включению данных или кода в программу, нот не соответствующая никакой команде процессора.

Псевдокоманды определения данных указывают, что в соответствующем месте располагается переменная, резервируют под нее место заданного типа, заполняют значением и ставят в соответствие метку.

Виды: DB(1), DW(2), DD(4), DF(6), DQ(8), DT(10)

Примеры:

- a DB 1 – выделить ячейку размером 1 байт, присвоить ей имя a и положить туда значение 1
- float_number DD 3.5e7 – объявляем вещественную переменную размером 4 байта и помещаем в нее вещественное число 3.5e7
- text_string DB ‘Hello world!’ – если строка задается вот так в кавычках (или несколько чисел перечисляются через запятую), то будет выделен не один байт, а столько сколько символов в строке (перечисленных значений) по одному байту.

DUP – заполнение повторяющимися данными
метка DB 100 DUP какое-то значение – будет выделено 100 байт с указанным значением

? – неинициализированное значение
uninit DW 512 DUP (?)

Объявление сегмента программы
имя SEGMENT [READONLY] [выравнивание] [тип] [разрядность] [‘класс’]
…
имя ENDS

READONLY – если во время выполнения компилятор увидит, что идет попытка записи в этот сегмент, компиляция будет остановлена с ошибкой.

Выравнивание – обозначает с каких адресов будет начинаться сегмент:

- BYTE – сегмент может начаться с произвольного адреса
- WORD – сегмент всегда начинается с адреса, кратного 2
- DWORD – сегмент всегда начинается с адреса, кратного 4
- PARA – сегмент всегда начинается с начала параграфа (адреса, кратного 16) ! это значение по умолчанию
- PAGE – сегмент начинается с адреса, кратного 256

Тип:

- PUBLIC – сегменты с одним именем будут располагаться в памяти непосредственно друг за другом, независимо от того, в каком порядке они были объявлены в исходном коде

- STACK – сегмент будет использоваться под стек. Все сегменты в исходном коде с таким типом будут объединяться в один для увеличения размера стека.
- COMMON – сегменты также будут объединяться, но не друг за другом, а как бы накладываться (те начинаться с одного и того же адреса)
- AT – должен иметь определенный аргумент (номер параграфа начала сегмента), обозначает, что сегмент будет загружаться в память по некоторому фиксированному постоянному адресу (вне зависимости от других сегментов)
- PRIVATE – сегмент не объединяется с другими, существует сам по себе !это значение по умолчанию

Класс – любая метка, взятая в одинарные кавычки. Сегменты одного класса будут расположены в памяти друг за другом.

Директива .model (модель памяти)
.model модель, язык, модификатор

// Нужны для сокращения записи программ определенных типов

Модели:
- TINY – один сегмент на все (пример – COM программа)
- SMALL – код в одном сегменте, данные и стек – в другом
- COMPACT – допустимо несколько сегментов данных
- MEDIUM – код в нескольких сегментах, данные в одном
- LARGE, HUGE

Язык: C, PASCAL, BASIC, SYSCALL, STDCALL.
(этот параметр нужен для связывания с языками высокого уровня и вызова подпрограмм, то есть на случай, если мы код на ассемблере захотим подключить куда-нибудь)

Модификатор – способ подключения стека:
- NEARSTACK – ближний
- FARSTACK – дальний

Директива END
…
END [точка_входа]

Этой директивой должно заканчиваться описание любого модуля.

точка_входа – имя метки в сегменте кода, указывающей на команду, с которой начнется исполнение программы
Если в программе несколько модулей, то только один может содержать точку входа.
звучит как бред

Сегментный префикс. Директива ASSUME.

```
Data1 SEGMENT WORD 'DATA'
Var1  DW   0
Data1 ENDS

Data2 SEGMENT WORD 'DATA'
Var2  DW   0
Data2 ENDS

Code  SEGMENT WORD 'CODE'
      ASSUME  CS:Code
ProgramStart:
      mov   ax,Data1
      mov   ds,ax
      ASSUME  DS:Data1
      mov   ax,Data2
      mov   es,ax
      ASSUME  ES:Data2
      mov   ax,[Var2]

      .
      .
      .
Code  ENDS
END   ProgramStart
```

Если мы собираемся работать с переменными, то зная, что переменная находится в каком-либо из сегментов данных (с этим сегментом может быть связано целых 4 сегментных регистра – DS, ES, FS, GS) мы понимаем, что процессору надо явно давать понять с каким именно из сегментов мы собираемся работать.

Можно писать полную запись DS:Var1
Можно пользоваться упрощением через директиву ASSUME регистр : имя сегмента. Эта директива устанавливает значение сегментного регистра по умолчанию.

В программе слева нас интересует строчка *mov ax, [Var2]*. Когда компилятор до нее дойдет, он посмотрит из какого сегмента переменная Var2, найдет последний ASSUME, который связан с Data2 и неявно сам подставит правильный сегментный префикс.

Прочие директивы

- Задание набора допустимых команд: .8086, .186, .286, … .586, .686
- Управление программным счетчиком:
    - ORG значение – начиная с этой директивы отступ в сегменте будет идти с переданного значения (org 100h – сразу пропустить 256 байт)
    - EVEN – автоматически выровнять по четному адресу то, что идет после нее
    - ALIGN значение – явно задать какую-то кратность выравнивания

  // Выравнивание имеет смысл для ускорения работы программы в будущем, потому что процессор работает с памятью не по байтам, а по ячейкам, размер которых равен машинному слову.
- Глобальные объявления
    - public – объявление метки доступной из других модулей
    - comm
    - extrn – подключить метку из другого модуля
    - global
- Условное ассемблирование

  IF выражение

  …

  ELSE

  …

  ENDIF

Виды переходов для команды JMP:

- short (короткий) – -128..+127 байт

- near (ближний) – в том же сегменте (без изменения регистра CS)
- far (дальний) – в другой сегмент (с изменением значения в регистре CS)

Для короткого и ближнего переходов непосредственный операнд (константа) прибавляется к IP
Операнды – регистры и переменные заменяют старое значение в IP (CS:IP)

Почему существует три вида переходов? – для экономии места.

Индексные регистры SI и DI
SI (source index) – индекс источника
DI (destination index) – индекс приемника

Могут использоваться в большинстве команд, как регистры общего назначения. Применяются в специфических командах поточной обработки данных.

Способы адресации
Для процессора 8086 выделяют 6 возможных способов адресации ячеек:

- Регистровая адресация (mov ax, bx)
- Непосредственная адресация (mov ax, 2)
- Прямая адресация (mov ax, ds:0032)
- Косвенная адресация (mov ax, [bx])
  В 8086 допустимы BX, BP, SI, DI
- Адресация по базе со сдвигом (mov ax, [bx]+2; mov ax, 2[bx])
- Адресация по базе с индексированием (допустимы BX + SI, BX + DI, BP + SI, BP + DI)
    - mov ax, [bx + si + 2]
    - mov ax, [bx + 2][si]
    - mov ax, 2[bx][si]
    - mov ax, [bx][si] + 2
    - mov ax, [bx][si + 2]

Регистр FLAGS
– специальный регистр, к которому напрямую обращаться нельзя. Используется ни как единое число, а как отдельные биты (то есть все его 16 битов независимы друг от друга, каждый имеет свое собственное значение)

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

Черные – биты состояния
Синие – системные флаги
Зеленый – флаг управления DF
Жирные – флаги, которые можно непосредственно изменять

Состояния:
CF – флаг переноса. Возникает в случае перехода через разрядную сетку при работе с беззнаковыми числами.
PF – флаг четности (0 или 1). Указывает, какое количество бит получилось в младшем байте результата. Нужен для взаимодействия с какими-то устройствами, передачи данных, контроля корректности передачи данных.
AF – вспомогательный флаг переноса. Выставляется, если произошел перенос из младшего полубайта в старший полубайт (если биты нумеровать от 1 до 8, то из 4 в 5 бит). Нужен при работе с упакованными двоично-десятичными числами, когда у нас в байте не 256 различных значений, а всего 100.
ZF – флаг нуля. Принимает единичное значение, когда в качестве результата арифметической операции получился 0.
SF – флаг знака. Этот флаг всегда равен старшему биту результата.
OF – флаг переполнения. Предназначается для работы со знаковыми числами, когда старший бит предполагается знаковым и происходит его переполнение.

Системные флаги:
TF – флаг трассировки. Служит для отладки программы
IF – флаг разрешения прерываний. Если этот флаг выставлен в 1, то прерывания будут срабатывать.
IOPL (появился в 256 процессоре) – уровень приоритета ввода-вывода.
NT (появился в 256 процессоре) – флаг вложенности задач.

Флаг управления
DF – используется в командах поточной обработки данных, для которых нужны регистры SI и DI.

Команда сравнения CMP <приемник>, <источник>
Источник – число, регистр или переменная
Приемник – регистр или переменная (не может быть переменной одновременно с источником)
Вычитает источник из приемника, результат никуда не сохраняется, выставляются флаги CF, PF, AF, ZF, SF, OF

Команды условных переходов J..
// работают как jmp, но переход выполняют не всегда, а только при наличии определенной комбинации флагов

- Переход только 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

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	-

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

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JL JNGE	Если меньше Если не больше и не равно	SF <> OF	да
JGE JNL	Если больше или равно Если не меньше	SF = OF	да
JLE JNG	Если меньше или равно Если не больше	ZF = 1 или SF <> OF	да
JG JNLE	Если больше Если не меньше и не равно	ZF = 0 и SF = OF	да

Команда TEST <приемник>, <источник>

Аналогична CMP, но выполняет не арифметическое вычитание, а логическое умножение (те AND). Результат также, как и в CMP никуда не сохраняется. Выставляются флаги SF, ZF, PF.

С помощью этой команды проверяют не равен ли какой-то регистр 0.

Прерывание
– особая ситуация, когда выполнение текущей программы приостанавливается и управление передается программе-обработчику возникшего прерывания.

Виды прерываний:

- аппаратные (асинхронные) – события от внешних устройств
- внутренние (синхронные) – события в самом процессоре (деление на 0)
- программные – вызванные командой INT

Прерывание DOS 21h (33 в десятичной системе):

- Аналог системного вызова в современных ОС
- Используется наподобие вызова подпрограммы
- Номер функции передается 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 $	-

02 – вывод одного символа в стандартный поток вывода. При этом в ah надо положить 02, в dln нужно положить ascii-код символа и вызвать 21-ое прерывание.

09 – вывод строки символов. На вход этой функции передается адрес строки, то есть должно быть заполнено два регистра. В DS должна быть сегментная часть, а в DX смещение строки. Строка обязательно должна заканчиваться символом $ (по нему определяется докуда нужно вывести).

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	

// “с эхом” означает, что символ не только будет считан, но и окажется на экране в положении курсора

Чтобы считать один символ мы помещаем в ah 01, вызываем прерывание и после этого из al считываем значение.

Чтобы считать строку нужно подготовить соответствующий буфер
