Организация и модели памяти, адресация

Память – способность объекта обеспечивать хранение данных.
Все объекты, над которыми выполняются команды, как и сами команды, хранятся в памяти компьютера.

Память состоит из ячеек, в каждой из которых содержится 1 бит информации, принимающий одно из двух значений: 0 или 1. Биты обрабатывают группами фиксированного размера. Для этого группы бит могут записываться и считываться за одну базовую операцию. Группа из 8 бит называется .

Байты последовательно располагаются в памяти компьютера.

- 1 килобайт (Кбайт) = 2^{10} = 1 024 байт
- 1 мегабайт (Мбайт) = 2^{10} Кбайт = 2^{20} байт = 1 048 576 байт
- 1 гигабайт (Гбайт) = 2^{10} Мбайт = 2^{30} байт = 1 073 741 824 байт

Для доступа к памяти с целью записи или чтения отдельных элементов информации используются идентификаторы, определяющие их расположение в памяти. Каждому идентификатору в соответствие ставится адрес. В качестве адресов используются числа из диапазона от 0 до 2^k-1 со значением k, достаточным для адресации всей памяти компьютера.Все 2^k адресов составляют адресное пространство компьютера.

Способы адресации байтов

Существует прямой и обратный способы адресации байтов.
При обратном способе адресации байты адресуются слева направо, так что самый старший (левый) байт слова имеет наименьший адрес.

Прямым способом называется противоположная система адресации. Компиляторы высокоуровневых языков поддерживают прямой способ адресации.

Объект занимает целое слово. Поэтому для того, чтобы обратиться к нему в памяти, нужно указать адрес, по которому этот объект хранится.

Организация памяти

Физическая память, к которой микропроцессор имеет доступ по шине адреса, называется оперативной памятью ОП (или оперативным запоминающим устройством — ОЗУ). Механизм управления памятью полностью аппаратный, т.е. программа сама не может сформировать физический адрес памяти на адресной шине.

Микропроцессор аппаратно поддерживает несколько моделей использования оперативной памяти:

- сегментированную модель
- страничную модель
- плоскую модель

В сегментированной модели память для программы делится на непрерывные области памяти, называемые сегментами. Программа может обращаться только к данным, которые находятся в этих сегментах.
Сегмент представляет собой независимый, поддерживаемый на аппаратном уровне блок памяти.

Сегментация — механизм адресации, обеспечивающий существование нескольких независимых адресных пространств как в пределах одной задачи, так и в системе в целом для защиты задач от взаимного влияния.

Каждая программа в общем случае может состоять из любого количества сегментов, но непосредственный доступ она имеет только к 3 основным сегментам и к 3 дополнительным сегментам, обслуживаемых 6 сегментными регистрами. К основным сегментам относятся:

- Сегмент кодов (.CODE) – содержит машинные команды для выполнения. Обычно первая выполняемая команда находится в начале этого сегмента, и операционная система передает управление по адресу данного сегмента для выполнения программы. Регистр сегмента кодов (CS) адресует данный сегмент.
- Сегмент данных (.DATA) – содержит определенные данные, константы и рабочие области, необходимые программе. Регистр сегмента данных (DS) адресует данный сегмент.
- Сегмент стека (.STACK). Стек содержит адреса возврата как для программы (для возврата в операционную систему), так и для вызовов подпрограмм (для возврата в главную программу). Регистр сегмента стека (SS) адресует данный сегмент. Адрес текущей вершины стека задается регистрами SS:ESP.

Регистры дополнительных сегментов (ES, FS, GS), предназначены для специального использования.

Для доступа к данным внутри сегмента обращение производится относительно начала сегмента линейно, т.е. начиная с 0 и заканчивая адресом, равным размеру сегмента. Для обращения к любому адресу в программе, компьютер складывает адрес в регистре сегмента и смещение — расположение требуемого адреса относительно начала сегмента. Например, первый байт в сегменте кодов имеет смещение 0, второй байт – 1 и так далее.

Таким образом, для обращения к конкретному физическому адресу ОЗУ необходимо определить адрес начала сегмента и смещение внутри сегмента.
Физический адрес принято записывать парой этих значений, разделенных двоеточием

сегмент : смещение

Страничная модель памяти – это надстройка над сегментной моделью. ОЗУ делится на блоки фиксированного размера, кратные степени 2, например 4 Кб. Каждый такой блок называется страницей. Основное достоинство страничного способа распределения памяти — минимально возможная фрагментация. Однако такая организация памяти не использует память достаточно эффективно за счет фиксированного размера страниц.

Плоская модель памяти предполагает, что задача состоит из одного сегмента, который, в свою очередь, разбит на страницы.
Достоинства:

- при использовании плоской модели памяти упрощается создание и операционной системы, и систем программирования;
- уменьшаются расходы памяти на поддержку системных информационных структур.

В абсолютном большинстве современных 32(64)-разрядных операционных систем (для микропроцессоров Intel) используется плоская модель памяти.

Модели памяти

Директива .MODEL определяет модель памяти, используемую программой. После этой директивы в программе находятся директивы объявления сегментов (.DATA, .STACK, .CODE, SEGMENT). Синтаксис задания модели памяти

.MODEL модификатор **МодельПамяти** СоглашениеОВызовах

Параметр **МодельПамяти** является обязательным.
Основные модели памяти:

Модель памяти	Адресация кода	Адресация данных	Операционная система	Чередование кода и данных
TINY	NEAR	NEAR	MS-DOS	Допустимо
SMALL	NEAR	NEAR	MS-DOS, Windows	Нет
MEDIUM	FAR	NEAR	MS-DOS, Windows	Нет

Модель памяти	Адресация кода	Адресация данных	Операци-онная система	Чередование кода и данных
COMPACT	NEAR	FAR	MS-DOS, Windows	Нет
LARGE	FAR	FAR	MS-DOS, Windows	Нет
HUGE	FAR	FAR	MS-DOS, Windows	Нет
FLAT	NEAR	NEAR	Windows NT, Windows 2000, Windows XP, Windows Vista	Допустимо

Модель tiny работает только в 16-разрядных приложениях MS-DOS. В этой модели все данные и код располагаются в одном физическом сегменте. Размер программного файла в этом случае не превышает 64 Кбайт.
Модель small поддерживает один сегмент кода и один сегмент данных. Данные и код при использовании этой модели адресуются как near (ближние).

Модель medium поддерживает несколько сегментов программного кода и один сегмент данных, при этом все ссылки в сегментах программного кода по умолчанию считаются дальними (far), а ссылки в сегменте данных — ближними (near).
Модель compact поддерживает несколько сегментов данных, в которых используется дальняя адресация данных (far), и один сегмент кода с ближней адресацией (near).
Модель large поддерживает несколько сегментов кода и несколько сегментов данных. По умолчанию все ссылки на код и данные считаются дальними (far).
Модель huge практически эквивалентна модели памяти large.

Особого внимания заслуживает модель памяти flat, которая используется только в 32-разрядных операционных системах. В ней данные и код размещены в одном 32-разрядном сегменте. Для использования в программе модели flat перед директивой .model flat следует разместить одну из директив:

- .386
- .486
- .586
- .686

Желательно указывать тот тип процессора, который используется в машине, хотя это не является обязательным требованием. Операционная система автоматически инициализирует сегментные регистры при загрузке программы, поэтому модифицировать их нужно только в случае если требуется смешивать в одной программе 16-разрядный и 32-разрядный код. Адресация данных и кода является ближней (near), при этом все адреса и указатели являются 32-разрядными.

Параметр **модификатор** используется для определения типов сегментов и может принимать значения use16 (сегменты выбранной модели используются как 16-битные) или use32 (сегменты выбранной модели используются как 32-битные).

Параметр **СоглашениеОВызовах** используется для определения способа передачи параметров при вызове процедуры из других языков, в том числе и языков высокого уровня (C++, Pascal). Параметр может принимать следующие значения:

- C,
- BASIC,

- FORTRAN,
- PASCAL,
- SYSCALL,
- STDCALL.

При разработке модулей на ассемблере, которые будут применяться в программах, написанных на языках высокого уровня, обращайте внимание на то, какие соглашения о вызовах поддерживает тот или иной язык. Используются при анализе интерфейса программ на ассемблере с программами на языках высокого уровня.

Физически ***сегмент*** представляет собой область памяти, занятую командами и (или) данными, адреса которых вычисляются относительно значения в соответствующем сегментном регистре.

Каждая программа содержит 3 типа сегментов:

- Сегмент кодов – содержит машинные команды для выполнения. Обычно первая выполняемая команда находится в начале этого сегмента, и операционная система передает управление по адресу данного сегмента для выполнения программы. Регистр сегмента кодов (CS) адресует данный сегмент.
- Сегмент данных – содержит данные, константы и рабочие области, необходимые программе. Регистр сегмента данных (DS) адресует данный сегмент.
- Сегмент стека — содержит адреса возврата как для программы (для возврата в операционную систему), так и для вызовов подпрограмм (для возврата в главную программу), а также используется для передачи параметров в процедуры. Регистр сегмента стека (SS) адресует данный сегмент. Адрес текущей вершины стека задается регистрами SS:ESP.

Функциональное назначение сегмента несколько шире, чем простое разбиение программы на блоки кода, данных и стека. Сегментация является частью более общего механизма, связанного с концепцией модульного программирования. Она предполагает унификацию оформления объектных модулей, создаваемых компилятором, в том числе с разных языков программирования. Это позволяет объединять программы, написанные на разных языках. Именно для реализации различных вариантов такого объединения и предназначены директивы сегментации.

Упрощенные директивы сегментации

Для задания сегментов в тексте программы можно пользоваться упрощенными директивами:

- .CODE — для указания начала сегмента кода;
- .DATA — для указания начала сегмента данных;
- .STACK — для указания начала сегмента стека.

Однако использование упрощенных директив сегментации не позволяет создать более трех сегментов для одной программы.

Стандартные директивы сегментации

Наряду с упрощенными директивами сегментации может также использоваться стандартная директива SEGMENT, которая определяет начало любого сегмента.
Синтаксис:

ИмяСегмента SEGMENT align combine dim 'class'
…
ИмяСегмента ENDS

Директива ENDS определяет конец сегмента.

Атрибут выравнивания сегмента (тип выравнивания) align сообщает компоновщику о том, что нужно обеспечить размещение начала сегмента на заданной границе. Это важно, поскольку при правильном выравнивании доступ к данным в процессорах, совместимых с базовым i8086, выполняется быстрее. Допустимые значения этого атрибута следующие:

- BYTE — выравнивание не выполняется. Сегмент может начинаться с любого адреса памяти;
- WORD — сегмент начинается по адресу, кратному двум, то есть последний (младший) значащий бит физического адреса равен 0 (выравнивание на границу слова);
- DWORD — сегмент начинается по адресу, кратному четырем, то есть два последних (младших) значащих бита равны 0 (выравнивание по границе двойного слова);
- PARA — сегмент начинается по адресу, кратному 16, то есть последняя шестнадцатеричная цифра адреса должна быть 0h (выравнивание по границе параграфа);
- PAGE — сегмент начинается по адресу, кратному 256, то есть две последние шестнадцатеричные цифры должны быть 00h (выравнивание по границе страницы размером 256 байт);
- MEMPAGE — сегмент начинается по адресу, кратному 4 Кбайт, то есть три последние шестнадцатеричные цифры должны быть 000h (адрес следующей страницы памяти размером 4 Кбайт);

По умолчанию тип выравнивания имеет значение PARA.
Атрибут комбинирования сегментов (комбинаторный тип) combine сообщает компоновщику, как нужно комбинировать сегменты различных модулей, имеющие одно и то же имя. По умолчанию атрибут комбинирования принимает значение PRIVATE.
Значениями атрибута комбинирования сегмента могут быть:

- PRIVATE — сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля;
- PUBLIC — заставляет компоновщик соединить все сегменты с одинаковым именем. Новый объединенный сегмент будет целым и непрерывным. Все адреса (смещения) объектов, а это могут быть, в зависимости от типа сегмента, команды или данные, будут вычисляться относительно начала этого нового сегмента;
- COMMON — располагает все сегменты с одним и тем же именем по одному адресу. Все сегменты с данным именем будут перекрываться и совместно использовать память. Размер полученного в результате сегмента будет равен размеру самого большого сегмента;
- AT xxxx — располагает сегмент по абсолютному адресу параграфа (параграф — объем памяти, кратный 16, поэтому последняя шестнадцатеричная цифра адреса параграфа равна 0). Абсолютный адрес параграфа задается выражением xxxx.

Компоновщик располагает сегмент по заданному адресу памяти (это можно использовать, например, для доступа к видеопамяти или области ПЗУ), учитывая атрибут комбинирования. Физически это означает, что сегмент при загрузке в память будет расположен, начиная с этого абсолютного адреса параграфа, но для доступа к нему в соответствующий сегментный регистр должно быть загружено заданное в атрибуте значение. Все метки и адреса в определенном таким образом сегменте отсчитываются относительно заданного абсолютного адреса;

- STACK — определение сегмента стека. Заставляет компоновщик соединить все одноименные сегменты и вычислять адреса в этих сегментах относительно регистра SS. Комбинированный тип STACK (стек) аналогичен комбинированному типу PUBLIC, за исключением того, что регистр SS является стандартным сегментным регистром для сегментов стека. Регистр SP устанавливается на конец объединенного сегмента стека. Если не указано ни одного сегмента стека, компоновщик выдаст предупреждение, что стековый сегмент не найден. Если сегмент стека создан, а комбинированный тип STACK не используется, программист должен явно загрузить в регистр SS адрес сегмента (подобно тому, как это делается для регистра DS).

Атрибут размера сегмента dim. Для процессоров i80386 и выше сегменты могут быть 16- или 32-разрядными. Это влияет прежде всего на размер сегмента и порядок формирования физического адреса внутри него. Атрибут может принимать следующие значения:

- USE16 — это означает, что сегмент допускает 16-разрядную адресацию. При формировании физического адреса может использоваться только 16-разрядное смещение. Соответственно, такой сегмент может содержать до 64 Кбайт кода или данных;
- USE32 — сегмент будет 32-разрядным. При формировании физического адреса может использоваться 32-разрядное смещение. Поэтому такой сегмент может содержать до 4 Гбайт кода или данных. В модели памяти FLAT используется по умолчанию именно это значение атрибута размера сегмента

Атрибут класса сегмента (тип класса) ‘class’ — это заключенная в кавычки строка, помогающая компоновщику определить соответствующий порядок следования сегментов при сборке программы из сегментов нескольких модулей. Компоновщик объединяет вместе в памяти все сегменты с одним и тем же именем класса (имя класса, в общем случае, может быть любым, но лучше, если оно будет отражать функциональное назначение сегмента). Типичным примером использования имени класса является объединение в группу всех сегментов кода программы (обычно для этого используется класс ‘code’). С помощью механизма типизации класса можно группировать также сегменты инициализированных и неинициализированных данных.

Все сегменты сами по себе равноправны, так как директивы SEGMENT и ENDS не содержат информации о функциональном назначении сегментов. Для того чтобы использовать их как сегменты кода, данных или стека, необходимо предварительно сообщить транслятору об этом, для чего используют специальную директиву ASSUME. Эта директива сообщает транслятору о том, какой сегмент к какому сегментному регистру привязан. В свою очередь, это позволит транслятору корректно связывать символические имена, определенные в сегментах. Привязка сегментов к сегментным регистрам осуществляется с помощью операндов этой директивы, в которых **ИмяСегмента** должно быть именем сегмента, определенным в исходном тексте программы директивой SEGMENT или ключевым словом nothing. Если в качестве операнда используется только ключевое слово nothing, то предшествующие назначения сегментных регистров

аннулируются, причем сразу для всех шести сегментных регистров. Но ключевое слово nothing можно использовать вместо аргумента **ИмяСегмента**, в этом случае будет выборочно разрываться связь между сегментом с именем **ИмяСегмента** и соответствующим сегментным регистром.
Директива SEGMENT может применяться с любой моделью памяти. При использовании директивы SEGMENT с моделью flat требуется указать транслятору, что все сегментные регистры устанавливаются в соответствии с моделью памяти flat. Это можно сделать при помощи директивы ASSUME:

```
ASSUME CS:FLAT, DS:FLAT, SS:FLAT, ES:FLAT, FS:ERROR, GS:ERROR
```

Регистры FS и GS программами не используются, поэтому для них указывается атрибут ERROR.