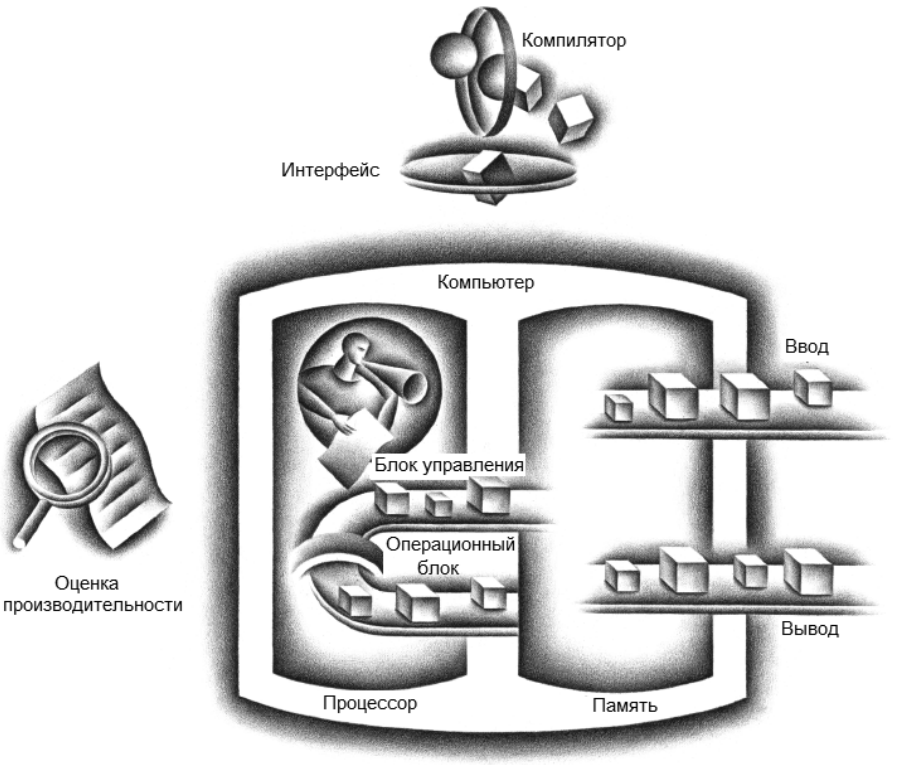
Глава 5

Объемная и быстродействующая: анализ иерархии памяти

В идеале желательно получить бесконечно большой объем памяти, и чтобы любое конкретное ... слово было моментально доступно ... Мы ... вынуждены признать возможность построения иерархии устройств памяти, каждое из которых имеет больший объем, чем предыдущее, но при менее быстром доступе к нему.

А. В. Буркс, Х. Х. Гольдштейн и Дж. фон Нейман



5.1. Введение

С самых первых дней существования компьютерной техники программистам хотелось иметь неограниченные объемы быстродействующей памяти. Темы, рассматриваемые в данной главе, помогают программистам в воплощении этой иллюзии в жизнь. Перед тем как приступить к созданию этой иллюзии, давайте рассмотрим простую аналогию, которая послужит иллюстрацией используемых нами ключевых принципов и механизмов.

Предположим, что вам приходилось в студенческие времена писать курсовую работу о важных исторических событиях в мире компьютерного оборудования. Вы сидите за столом в библиотеке с подборкой книг, которые вы сняли с полок и изучаете. Оказалось, что часть важных компьютеров, о которых нужно было написать, описаны в имеющихся у вас книгах, но в них нет никаких сведений о компьютере EDSAC. Поэтому вы возвращаетесь к полкам и ищете еще одну книгу. Вы нашли книгу о первых английских компьютерах, в которой есть сведения о EDSAC. Как только у вас на столе окажется хорошая подборка литературы, возникнет высокая вероятность, что многое о том, что вам нужно, будет найдено в этих книгах, и основная часть времени будет затрачена на работу.

Такой же принцип позволяет нам создать иллюзию большого объема памяти, к которой можно обращаться так же быстро, как и к очень маленькому по объему устройству памяти. Точно так же, как у вас не возникает потребность просмотра сразу всех книг в библиотеке, программа не обращается одновременно с равной вероятностью ко всему своему коду или данным. В противном случае было бы невозможно обеспечить быстрый доступ к основному объему памяти и при этом оснащать компьютеры большими объемами памяти, точно так же, как вам было бы невозможно поместить все книги, имеющиеся в библиотеке, на вашем столе и сохранить возможность быстрого поиска нужного материала.

В обоих случаях, и при работе в библиотеке, и в работе программы, в основу кладется *принцип локальности*. Этот принцип гласит, что программы в любой момент времени обращаются к относительно небольшой части своего адресного пространства, точно так же, как вы обращаетесь к очень маленькой части библиотечной коллекции. Существуют две разновидности локальности:

- **Локальность, связанная со временем:** если было обращение к элементу, то есть вероятность, что к нему скоро последует новое обращение. Если вы недавно приносили книгу на свой стол для просмотра, то, возможно, совсем скоро она понадобится вам еще раз.
- **Локальность, связанная с пространством:** если было обращение к элементу, то, скорее всего, скоро будет обращение и к тем элементам, чьи адреса находятся поблизости.

Локальность, связанная со временем
Принцип, в котором утверждается, что если было обращение к элементу данных, то высока вероятность, что к нему скоро последует новое обращение.

Локальность, связанная с пространством
Принцип локальности, в котором утверждается, что если было обращение к элементу данных, то высока вероятность, что скоро последует обращение и к элементам данных по соседним адресам.

Например, когда вы нашли книгу по первым английским компьютерам для поиска материалов по EDSAC, вы при этом заметили, что рядом с ней на полке стояла еще одна книга по первым механическим компьютерам, и вы захватили с собой еще и эту книгу, и позже нашли в ней кое-что полезное для вас. В библиотеках книги по сходной тематике соседствуют друг с другом, повышая показатель локальности, связанной с пространством. Как локальность, связанная с пространством, используется в иерархии памяти, мы увидим в данной главе чуть позже.

Точно так же, как обращение к книгам, лежащим на столе, является естественным проявлением локальности, в программах локальность возникает благодаря простым и естественным программным структурам. Например, многие программы содержат циклы, поэтому инструкции и данные, к которым, скорее всего, последует повторное обращение, демонстрируют локальность, связанную со временем. Поскольку обычно используется последовательное обращение к инструкциям, программы также демонстрируют высокую локальность, связанную с пространством. Обращения к данным также являются естественным проявлением локальности, связанной с пространством. Например, у последовательного обращения к элементам массива или к записи вполне естественной будет высокая степень локальности, связанной с пространством.

Принцип локальности успешно используется путем реализации памяти компьютера в виде **иерархии памяти**. Эта иерархия состоит из нескольких уровней памяти с разными скоростями и объемами. Более быстродействующие устройства памяти имеют более высокий показатель стоимости бита, чем менее быстродействующие устройства памяти, поэтому они имеют меньший объем.

Сегодня для построения иерархии памяти используются три основные технологии. Оперативная память выполнена из динамических запоминающих устройств с произвольной выборкой — DRAM (dynamic random access memory), а уровни, находящиеся ближе к процессору (кэш-память), используют статические запоминающие устройства с произвольной выборкой — SRAM (static random access memory). У DRAM-памяти меньше стоимость бита, чем у SRAM, хотя она существенно медленнее. Разница в стоимости обусловлена тем, что в DRAM используется значительно меньшая площадь на один бит, и у DRAM-элементов более высокая емкость; разница в скорости обусловлена сразу несколькими факторами. Третьей технологией, используемой для реализации наиболее объемного и самого медленного уровня иерархии, обычно является магнитный диск. (Во многих встроенных устройствах вместо дисков используется флэш-память; см. раздел 6.4.) Время доступа к биту данных и его стоимость широко варьируются в зависимости от того, какая из этих технологий применяется. Эти показатели, применительно к значениям 2008 года, показаны в следующей таблице.

Иерархия памяти
Структура, использующая несколько уровней памяти; по мере удаления от процессора увеличивается объем памяти и время доступа к ней.

Технология памяти	Обычное время доступа, нс	Цена в долларах за 1 Гбайт в 2008 году
SRAM	0,5–2,5	2000–5000
DRAM	50–70	20–75
Магнитный диск	5 000 000–20 000 000	0,20–2

Из-за различий в стоимости и времени доступа предпочтительнее строить память в виде иерархии уровней. На рис. 5.1 показано, что самая быстродействующая память находится ближе к процессору, а медленная, менее дорогая память, находится ниже его. Цель состоит в том, чтобы снабдить пользователя как можно большим объемом по самой дешевой технологии, обеспечивая ему доступ на скорости, предлагаемой самой быстродействующей памятью.

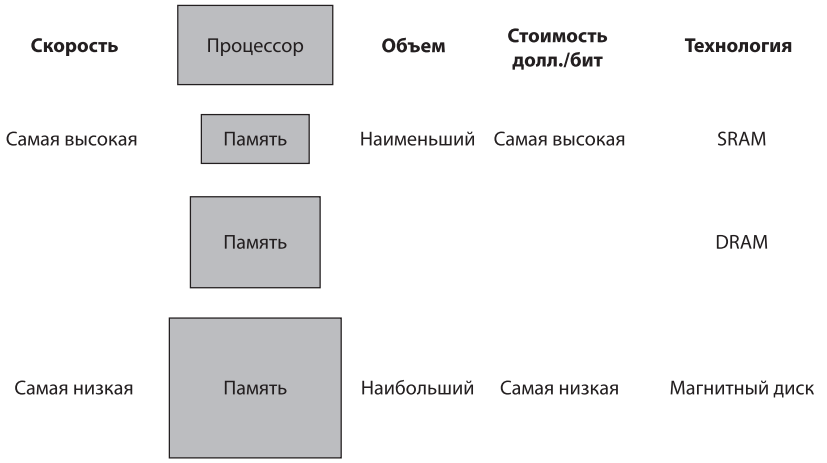


Рис. 5.1. Основная структура иерархии памяти. За счет иерархической реализации системы памяти у пользователя создается иллюзия, что она по объему соответствует самому большому уровню иерархии, но может быть доступна, как будто изготовлена из самых быстродействующих элементов. Флэш-память заменила диски во многих встраиваемых устройствах и может привести к появлению нового уровня в иерархии запоминающих устройств для настольных компьютеров и серверов; см. раздел 6.4

Данные имеют аналогичную иерархию: тот уровень, который ближе к процессору, обычно является поднабором любого, более отдаленного уровня, а все данные хранятся на самом нижнем уровне. По аналогии, книги на вашем столе составляют поднабор библиотеки, в которой вы работаете, которая, в свою очередь, является поднабором всех библиотек на территории университета. Более того, по мере удаления от процессора доступ к уровням постепенно становятся более длительным, точно так же, как это может быть в иерархии университетских библиотек.

Иерархия памяти может состоять из нескольких уровней, но данные единовременно копируются только между двумя смежными уровнями, поэтому мы можем сконцентрировать свое внимание только на двух уровнях. Самый верхний уровень — тот, что ближе всех к процессору, — меньше и быстрее расположенного ниже уровня, поскольку на более высоком уровне используется более дорогостоящая технология. На рис. 5.2 показано, что минимальная порция информации, которая может либо присутствовать, либо не присутствовать в двухуровневой иерархии, называется **блоком** или **строкой**; в нашей библиотечной аналогии блок информации соответствует книге.

Если данные, запрашиваемые процессором, имеются в каком-нибудь блоке на верхнем уровне, то это называется *попаданием* (по аналогии с тем, как если бы вы нашли информацию в одной из книг на своем столе). Если данные не найдены на верхнем уровне, запрос называется *промахом* (неудачей). Затем следует обращение к более низкому уровню для извлечения блока, содержащего запрошенные данные. (Продолжая нашу аналогию, вы идете от стола к полкам, чтобы найти нужную книгу.) **Коэффициент попаданий**, или *результативность*, является долей обращений к памяти, удовлетворенных на верхнем уровне; он часто используется в качестве показателя производительности иерархии памяти. **Коэффициент промахов** (1 – коэффициент попаданий) является долей обращений к памяти, которые не были удовлетворены на верхнем уровне.

Поскольку иерархия памяти главным образом создавалась ради повышения производительности, важная роль уделяется времени обслуживания попаданий и промахов. **Время попадания** затрачивается на доступ к верхнему уровню иерархии памяти, в него включается время, необходимое для определения, будет ли обращение попаданием или промахом (то есть время, необходимое для просмотра всех книг на столе). Время промаха, или **издержка**, складывается из времени на замену блока в верхнем уровне соответствующим блоком из нижнего уровня, сюда же прибавляется и время на доставку этого блока процессору (время, затрачиваемое на то, чтобы взять с полки еще одну книгу и положить ее на стол). Поскольку верхний уровень меньше по объему и построен из элементов памяти с более высоким быстродействием, время попадания будет намного меньше, чем время доступа к следующему уровню иерархии, которое является основной составляющей издержки. (Время на изучение книг на столе намного меньше времени, необходимого на то, чтобы встать и принести с полки новую книгу.)

Как будет показано в данной главе, концепция, использованная для создания систем памя-

Блок (или строка)
Минимальная порция информации, которая может либо присутствовать, либо не присутствовать в кэш-памяти.

Коэффициент попаданий
Доля обращений к памяти, удовлетворенных на верхнем уровне ее иерархии.

Коэффициент промахов
Доля обращений к памяти, не удовлетворенных на верхнем уровне ее иерархии.

Время попадания
Время, необходимое для доступа к уровню иерархии памяти, включая и время, необходимое для определения попадания или промаха.

Издержка промаха
Время, необходимое для извлечения блока на уровень иерархии памяти из нижнего уровня, включая время доступа к блоку, передачи его с одного уровня на другой, вставки его в уровень, обращение к которому было неудачным, и затем передачи блока инициатору запроса.

ти, оказывает влияние на многие другие аспекты компьютера, включая и то, как операционная система управляет памятью и вводом-выводом, как компиляторы генерируют код и даже то, как приложения используют возможности компьютера. Разумеется, поскольку все программы тратят много времени на доступ к памяти, система памяти неизменно является главным фактором в определении производительности. Зависимость достижения высокой производительности от иерархии памяти означает, что программисты, привыкшие думать о памяти, как о неком одномерном запоминающем устройстве с произвольным доступом, теперь для получения высокой производительности должны понимать, что память представляет собой сложную иерархию. Вся важность такого представления о системе памяти будет показана на следующих примерах, в частности она будет продемонстрирована на рис. 5.15.

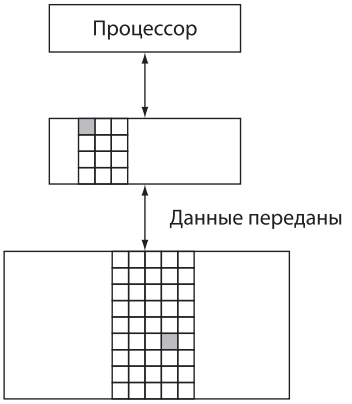


Рис. 5.2. Каждую пару уровней в иерархии памяти можно представлять в виде верхнего и нижнего уровней. В пределах каждого уровня порция присутствующей или отсутствующей в нем информации называется блоком или строкой. Обычно при копировании каких-либо данных между уровнями передается сразу весь блок

Поскольку системы памяти играют важную роль для достижения высокой производительности, разработчики компьютеров уделяют большое внимание этим системам и разрабатывают сложные механизмы для повышения производительности их работы. В данной главе будут рассмотрены основные концептуальные идеи, но при этом будет использоваться множество упрощений и абстракций, чтобы не усложнять материал.

Общее представление

Программы демонстрируют как локальность, связанную со временем, то есть тенденцию к повторному использованию недавно запрошенных элементов данных, так и локальность, связанную с пространством, то есть тенденцию обращения к элементам данных, находящимся по соседству с недавно запрошенными элементами данных. Иерархии памяти используют локальность, связанную со временем, располагая последние запрошенные данные ближе к процессору. А локальность,

связанная с пространством, используется ими путем перемещения блоков, состоящих из нескольких смежных слов в памяти, на более высокие уровни иерархии.

На рис. 5.3 показано, что ближе к процессору иерархии памяти используют менее объемные и более быстродействующие технологии хранения данных. Таким образом, обращение, удовлетворяемое на самом высоком уровне иерархии, может быть обработано быстрее. Неудовлетворенные обращения перенаправляются на нижние уровни иерархии, у которых больше объем, но ниже быстродействие. Если коэффициент попадания достаточно высок, иерархия памяти обладает эффективным временем доступа, приближающимся ко времени, характерному для наивысшего (и самого быстродействующего) уровня, и к объему, соизмеримому с тем, который характерен для самого низкого (и самого объемного) уровня.

Фактически большинство систем имеют память с истинно иерархической структурой, это означает, что данные не могут присутствовать на уровне i, пока они не будут также присутствовать и на уровне $i + 1$.

Рис. 5.3. На этой схеме показана структура иерархии памяти: по мере увеличения расстояния от процессора растет и объем. Эта структура с соответствующими рабочими механизмами позволяет процессору иметь время доступа, которое определяется главным образом уровнем *1* иерархии, и при этом обладать памятью с объемом, соответствующим уровню *n*. Воплощение в жизнь этой иллюзии и является темой данной главы. Хотя в нижней части иерархии обычно находится локальный диск, в некоторых системах в качестве следующих уровней иерархии используется магнитная лента или файловый сервер, доступный по локальной сети

Самопроверка

Какие из следующих утверждений в целом соответствуют действительности?

1. Кэш-память использует локальность, связанную со временем.
2. При чтении возвращаемое значение зависит от того, какой блок находится в кэш-памяти.
3. Основная стоимость иерархии памяти приходится на самый верхний уровень.
4. Основной объем иерархии памяти приходится на самый нижний уровень.

5.2. Основы кэш-памяти

> Кэш: укромное место, позволяющее прятать или хранить вещи.
>
> *«Webster's New World Dictionary of the American Language», Third College Edition, 1988 г.*

В нашем примере с библиотекой стол работает как кэш — укромное место для хранения вещей (книг), которые нужно изучить. Название *кэш* было выбрано для представления дополнительного уровня иерархии памяти между процессором и оперативной памятью в первом коммерческом компьютере. Кэш является простой заменой устройств памяти в операционном блоке, рассмотренном в главе 4. Но в наши дни употребление слова *кэш* в данном смысле стало доминирующим, этот термин используется также для ссылки на любые устройства хранения информации, предназначенные для использования принципа локальности доступа. Сначала кэш-память появилась на исследовательских компьютерах начала 1960-х годов, а чуть позже, в том же десятилетии, она появилась и на компьютерах, предназначенных для управления производством; сегодня кэш-память является атрибутом всех универсальных компьютеров, от серверов и до маломощных встроенных процессоров.

В данном разделе сначала будет рассмотрена очень простая кэш-память, в которой процессор делает пословные запросы, а блоки также состоят из одного слова. (Читатели, уже знакомые с основами кэш-памяти, могут раздел 5.2 пропустить.) На рис. 5.4 эта простая кэш-память показана до и после запроса элемента данных, который изначально в ней отсутствовал. Перед запросом в кэш-памяти находилась коллекция данных, на которые были последние ссылки X_1, X_2, ... , X_{n-1}, а процессор запросил слово X_n, которого в кэш-памяти нет. Этот запрос оказался неудачным, и слово X_n было перенесено из оперативной памяти в кэш.

При рассмотрении сценария на рис. 5.4 возникают два вопроса. Как мы узнаем, что элемент данных находится в кэш-памяти? И потом, если он там, то как мы его найдем? Ответы на эти вопросы взаимосвязаны. Если каждое слово может попадать во вполне определенное одно и то же место в кэш-памяти, то найти слово, находящееся в кэш-памяти, будет нетрудно. Наиболее простым способом определения места в кэш-памяти для каждого слова является определение места в кэш-памяти на основе *адреса* слова в памяти. Такая структура кэша называется **непосредственно отображенной**, поскольку каждое место в памяти отображается непосредственно на одно место в кэш-памяти. Обычное отображение между адресами и местами в кэше для кэш-памяти с непосредственным отображением, как правило, устроено довольно просто. Например, почти во всех устройствах кэш-памяти с непосредственным отображением для поиска блока используется следующее проецирование:

(Адрес блока) modulo (Количество блоков в кэш-памяти)

Если количество записей в кэш-памяти является степенью числа 2, то модуль (modulo) может быть вычислен просто за счет использования младших $\log_2$ (размер кэш-памяти в бло-

Кэш-память с непосредственным отображением
Структура кэша, в которой каждое место в памяти проецируется на одно конкретное место в кэш-памяти.

ках) разрядов адреса. Таким образом, кэш-память из восьми блоков использует три самых младших разряда ($8 = 2^3$) адреса блока. Например, на рис. 5.5 показано, как память с адресами между 1_{10} (00001_2) и 29_{10} (11101_2) отображается на места 1_{10} (001_2) и 5_{10} (101_2) в кэш-памяти с непосредственным отображением, состоящей из восьми слов.

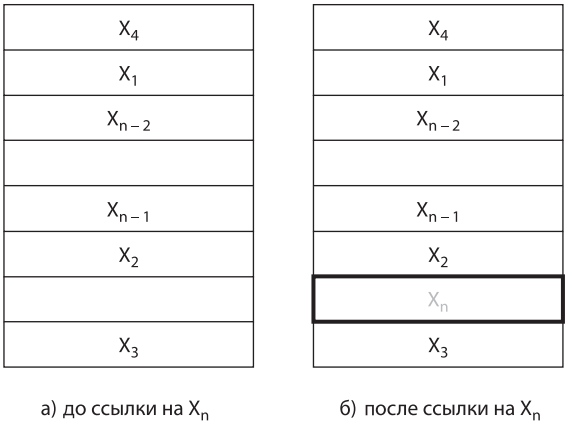


Рис. 5.4. Кэш-память непосредственно перед и сразу после обращения к слову X_n, изначально отсутствующему в кэш-памяти. Это обращение становится причиной промаха, заставляющего кэш-память извлечь X_n из памяти и вставить в себя этот элемент

Поскольку в каждом месте кэш-памяти может находиться содержимое из разных мест памяти, то как мы узнаем, соответствуют данные запрошенному слову или нет? То есть как мы узнаем, имеется ли запрошенное слово в кэш-памяти или нет? Ответ на этот вопрос будет дан за счет добавления к кэш-памяти **тегов**. Тег содержит адресную информацию, необходимую для идентификации соответствия слова в кэш-памяти запрашиваемому слову. В теге нужно содержать лишь старшую часть адреса, соответствующую разрядам, не используемым в качестве индекса в кэш-памяти. Например, на рис. 5.5 нам нужны в теге только старшие два из пяти адресных разрядов, поскольку младшие три разряда составляют индексное поле адреса, выбирающее блок. Разработчики опускают разряды индекса, потому что они избыточны, поскольку по определению индексное поле любого адреса блока кэш-памяти должно быть номером этого блока.

Тег
Поле в таблице, используемой для иерархии памяти, которое содержит адресную информацию, необходимую для определения, соответствует ли связанный блок в иерархии запрошенному слову.

Бит достоверности
Поле в таблицах иерархии памяти, показывающее, что связанный с ним блок в иерархии содержит достоверные данные.

Нам также нужен способ определения того, что блок кэш-памяти не содержит достоверной информации. Например, когда процессор начинает работу, кэш-память не содержит достоверных данных, и поля тегов не будут иметь никакого смысла. Даже после выполнения множества инструкций некоторые из записей кэш-памяти могут все еще быть пустыми, как показано на рис. 5.4. Таким образом, нам нужно знать, что тег для таких записей должен быть

проигнорирован. Наиболее распространенным является метод добавления **бита достоверности**, показывающего, содержит ли запись достоверный адрес. Если разряд не установлен, то для этого блока не может быть соответствия.

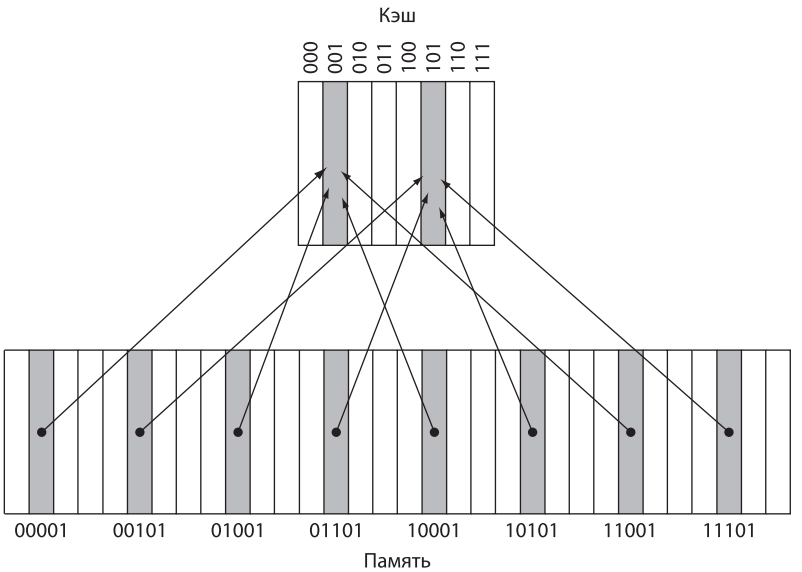


Рис. 5.5. Кэш-память с непосредственным отображением, имеющая восемь записей, с указанием адресов слов памяти в диапазоне от 0 до 31, отображаемых на одни и те же места в кэш-памяти. Поскольку в кэш-памяти восемь слов, адрес X проецируется на имеющееся в кэш-памяти с непосредственным отображением слово X по модулю 8. То есть младшие $\log_2(8) = 3$ разряда используются в качестве индекса кэш-памяти. Таким образом, адреса 00001_2, 01001_2, 10001_2 и 11001_2 проецируются на запись 001_2 кэш-памяти, а адреса 00101_2, 01101_2, 10101_2 и 11101_2 проецируются на запись 101_2 кэш-памяти

В остальной части данного раздела основное внимание будет уделено объяснению того, как кэш-память справляется с чтением. В целом, справиться с чтением немного легче, чем с записью, поскольку чтение не должно изменять содержимое кэш-памяти. После изучения основ работы чтения и обработки промахов при обращениях к кэш-памяти мы рассмотрим конструкции кэш-памяти для настоящих компьютеров и займемся вопросом о том, как эти конструкции справляются с записью.

Обращение к кэш-памяти

Ниже показана последовательность из девяти обращений к памяти, приходящихся на пустую кэш-память, состоящую из восьми блоков, включая действие, предпринимаемое при каждом обращении. На рис. 5.6 показано, как содержимое кэш-памяти изменяется после каждого промаха. Поскольку в кэш-памяти восемь блоков, самые младшие три разряда адреса дают номер блока: