Урок 4

Память

Оперативное запоминающее устройство и его связь с процессором устройством. ОЗУ, его быстродействие, объём как ресурс операционной системы. Тип памяти и способы работы с ней.

Введение

В настоящее время определить, чем физически является оперативная память, довольно сложно. Сверхбыстрая оперативная память – регистры процессора, – фактически реализованы внутри CPU. Также в составе процессора имеется многоуровневый кеш, получающий код и данные из памяти ещё до начала их выполнения.

Во многом оперативная память – это то, что находится в RAM, но не обязательно.

Виртуальная память (файл подкачки, раздел подкачки) позволяет выгрузить часть относительно редко используемых данных на диск.

Таким образом, оперативная память, используемая в системе, распределена по нескольким разнородным устройствам.

Разрядность памяти

Способность процессора работать с тем или иным объёмом памяти определяется его разрядностью. В простейшем случае регистр указателя инструкции и регистры для указателя на данные позволяют использовать адреса, не превосходящие максимальное число, которое может быть записано в регистр. Таким образом, максимальный размер памяти, адресуемый неким регистром X с битностью Y, вычисляется как 2^Y байт: теоретически четырехбитный регистр может адресовать 2^4 =16 байт, восьмибитный – уже 2^8 =256 байт, а шестнадцатибитный, соответственно, 2^{16} =65 636 байт.

Объем растет экспоненциально.

32-битный 2^{32}= 4 294 967 296 байт, что равно порядка 4 Гигабайт. Отсюда и ограничение по памяти 32-битных машин, которого нет у 64-битных.

64-битный 2^{64}= 18 446 744 073 709 551 616 = 16 777 216 Терабайт. Кстати, такое число зерен находится на последней клетке в задаче о зёрнах на шахматной доске.

Существуют также 128-битные регистры – к слову сказать, 128-битные адреса используются в GUID в GPT, IPv6-адреса имеют размер в 128 бит, современные процессоры содержат 128-битные регистры SSE для операций над числами с плавающей точкой. Впрочем, это ещё не говорит, что сам процессор – 128-битный: битность процессора определяется основными используемыми регистрами, операциями над ними и способом манипуляции с памятью. Теоретически возможны и полноценные 128-битные процессоры, а графические процессоры могут использовать и 512-битные регистры. Впрочем, в настоящее время фактическим пределом битности является все же показатель в 64 бита.

Если ситуация с 32-битной и 64-битной адресацией понятна – 32-битные машины могут использовать до 4 Гб, а 64-битные свыше 4 Гб, – то 16-битная машина, получается, может задействовать всего 64 Кбайта. Это даже не 640 Кбайт, которых, как обещал в своё время Билл Гейтс, хватит всем.

4004 процессор адресовал до 80 ячеек по 4 бита, но даже для этого не хватило бы содержимого регистра. В этом случае применялось значение пары регистров.

8080 был восьмибитным, но использовал 16-битную шину данных, что позволяло ему адресовать до 64 Кбайт памяти (и инструкций, и данных). Для этого также использовалась пара восьмибитных регистров, воспринимаемая как один 16-битный. Тем не менее процессор был 8-битным, так как операции выполнялись над 8-битными регистрами. Многие восьмибитные процессоры той эпохи (8080, ZX80 и т.д.) позволяли работать с 64 Кб памяти.

8086 – первый 16-битный процессор. Тем не менее 64 Кбайт памяти было уже мало. 20-битная шина адреса позволяла адресовать 2^{20}=1 048 576 = 1 Мбайт памяти. Но для адресации 1 Мбайта уже не хватало бы даже 16-битного регистра. Для этого была придумана схема сегмент:смещение, которая надолго определила стиль написания программ для MS DOS. Такой режим позже получил название реального режима 8086.

Реальный режим 8086

Операция сдвига

Одна из наиболее простых в аппаратной реализации арифметических операций в процессорах – операция сдвига. Сдвиг на единицу влево преобразует двоичный 1 в 10 (2), далее в 100 (4), далее в 1000 (8) и т.д. Фактически в этом случае происходит умножение на соответствующие степени двойки. Операция сдвига вправо аналогична операции деления. Существуют разновидности сдвига (арифметический, циклический), но в целом в них действует общая идея.

Сегментные регистры

В 8086 появились сегментные регистры:

- CS (Code Segment) – сегмент кода;
- DS (Data Segment) – сегмент данных;
- ES (ExtraSegment) – дополнительный сегмент;
- SS (Stack Segment) – сегмент стека.

Сегмент CS использовался для адресации к коду программы, DS и ES – для данных, SS – для стека.

Стек

Стек – это особый способ адресации памяти, так называемый LIFO (Last In, First Out, «последним пришёл – первым ушёл»). Для работы со стеком используются команды ассемблера push (положить на вершину стека) и pop (снять с вершины стека). Отдельные операции pushf и popf позволяли поместить или извлечь из стека флаговые регистры. Стек активно принимает участие в сохранении состояний, при вызове функций и прерываний. Так как стек растёт только в одну сторону, может произойти ошибка переполнения стека (Stack overflow). Такая ситуация может случаться при применении рекурсии и неизбежно произойдёт при бесконечной рекурсии, что вызовет переполнение процессора. Существуют языки программирования и процессоры, ориентированные на стек. Это язык Forth и Forth-процессоры.

Сегментная адресация

Что же такое сегменты? Если в 8080-подобных процессорах (ZX80 и т.д.) адресное пространство ограничивалось 64 Кбайтами, то для доступа к коду (регистр IP) и данным (например, регистр BX) по-прежнему использовались 16-битные регистры – они позволяли адресовать 64 Кбайт, но памяти было больше. Для увеличения памяти использовались так называемые сегменты, содержащие по 64 Кбайта. Доступ к ячейке памяти происходил путём адресации к памяти через 20-битную шину данных, фактический адрес высчитывался благодаря сдвигу влево сегментного регистра на 4 бита (фактически аппаратное умножение на 16) и прибавление так называемого смещения. В INTEL-ассемблере сегмент:смещение записывается через двоеточие.

Примеры:

- mov ax, [bx] – указание на пересылку значения из памяти в ds:bx в регистр ax;
- mov ax, es:[bx] – явное указание переслать значение из памяти в es:bx в регистр ax;
- jmp xxx – изменение регистра ip и переход к коду, находящемуся в памяти по адресу cs:ip.

Подобный способ адресации позволял адресовать объём памяти, превышающий 64 Кбайт. Благодаря такому способу расчета физической памяти становилось возможно адресовать одну и ту же ячейку памяти разными способами, то есть сегмент – это не некая изолированная область памяти, а логическая абстракция.

Так, адреса 0x0400:0x0001 и 0x0000:0x4001 будут ссылаться на одну и ту же ячейку памяти, поскольку 0x400 × 16 + 1 = 0 × 16 + 0x4001.

Более того, попытки таким образом адресовать данные за пределами 1 Мб приводили к обращению к начальной области данных. В 80286 даже понадобилось реализовать режим совместимости с учётом данной «недокументированной возможности» (то есть ошибки).

Достоинства сегментной адресации:

- возможность выделять программам разные сегменты;
- возможность адресовать адресное пространство большее, чем позволяет разрядность процессора.

Недостатки сегментной адресации:

- неоднозначность задания адресом сегмент:смещение адреса, выставляемого на адресную шину для доступа к памяти;
- неизолированность сегментов;
- неинтуитивный механизм формирования.

Реальный режим

Наиболее серьёзным недостатком механизма управления памятью в 8086 (а также в 8088 и 80186) была невозможность защиты памяти. Все программы могли влиять друг на друга, не приходилось говорить о пространстве ядра и пространстве пользователя: все приложения имели равный доступ ко всей памяти ЭВМ.

Такой режим не носил определённого названия, и лишь в дальнейшем, с появлением процессора 80286 и «защищённого режима» 286, режим работы 8086 получил название «реального режима».

Реальный режим продолжал использоваться и в процессорах 80286, 80386, 80486. Именно в реальном режиме по умолчанию выполнялась операционная система MS DOS.

Итак, процессор 8086 мог адресовать память размером до 1 Мб. Из них прикладным программам отводилось всего 640 Кб (те самые, о которых Билл Гейтс говорил, что «хватит всем»). Конечно, так продолжаться не могло, и появился процессор 80286.

Реальный режим процессора 80286

Процессор 80286 имел не 20-битную, а 24-битную шину, что позволяло адресовать уже 2^{24}=16 Мб памяти. Однако возникли проблемы: например, как быть с совместимостью с привичной багофичей процессора 8086? Был добавлен элемент (Gate 20), который позволял отключить 21 линию адресной шины – таким образом, процессор работал аналогично 8086, обращаясь к начальной памяти. При этом появлялась возможность адресовать до 16 Мбайт верхней памяти (High Memory Area).

HMA (High Memory Area)

В реальном режиме из 24 бит адресной шины процессора 80286 (и 32 бит процессора 80386) доступны только 20 бит. Тем не менее уже упоминалась ошибка 8086, которая при обращении к сегменту 0xFFFF приводила к обращению к первым байтам памяти. Однако если снять блокировку 21 линии, появлялась возможность адресовать память за пределами 0xFFFF. Дополнительный сегмент (размером 64 Кб – 16 байт) был назван HMA (High Memory Area). Драйвер MS DOS himem.sys позволял использовать эту память, и при инструкции DOS=HIGH помещать ядро MS DOS в HMA. Впервые himem.sys появился в Windows 2.1 и служил для разблокировки отключенных по умолчанию линии (A20). Таким образом, в реальном времени появилась возможность использовать HMA.

Защищённый режим

В реальном режиме не было возможности защитить программы (память, выполнение прерываний и т.д.) от других программ, поэтому был разработан защищённый режим. Впервые он появился в 80286 процессоре, но полноценно был реализован был уже в 80386

Защищённый режим 80286

Процессор 80286 мог работать в двух режимах: реальном, который был полностью аналогичен режиму работы процессора 8086, и защищённом. Если в реальном режиме всё ПО, написанное для 8086, могло выполняться без изменений, то в защищённом появился полноценный механизм доступа к 16 Мбайт памяти.

В защищённом режиме 80286 впервые появились кольца защиты (от 0 до 3).

Физический адрес в 80286 процессоре формировался следующим образом. В сегментных регистрах хранился селектор, содержащий индекс дескриптора в таблице дескрипторов (13 бит), 1 бит, определяющий, к какой таблице дескрипторов будет производиться обращение (к локальной или к глобальной) и 2 бита запрашиваемого уровня привилегий. Далее происходило обращение к соответствующей таблице дескрипторов и соответствующему дескриптору, который содержал начальный, 24-битный, адрес сегмента, размер сегмента и права доступа. После этого необходимый физический адрес вычислялся путём сложения адреса сегмента со смещением, хранящемся в 16-разрядном указательном регистре.

Фактически защищённый режим 80286 использовал значение сегмента в качестве индекса в таблице дескрипторов. Более того, после перехода в защищённый режим процессора 80286 вернуться в реальный режим можно было только сбросом. В реальном режиме код и прерывания 8086 не могли выполняться, что породило среди разработчиков дилемму: либо использовать преимущества защищённого режима, но отказаться от совместимости со старым ПО, либо поддерживать совместимость, но работать в реальном режиме.

В защищённом режиме 80286 работали ОС Windows 2.0, OS/2.

Реализация 80286 не позволила Windows выполнять одновременно несколько программ MS DOS и стала одной из причиной раздора между Microsoft и IBM, которая настаивала на совместимости OS/2 процессора 80286.

Виртуальная память процессора 80286

Несмотря на возможность адресовать до 16 Мб, компьютеры редко оснащались более чем 2-4 Мб оперативной памяти. В связи с этим появилась идея подкачки, или свопинга. В то время как

программа могла адресовать до 16 Мб, в распоряжении было гораздо меньше памяти. Однако благодаря тому, что какие-то сегменты могли быть выгружены на диск, а при необходимости снова загружены в оперативную память, появилась возможность использовать для программы больше памяти, чем могла предоставить RAM. Подобным образом механизм виртуальной памяти был реализован в OS/2 (версии 1.0 – 1.3).

Фактически этот механизм на 80286 не был эффективен, и полноценно реализовать его удалось уже на 80386.

Защищенный режим 80386

Полноценный защищённый режим был реализован в Intel 80386. В отличие от 80286, процессор был 32-битным и обладал также 32-битной адресной шиной, что позволяло адресовать до 4 Гигабайт памяти и, более того, отказаться от идеи сегмент:смещение. 32-битные регистры (как EIP, так и общего назначения) позволяли адресовать память до 4 Гбайт.

Плоская модель памяти

Способ бессегментной адресации, когда регистр непосредственно указывает на область памяти, называется плоской моделью памяти (flat model). Такой способ адресации используется, например, в Linux. Впрочем, такой механизм не всегда может оказаться удобным. Так, в разработке виртуализации VMware разработчики столкнулись с проблемами при реализации 64-битного процессора от AMD. Из-за отказа от сегментации памяти не удалось сразу реализовать механизм работы виртуальной машины. Именно сегментация памяти использовалась для различия кода монитора и гостя.

Страничная адресация памяти

В Intel 80386 появилась страничная адресация памяти. Теперь память, к которой обращался процессор, могла не соответствовать физической памяти. Это позволило полноценным образом реализовать как защиту приложений, так и работу с виртуальной памятью.

Виртуальный режим 8086

В 80286 было два режима: защищённый и реальный. В 80386 появился новый режим, который позволил писать приложения в защищённом режиме, но исполнять ПО, написанное для 8086. Новый режим, получивший название «режим виртуального 8086», стал первой ласточкой технологий «аппаратной виртуализации».

Программа, исполняемая в таком режиме, имеет ограничение в объёме памяти, как и в 8086. Тем не менее благодаря страничному отображению на самом деле эта память может находиться в произвольном месте памяти, и фактически две программы, исполняемые в виртуальном режиме, могут адресовывать разные области памяти, чего не получилось бы в реальном режиме.

Программа выполняется с низким приоритетом, в кольце 3 – благодаря этому прерывания не должны были выполняться, однако фактически они выполнялись. Вызов прерывания создавал исключение, при котором управление передавалось операционной системе, а она, получив данные, какое прерывание пыталась выполнить 8086-программа, могла, соответственно, выполнить это или иное прерывание либо совершить эмуляцию и т.д.

В виртуальном режиме исполняется MS DOS при использовании драйвера EMM386, именно в виртуальном режиме запускались приложения MS DOS в операционных системах Windows 3.11 и Windows 95 (тот самый «DOS в оконном режиме»).

DOS в защищённом режиме, EMM386

Защищённый режим в MS DOS использовался при запуске драйвера emm386.sys (emm386.exe) или qemm.sys. Фактически защищённый режим использовался не для защиты программ, по крайней мере в MS DOS, а для возможности доступа к дополнительной памяти.

EMM386 переводил процессор в защищённый режим, запуская ядро MS DOS в виртуальном режиме и транслируя ему вызовы от других задач MS DOS.

Полноценная реализация защищённого режима в MS DOS, DPMI

EMM386 – это не полноценный защищённый режим в MS DOS: MS DOS по-прежнему выполняла задачи, написанные для 8086, поэтому для защищенного режима был разработан интерфейс DPMI. Изначально DPMI (DOS Protected Mode Interface) был разработан для Windows 3.0 и применяется при запуске DOS-программ. В этом случае сервером DPMI будет соответствующая реализация DPMI в Windows.

Ещё одна реализация DPMI – DOS4/GW, ставшая самым популярным так называемым 32-битным расширителем DOS. Расширитель поставлялся в составе компилятора Watcom C и часто использовался в компьютерных играх (одна из известных DOOM). Другие производители компиляторов также поставляли «расширители DOS».

Защищённый режим в LINUX

Ядро Linux изначально разрабатывалось для процессора 80386, поэтому проблем с совместимостью, которые возникли у MS DOS и Windows, у него не было.

Ядро Linux использует плоскую модель памяти. Фактически Linux работает с виртуальной памятью, то есть приложения в общем случае не знают, с какой областью памяти они работают, находится ли она в RAM или на диске. Приложениям доступна также расширенная память, которая находится в единственном экземпляре в RAM, но каждым приложением воспринимается как отдельная память.

Подкачка виртуальной памяти реализуется через специальный раздел диска swap.

Для просмотра информации о памяти можно использовать команду:

```
sudo ps aux --sort %mem
```

Показываемые значения:

- RSS (Resident Set Size) – показывает, сколько физической памяти RAM использует процесс; не включает память свопа и память из разделяемых библиотек (shared library), но включает стек и кучу (heap – динамически распределяемую память приложения);
- VSZ (Virtual Set Size) – показывает, сколько виртуальной памяти использует процесс; включает физическую память и swap + shared память.

Итак, если процесс А имеет 500K кода и задействует 2500K шаред библиотек, имеет 200K стека и кучи, из которых 100K в памяти (остальное в свопе), и имеет 1000K, загруженных из шаред-библиотек, и 400K собственного кода, то:

- RSS: 400K + 1000K + 100K = 1500K
- VSZ: 500K + 2500K + 200K = 3200K

Однако часть памяти расшарена, многие процессы используют её, и если сложить RSS всех процессов, получится, что памяти больше, чем имеется в системе. Поэтому существует параметр PSS (proportional set size). Он показывает разделяемую память пропорционально количеству запущенных процессов. Так, если бы процессов, использующих одну и ту же разделяемую память, было два, то показатель был бы следующим:

- PSS: 400K + (1000K/2) + 100K = 400K + 500K + 100K = 1000K

Для запуска понадобится установить программу smem.

В Ubuntu это:

```
sudo apt-get install smem
```

Вывод smem похож на вывод ps aux. Например, просмотреть процент памяти, используемый пользователем:

```
sudo smem -u -p
```

В режиме X11 Server позволяет строить графики:

```
sudo smem --pie name -c "pss"
```

Практическое задание

Для работы понадобится установленная система GNU/Linux.

1. В файловой системе /proc найти информацию об использовании памяти любой программы. Например, cat /proc/7479/status.
2. Узнать, насколько используется swap. Открыть /proc/meminfo и вычесть из значения SwapUsed значение SwapCached.
3. Отфильтровать несколько программ в ps aux, оценить использование памяти.
4. * Сделать то же самое через smem.
5. * Построить график использования памяти через smem.

Примечание. Задания со звёздочкой предназначены для тех, кому недостаточно заданий 1-3 и требуются более сложные задачи.

Дополнительные материалы

1. Кольца, уровни защиты https://habrahabr.ru/company/smart_soft/blog/184174/
2. Сегментация памяти в защищённом режиме http://e-zine.excode.ru/online/3/syscall.html
3. Защищенный режим процессоров Intel 80286/80386/80486. http://palien.narod.ru/Documents/Programms/ZashRej/index.htm

4. Как Linux работает с памятьюю https://habrahabr.ru/company/yandex/blog/250753/
5. Как Linux работает с памятью http://citforum.ru/operating_systems/articles/linuxmem.shtml
6. http://www.rootfront.com/article/9889151/2013-10-22/linux-terminal-proverit-pri-pomoschi-smem_-kto-ispolzuet-vsju-pamjat
7. http://rus-linux.net/MyLDP/sys-conf/memory.html
8. http://stackoverflow.com/questions/7880784/what-is-rss-and-vsz-in-linux-memory-management
9. 8086 в схемах http://www.parl.clemson.edu/~wjones/371/gowdy/documents/materials/8086_OVERVIEW/
10. http://pandia.ru/text/78/031/11940.php

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. https://ru.wikipedia.org/wiki/Битовый_сдвиг
2. https://ru.wikipedia.org/wiki/Сегментная_адресация_памяти
3. https://ru.wikipedia.org/wiki/LIFO
4. https://ru.wikipedia.org/wiki/Реальный_режим
5. https://ru.wikipedia.org/wiki/X86-64
6. https://ru.wikipedia.org/wiki/X86-64
7. https://ru.wikipedia.org/wiki/80286
8. https://ru.wikipedia.org/wiki/80386