сидя на голом железе

крис касперски, ака мыщъх, ака souriz, ака nezumi, aka elraton, ака толстый бобер, no-email

> **в программировании голого железа есть какое-то непередаваемое словами очарование в котором мало практической пользы, но зато огромное эстетическое удовлетворение и наслаждение. сегодня мы спустимся на самый низкий уровень, который только возможно достигнуть на IBM PC! держитесь, мы будем программировать не только без операционной системы, но даже без BIOS'а!**

введение

На прикладном уровне обитать неинтересно. Здесь все приходится делать через готовые интерфейсы (типа win32 API и иже с ними), громоздящиеся своими иерархическими слоями друг на друга. С этой точки зрения, программирования под x86 мало чем отличается от той же Альфы. Мы упираемся в операционную систему, становясь ее пленниками, заключенными в тесную клетку со спертым воздухом.

Без оси жизнь становится намного более захватывающей и интересной. Можно напрямую обращаться ко всем портам ввода-вывода (и знать, что это реальные порты, а не какие-то там виртуальные), выполнять все привилегированные инструкции, переходить в защищенный режим и возвращаться обратно. В общем, делать все, что заблагорассудиться. Вот только…

…основное компьютерное оборудование (и, в первую очередь, чипсет) на этом уровне нам неподвластно. Точнее, подвластно, но не совсем. Конфигурирование и настройка чипсета осуществляется на стадии начальной инициализации компьютера во время загрузки BIOS, подготавливающей его к работе. Некоторые параметры могут быть изменены позднее как через порты ввода/вывода, так и через саму BIOS, но основной фундамент остается непоколебимым. А жаль… ведь далеко не всякая версия BIOS использует возможности аппаратуры на 100%, к тому же лишает нас радости живого общения с железом, подавая его на стол готовеньким. А если нам хочется отведать мяса с кровью?! Тогда необходимо перепрограммировать BIOS, а точнее небольшую его часть, называемую boot-блоком и наиболее приближенную к аппаратуре.

Перепрограммирование boot-блока открывает огромные возможности для трюкачества, позволяя раскрыть свой творческий потенциал и показать на что ты способен. Да что там говорить! Это по-настоящему сложно, а значит, _реально_ круто!

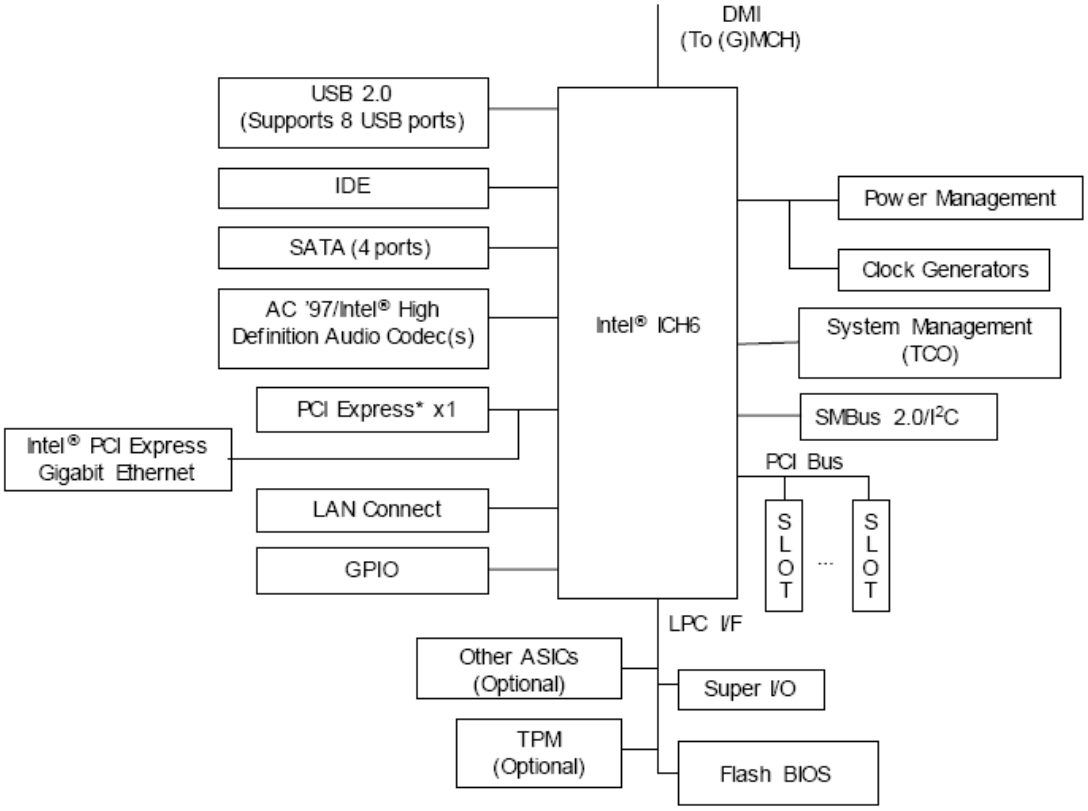
исходные реагенты или что нам понадобиться

Прежде всего нам понадобится **материнская плата**, которую не жалко потерять (в том смысле, что ее смерть не станет трагедией). Еще нужен **программатор** (поскольку не все матери дают прошивать boot-блок, а за пределами boot-блоком жизнь уже становится не такой интересной), который легко купить на радио рынке; любая **программа для редактирования BIOS'а** (обычно идущая на диске, прилагаемом к материнской плате или лежащая на сайте производителя); **прошивка BIOS'а** для изучения и подражания (скопированная из самого BIOS или скаченная с сайта); **транслятор ассемблера**, умеющий генерировать двоичные файлы (FASM, NASM) и, наконец, **документация** на чипсет (Intel и AMD раздают ее бесплатно, остальные производители — только своим партнерам, зачастую под подписку о неразглашении, как будто там есть что разглашать).

ликбез или как все это работает

После "холодной" перезагрузки или включения питания, процессор, находящийся в реальном режиме, передает управление по адресу **F000h:FFF0h**, где находится точка входа в BIOS. В древних IBM AT микросхема постоянной памяти физически "висела" на процессорной шине, непосредственно отображаясь на 64-Кбайтный регион памяти от F000:0000 до F000:FFFF. Современные прошивки в этот объем уже не вмещаются и занимают порядка 512 Кбайт (да и то в упакованном виде), что составляет половину адресного пространства реального режима. Поэтому, в память непосредственно отображаются лишь 64 Кбайт прошивки (порядка 4 Кбайт из которых составляет boot-блок), а остальные части прошивка должна уметь считывать из микросхемы Flash-BIOS'а самостоятельно, обращаясь к специальному контроллеру, как

правило, "вживленному" в южный мост чипсета и соединенному с BIOS'ом по LPC или ISA шине.



Листинг 1 Flash-BIOS, подключенный к южному мосту чипсета Intel 915 через шину LPC

В тот момент, когда BIOS получает управление, практически все имеющееся оборудование к работе еще не готово. Нет даже оперативной памяти, поскольку DRAM-контроллер не настроен и не инициализирован. Короче, как дальше жить! Поэтому, первым делом boot-блок проводит первичную инициализацию важнейших узлов, после чего считывает основной код BIOS'а и распаковывает его в оперативную память. На этом работа boot-блока закачивается и всю дальнейшую инициализацию системы выполняет распакованный им код.

Помимо инициализации, BIOS так же управляет оборудованием (например, выключает жесткие диски по прошествии определенного времени, следит за показанием датчиков напряжения и температуры, регулируя частоту процессора и оперативной памяти), а так же предоставляет в распоряжение программиста обширную библиотеку функций, абстрагирующую его от конкретного железа и доступную через вектора прерываний. В частности, за дисковую подсистему отвечает прерывание 13h, но вернемся к нашим баранам, то есть к голому железу.

Обычно, boot-блок располагается в последних 4 Кбайтах файла прошивки, а по смещению 10h от его конца находится точка входа в BIOS, представляющая собой jmp на подлинную точку входа (см. листинги 1, 2). Остальной код прошивки, как правило, упакован, поэтому непосредственно внедряться можно только в последние 4 Кбайта, т. е. в промежуток между F000:EFFF и F000:FFFF.

Внедряемый код должен быть либо полностью перемещаемым (т. е. сохранять свою работоспособность независимо от базового адреса загрузки), либо в начало ассемблерного листинга необходимо воткнуть директиву "ORG 0XXXXh", где 0XXXXh равно разнице конца boot-блока (лежащего по смещению FFFFh) и размеру внедряемого кода.

```
0007FFF0:  EA 5B E0 00-F0 2A 4D 52-42 2A 02 00-00 00 60 FF  ъ[р Ё*MRB*☻    `
```

Листинг 2 последние 10h байт прошивки материнской платы EPOX EP-5EPA+

```
0007FFF0: EA5BE000F0                jmp             0F000:0E05B
```

Листинг 3 …и их дизассемблерный код

Последние два байта boot-блока занимает контрольная сумма, рассчитываемая по следующему алгоритму (сохранившемуся еще со времен первых BIOS'ов): мы просто складываем все байты друг с другом и находим остаток от деления на 100h, что в псевдокоде занимается так: sum = (sum + next_byte) & 0xFF. Контрольная сумма всего boot-блока должна равняться нулю, следовательно, последний байт блока равен: (100h – sum) & 0xFF.

работа с шиной PCI

Шина PCI является основной шиной, через которую к процессору подключаются все остальные контроллеры и устройства, поэтому, чтобы научиться программировать голое железо, нам прежде всего необходимо разобраться как программировать саму шину PCI. Это легко. Достаточно выучить всего пару регистров: **CF8h** и **CFCh**.

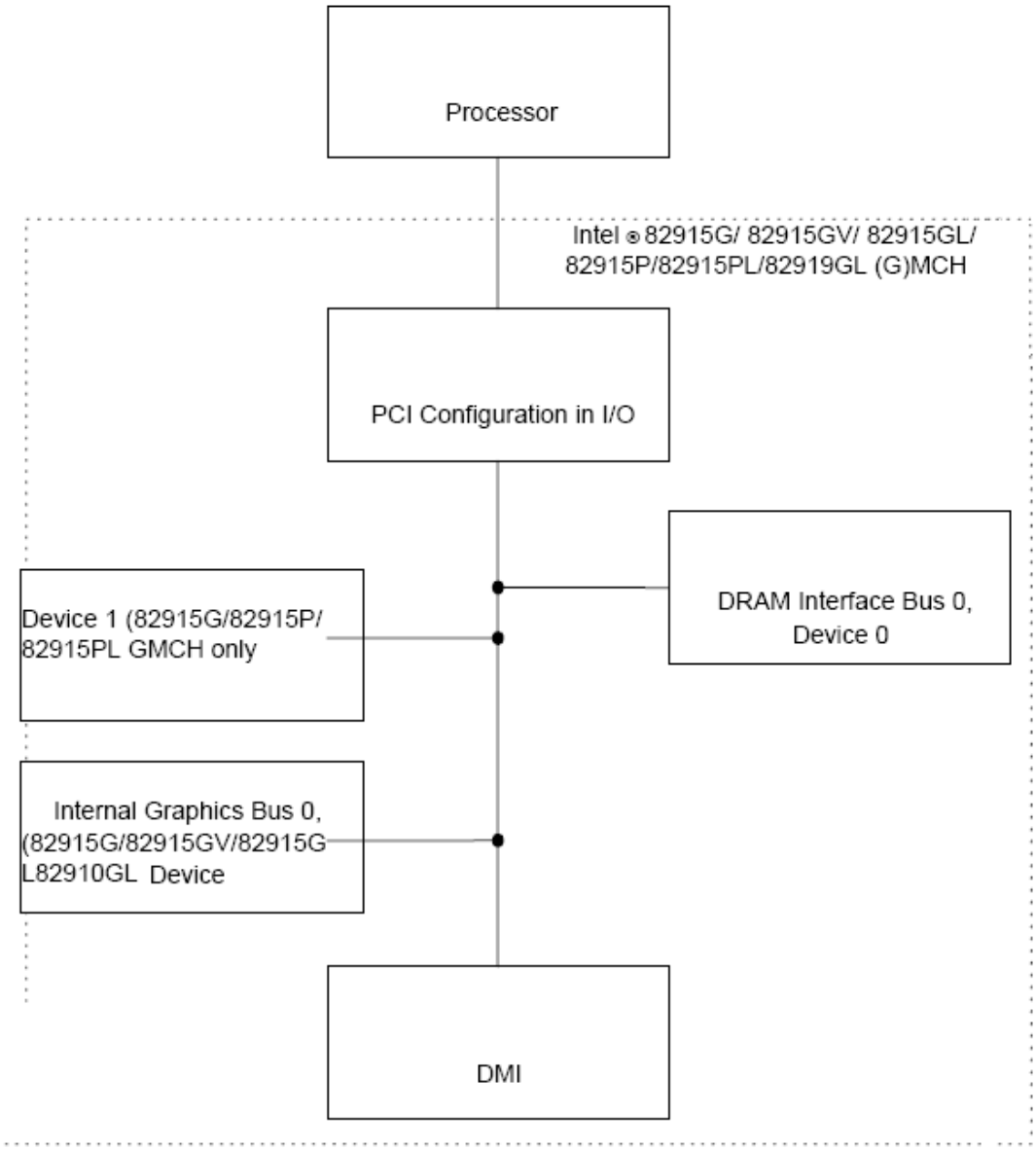


Рисунок 1 серверный мост чипсета, несущий на своем борту важнейшие устройства (такие, например, как контроллер памяти), подключается к процессору через PCI-шину

В порт CF8h заносится адрес регистра с которым мы хотим работать (называемый так же смещением или offset'ом), а через порт CFCh происходит обмен данными, который в зависимости от конструктивных особенностей конфигурируемого контроллера может быть доступен как на запись/чтение, так и только на чтение. Под "регистром" здесь понимается отнюдь не регистр процессора (типа EAX), а регистр контролера. Некоторые регистры

отображаются на порты ввода/вывода (и тогда с ними можно работать командами IN/OUT), некоторые — нет и с ними можно работать только через CF8h/CFCh.

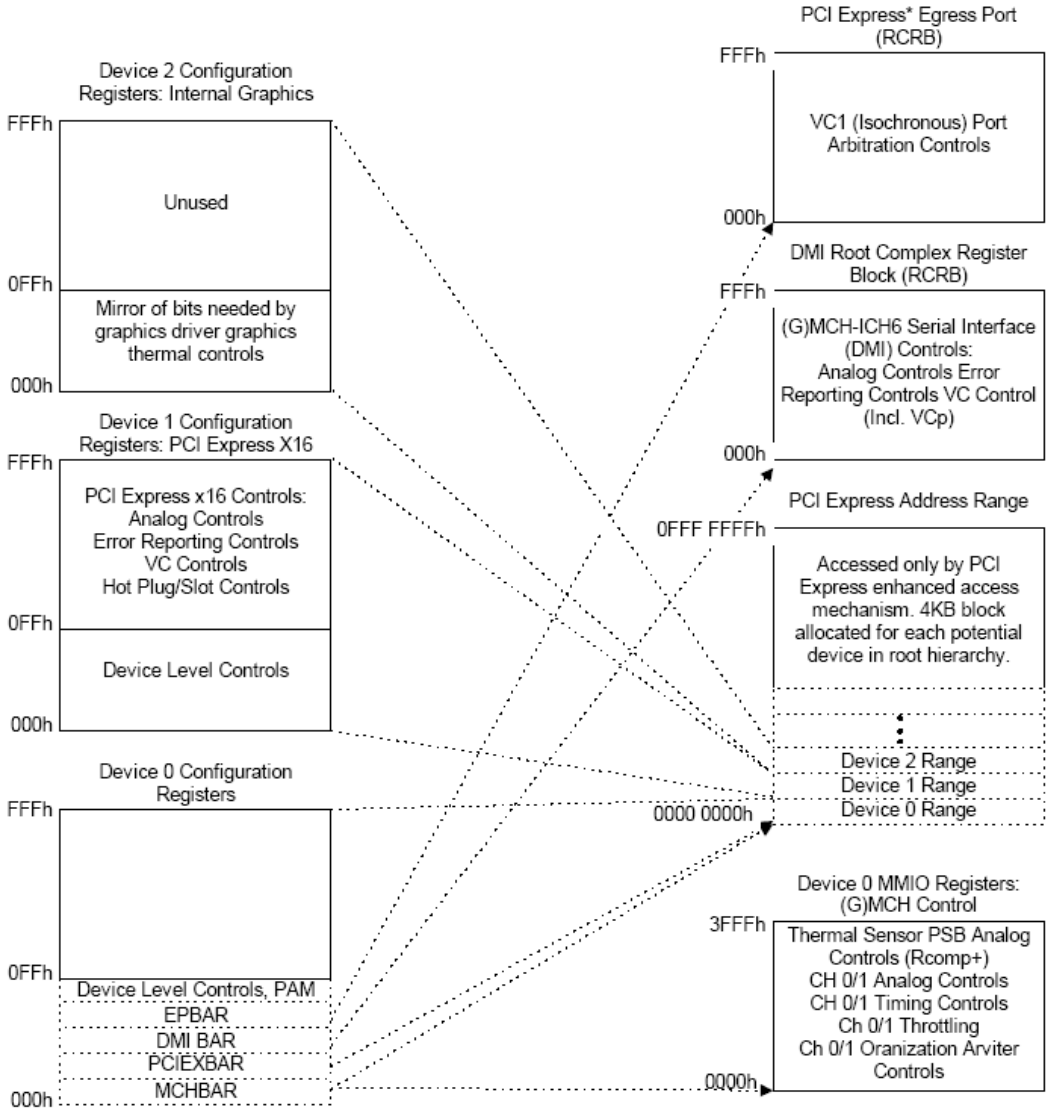


Рисунок 2 регистры аппаратных устройств, отображаемые на адресное пространство и порты ввода/вывода

Большинство регистров представляют собой совокупность управляющих битов, поэтому перед тем как что-то записывать в порт CFCh, мы, как правило, сперва должны прочитать текущее состояние чипсета, взвести/опустить нужные нам биты при помощи операций OR и AND, после чего затолкать обновленный регистр на место (однако, если текущее состояние нас не интересует, выполнять операцию чтения совершенно необязательно).

Описание самих регистров можно найти в документации на северный и южный мосты чипсета. Где-то там будет раздел "PCI Configuration Registers", "Registers Description" или что-то в этом роде.

В частности, регистр 114h северного моста чипсета Intel 915, управляет таймингами DDR-памяти и делает он это с помощью своих битов, комбинация которых задает то или иное состояние DRAM-контроллера. Конкретные значения приведены в таблице 1, которую можно найти на 102 странице оригинального руководства: intel.com/design/chipsets/datashts/301467.htm

биты	доступ	значение по умолчанию	назначение
31:24	–	—	зарезервированы
23:20	R/W	9	величина **tRAS** (она же **DRAM Precharge Delay**, она же

<table>
<tr><td></td><td></td><td></td><td>Active to Precharge Delay, она же Precharge Wait State, она же Row Active Delay, она же Row Precharge Delay) устанавливает минимальный промежуток времени между открытием/закрытием одной DRAM-страницы.
значения от 0 до 3 зарезервированы,
значения от 4 до 15 — время в тактах;</td></tr>
<tr><td>19</td><td>RO</td><td>0</td><td>tRAS MAX – максимальный промежуток времени в течении которого DRAM страница может оставаться открытой и, если контроллер не закроет ее, произойдет Panic Refresh (экстренное обновление), сопровождаемое закрытием всех страниц во всех банках.
в данном чипсете этот регистр доступен только на чтение, то есть управление tRAS MAX не реализовано и составляет 120 наносекунд</td></tr>
<tr><td>18:10</td><td>—</td><td>—</td><td>зарезервированы</td></tr>
<tr><td>09:08</td><td>R/W</td><td>01b</td><td>Величина tCL, более известная под именем CAS# Latency, задает количество тактов между отправкой DDR-микросхеме команды чтения (не записи!) и сбросом первой порции данных на шину, при этом DRAM-страница должна быть заблаговременно открыта, за что отвечает тайминг tRCD.
<table>
<tr><td>значение регистра</td><td>DDR tCL, такты</td><td>DDR2 tCL, такты</td></tr>
<tr><td>00b</td><td>3</td><td>5</td></tr>
<tr><td>01b</td><td>2,5</td><td>4</td></tr>
<tr><td>10b</td><td>2</td><td>3</td></tr>
<tr><td>11</td><td colspan="2">зарезервированы</td></tr>
</table></td></tr>
<tr><td>07</td><td>—</td><td>—</td><td>зарезервирован</td></tr>
<tr><td>06:04</td><td>R/W</td><td>010b</td><td>Величина tRCD, так же называемая RAS# to CAS# Delay или Active to CMD, определяет время открытия DRAM-страницы, в процессе которого со строки конденсаторов считывается заряд и заносится в буфер статической памяти, локально обрабатывающий все последующие обращения.
<table>
<tr><td>значение регистра</td><td>tRCD, такты</td></tr>
<tr><td>000b</td><td>2</td></tr>
<tr><td>001b</td><td>3</td></tr>
<tr><td>010b</td><td>4</td></tr>
<tr><td>011b</td><td>5</td></tr>
<tr><td>100b</td><td rowspan="4">зарезервированы</td></tr>
<tr><td>101b</td></tr>
<tr><td>110b</td></tr>
<tr><td>111b</td></tr>
</table></td></tr>
<tr><td>3</td><td>—</td><td>—</td><td>зарезервирован</td></tr>
</table>

<table>
<tr><td rowspan="9">2:0</td><td rowspan="9">R/W</td><td rowspan="9">010b</td><td colspan="2">Величина tRP (она же RAS# Precharge Delay, она же Precharge to active) определяет время закрытия DRAM-страницы, в процессе которого происходит возврат данных в банк памяти и его перезарядка. Во время перезаряда банк недоступен, но доступны все остальные банки (большинство DDR-модулей содержит четыре таких банка). Банк закрывается на перезарядку всякий раз, когда происходит обращение к другой странице из этого же самого банка.</td></tr>
<tr><td>значение регистра</td><td>tRP, такты</td></tr>
<tr><td>000b</td><td>2</td></tr>
<tr><td>001b</td><td>3</td></tr>
<tr><td>010b</td><td>4</td></tr>
<tr><td>011b</td><td>5</td></tr>
<tr><td>100b</td><td rowspan="4">зарезервированы</td></tr>
<tr><td>101b</td></tr>
<tr><td>110b</td></tr>
<tr><td>111b</td></tr>
</table>

Таблица 1 управление таймигами памяти через регистр 114h северного моста чипсета Intel 915 (на других чипсетах номер регистра и назначение бит с вероятностью, близкой к единице, будут совсем другими, поэтому данная таблица приводится _только_ как пример)

Таким образом, чтобы настроить память на максимальную производительность, необходимо занести в регистр 114h число 10000000000001000000000b (или 400200h в шестнадцатеричном виде), что на языке ассемблера делается так:

```
mov eax, 114h          ; регистр чипеста, управляющий DRAM-контроллером
mov dx, 0CF8h          ; PCI-порт (адрес регистра)
out dx, eax            ; выбираем регистр

mov     dx, 0CFCh      ; PCI-порт (данные)
in      eax, dx        ; читаем содержимое регистра 114h
or      eax, 400200h   ; конфигурируем таймиги памяти
out     dx, eax        ; записываем регистр чипсета
```

Листинг 4 ассемблерный код, устанавливающий таймиги DDR-памяти на максимальную производительность, игнорируя информацию записанную в SPD (только для чипсета Intel 915! обладатели других чипсетов должны заменить подчеркнутые константны в соответствии со своей документацией)

заключение

Общаться с оборудованием на "железном" уровне безумно интересно, но чрезвычайно утомительно и сложно. Прежде чем контроллер оперативной памяти "заведется" предстоит проделать немало работы и тщательно прочитать порядка тысячи страниц документации, поскольку любая пропущенная мелочь может пустить все кувырком. Положение усугубляется полным отсутствием отладочных средств, что дисциплинирует и учит искать ошибки "глазами".

Совет: прежде чем писать свои мини-BIOS, выводящий на экран зеленый травянистый семилистник под мелодию "семь сорок", раздающуюся из спикера, скачайте готовую прошивку и дизассемблируйте boot-блок, разобравшись как он работает. Мы увидим много обращений к портам, часть из которых удается расшифровать с помощью документации на чипсет, часть — обратившись к описанию остальных контроллеров и микросхем, установленных на плате.

```
seg010:DB6D 55              push    bp
seg010:DB6E 8B EC           mov     bp, sp
seg010:DB70 83 EC 08        sub     sp, 8
seg010:DB73 C6 45 F8 41     mov     byte ptr [di-8], 'A'
seg010:DB77 C6 45 F9 4D     mov     byte ptr [di-7], 'M'
seg010:DB7B C6 45 FA 49     mov     byte ptr [di-6], 'I'
seg010:DB7F C6 45 FB 42     mov     byte ptr [di-5], 'B'
seg010:DB83 C6 45 FC 49     mov     byte ptr [di-4], 'I'
seg010:DB87 C6 45 FD 4F     mov     byte ptr [di-3], 'O'
seg010:DB8B C6 45 FE 53     mov     byte ptr [di-2], 'S'
seg010:DB8F C6 45 FF 43     mov     byte ptr [di-1], 'C'
```

```
seg010:DB93 53              push    bx
seg010:DB94 56              push    si
seg010:DB95 33 F6           xor     si, si
seg010:DB97 8B 45 08        mov     ax, [di+8]
seg010:DB9A 8B 50 20        mov     dx, [bx+si+20h]
seg010:DB9D 85 D2           test    dx, dx
seg010:DB9F 74 3A           jz      short loc_A9BDB
```

Листинг 5 фрагмент прошивки AMI-BIOS

В конечном счете мы создадим свой boot-блок, ассемблируем и, воткнув в программатор FLASH-BIOS, зальем туда свое творение (предварительно сохранив оригинальную прошивку). Не стоит надеяться, что материнская плата "проглотит" его с первого раза. Будет только черный экран, не реагирующий на клавиатуру и все. Даже курсора не будет. Попробуй угадай в каком месте сидит ошибка!!! Но ведь только так из юноши программисты становятся настоящими мужчинами. В смысле хакерами. После недели-другой непрекращающихся мытарств мы, наверное, сможем оценить какого приходилось нашим предкам, дырявящим перфокарты и совсем незнакомым с понятием интерактивной отладки.

>>> врезка порядок инициализации оборудования

Порядок инициализации оборудования не высечен на камне и допускает довольно большие вольности, однако, определенные традиции и нормы приличия все-таки нужно соблюдать.

Материнские платы от EPOX выгодно отличаются тем, что имеют 2х разрядный сегментный индикатор, отображающий ход загрузки, что облегчает диагностику поломки (если таковая есть) и упрощает дизассемблирование BIOS, поскольку… код, отображаемый на индикаторе, в листинге присутствует в виде константы, расшифровку значения которой можно найти в руководстве в приложении E. Лучших комментарий к листингу, пожалуй, и не придумаешь!!!

На остальных материнских платах порядок инициализации оборудования не сильно отличается от EPOX'а и все происходит приблизительно следующим образом:

- тест CMOS на чтение/запись;
- отключение теневой RAM;
- инициализация базовых регистров чипсета;
- чтение информации из SPD и инициализация DRAM-контроллера;
- распаковка сжатого BIOS в оперативную память;
- копирование BIOS в теневую RAM в сегменты E000h и F000h;
- инициализация интерфейса 8042 (интегрированного в южный мост);
- запуск самотестирования 8042;
- детектирование микросхемы FLASH-ROM;
- загрузка процедуры "прожига" FLASH в сегмент F000h (для поддержки DMI & ESCD);
- установка всех регистров чипсета в значения по умолчанию (берутся из BIOS);
- детектирование типа ЦП;
- инициализация таблицы векторов прерываний;
- считывание установок CMOS в стек BIOS'а;
- построение карты ресурсов для PCI- и P&P-устройств;
- инициализация тактового генератора;
- сканирование всех устройств, подключенных к PCI-шине;
- назначение устройствам IRQ и портов ввода/вывода;
- поиск видео-карты и отображение VGA BIOS на сегмент C000h;
- инициализация буфера клавиатуры для вектора INT 09h;
- инициализация внутренних MTRP-регистров ЦП для отображения памяти 0-640 Кб;
- инициализация APIC-контроллера;
- установка регистров чипсета в соответствии с настройками CMOS;
- инициализация интегрированного IDE-контроллера;
- измерение тактовой частоты ЦП;
- вызов VIDEO-BIOS;
- вывод на экран информации о типе ЦП, тактовой частоте, логотипа BIOS и т. д.;
- сброс клавиатуры;
- тест битовой маски контроллера прерываний 8259 для канала 1;
- тест битовой маски контроллера прерываний 8259 для канала 2;

- первичная инициализация шины USB;
- тестирование и очистка памяти;
- готовность выйти в BIOS Setup по нажатию <DEL>;
- инициализация PS/2 мыши;
- инициализация и выделение ресурсов интегрированным COM и LPT-портам;
- инициализация привода гибких дисков;
- детектирование и инициализация IDE-приводов (HDD, LS120, ZIP, CDROM);
- детектирование и инициализация не интегрированных COM/LPT-портов;
- перепрограммирование шрифтов для вывода EPA-логотипа в текстовом режиме;
- вывод EPA-логотипа;
- вызов hock-процедуры менеджера энергопитания;
- восстановление шрифтов, используемых для вывода EPA-логотипа;
- запрос пароля (если загрузка компьютера защищена паролем);
- запись всех данных из стека BIOS обратно в CMOS;
- инициализация загрузочных устройств P&P;
- финальная инициализация шины USB;
- инициализация APCI-таблицы на вершине памяти;
- поиск ISA-устройств (если есть эта рухлядь) и вызов их BIOS'ов;
- финальное распределение между ISA и PCI устройствами;
- финальная инициализация чипсета;
- финальная инициализация менеджера энергопитания;
- очистка экрана и вывод таблицы с информацией о конфигурации всех устройств;
- инициализация клавиатурных индикаторов и установка скорости автоповтора;
- построение MP-таблицы;
- построение и обновление ESCD-таблицы;
- установка века в CMOS;
- загрузка даты/времени в область данных BIOS;
- построение таблицы маршрутизации аппаратных прерываний MSIRQ;
- попытка загрузки с загрузочного носителя (вызов прерывания INT 19h);
- на этом BIOS прекращает свою работу и передает управление boot-сектору;

>>> врезка ссылки по теме

- **BIOSmods:**
  - портал, посвященный BIOS'у и его доработке (на английском языке): http://www.biosmods.com/;
- **ROM:**
  - статьи по прошивке и доработке прошивок, уникальный инструментарий (на русском языке): http://www.rom.by;
- **Часто задаваемые вопросы о BIOS:**
  - довольно устаревшее, но все-таки полезное FAQ по BIOS'ам (на русском языке): http://www.ixbt.com/mainboard/faq/biosfaq.shtml;
- **The Official Website of Pinczakko:**
  - сайт улетного индонезийского хакера, исследовавшего кучу BIOS'ов вдоль и поперек и вытворяющий с ними такое, что другим даже не снилось (на английском и индонезийском языках): http://www.geocities.com/mamanzip/;
- **Modification of GigaByte GA-586HX BIOS rev 2.9 for support of HDD above 32 GiB:**
  - интересная статья о модификации BIOS, название которой говорит само за себя (на английском языке): http://www.ryston.cz/petr/bios/ga586hx_mod.html;
- **Award BIOS Reverse Engineering:**
  - статья известнейшего хакера BIOS'a Mappatutu Salihun Darmawan по внедрению своего кода в BIOS (на английском языке):http://www.codebreakers-journal.com/include/getdoc.php?id=83&article=38&mode=pdf;
- **Award BIOS Code Injection:**
  - новые идеи по внедрению своего кода в BIOS (на английском языке): http://www.codebreakers-journal.com/include/getdoc.php?id=127&article=58&mode=pdf;
- **AWARD BIOS Source:**
  - исходные тексты пары устаревших прошивок со скупыми комментариями. помогают понять общие принципы функционирования BIOS'a и разобраться в

некоторых тонких местах, неочевидных при дизассемблировании: (на языке ассемблера): http://miscellaneous.newmail.ru/;

- **Редактируем BIOS (Award Modular v.4.51):**
  - описание формата BIOS'a для его ручной распаковки (на русском языке): http://www.winsov.ru/bios002.php;
- **AMD 762 Chipset Tweaking (MP/MPX) Guide:**
  - статья по редактированию регистров чипсета (на русском языке): http://www.tweakfactor.com/articles/tweaks/amd762/1.html;
- **Using Oda's WPCREDIT On VIA Motherboards:**
  - разгон системы путем редактирования регистров чипеста (на английском языке): http://www.overclockers.com/tips105/index.asp;
- **H.Oda's Home Page:**
  - утилита для модификации регистров чипеста на лету: http://www.h-oda.com/;
- **Award BIOS Editor:**
  - редактор Award BIOS'ов, распространяемый с исходными текстами на бесплатной основе: http://awdbedit.sourceforge.net/;
- **AWDhack v1.3:**
  - утилита для автоматизированного внедрения своего кода в BIOS http://webzoom.freewebs.com/tmod/Awdhack.zip;