Технически университет – София



**Катедра „Компютърни системи”**

**Специалност „Компютърно и софтуерно инженерство”, степен бакалавър**

**Курсов проект по КА**

**На тема:**

**„х86 Процесори”**

**Изготвили: Александър Цветанов и Николай Чобанов**

**факултетен номер: 121214064, 121214189**

**София 2016**

**Съдържание**

1. [Въведение 3](file:///C:\Users\alexander\Downloads\40_PlamenKonstantinovKoynov-Copy.doc#_Toc439342996)

2. История

2.1. **КОМПЮТЪРНА СИСТЕМА IBM/ХТ I8086**

**2.2. КОМПЮТЪРНА СИСТЕМА IBM/АТ 80386 6**

**2.3. КОМПЮТЪРНА СИСТЕМА IBM/АТ 80486 9**

**2.4. КОМПЮТЪРНА СИСТЕМА С ПРОЦЕСОР 13**

**2.5 КОМПЮТЪРНА СИСТЕМА С ПРОЦЕСОР 16**

**3. Реален режим 20**

**3.1. История**

**3.2. Въведение**

**3.3. Технология**

**3.4. Регистри 21**

**3.5. Типове данни**

**3.6. Методи за адресиране 22**

**3.7. Входно-изходно адресно пространство 23**

**3.8. Инструкции**

**3.9. Проблеми 24**

**3.10. Приложения**

**4. Защитен режим 25**

**4.1. История**

**4.2. Въведение 25**

**4.3. Технология**

**4.4. Регистри**

**4.5. Типове данни 26**

**4.6. Методи за адресиране**

**4.7. Входно-изходно адресно пространство 28**

**4.8. Инструкции 29**

**4.9. Проблеми 30**

**4.10. Приложения 31**

**4.11. Процесори поддържащи защитен режим**

**5. Инструкции за числа с плаваща запетая (x87 инструкции)**

**6. ММХ 32**

**7. 3DNow!**

**8. SSE, SSE2, SSE3 34**

**9. х86-64**

**10. Симулация на х86 с Multi2sim 35**

**10.1. Симулация на х86**

**10.2. Обобщение на х86 статистика 37**

**10.3. Съставяне и симулране на Source Code 38**

**10.4. Конвейрни процесори 42**

**10.5. CISC инструкции - декодиране 43**

**11. Източници 44**

1. **Въведение.**

x86 е общото наименование на компютърна архитектура за микропроцесори, първоначално разработени и произвеждани от Intel. Първите процесори от тази фамилия са обозначавани с номера, завършващи на 86: 8086, 80186, 80286, 80386 (впоследствие i386) и 486 (i486). Водена от маркетингови съображения и ограничена от невъзможността да защити търговска марка върху число, Intel решава да именува следващите процесори от тази фамилия с имена: Pentium, Core. В крайна сметка обаче, името x86 остава като име на цялата фамилия.

Архитектурата на два пъти е била разширявана към по-голям размер на машинната дума. Първите процесори от тази фамилия са 16-битови. През 1985 г. Intel пуска 32-битовия процесор Intel 386. Обикновено обозначението x86 се използва за 32-битовата архитектура, като освен него се използва и IA-32 (на английски: Intel Architecture 32-bit). През 2003 г. AMD пусна на пазара процесора Athlon 64, който разшири архитектурата до 64 бита. 64-битовата x86 архитектура се обозначава с x86-64, като освен него се използват и други обозначения: AMD64 (от AMD), EM64T или IA-32e (от Intel) и x64 (от Microsoft).

Една от основните черти на x86 архитектурата е пълната софтуерна оперативна съвместимост с по-стари модели. Това означава, че всеки нов член на фамилията може да изпълнява програмите, създадени за неговите предшественици. Тази съвместимост е една от предпоставките за огромния успех на x86 архитектурата.

1. **История.**

x86 архитектурата започва своето съществуване с разработката на процесора Intel 8086 от Intel през 1978 г. 8086 е развитие на техния по-ранен успешен 8-битов процесор 8080. Широкото разпространение на архитектурата започва през 1981, когато IBM избира Intel 8088 (процесор, който е идентичен като програмен модел с 8086) за „сърцето“ на своя персонален компютър IBM PC.

И други компании започват да произвеждат процесори, които са съвместими с 8086/8088. Повечето от тези компании рано или късно се оттеглят от тази област поради огромната конкуренция от страна на Intel, като в момента единствените останали производители са Intel, AMD и VIA (VIA е собственик на бившите IDT и Cyrix).

* 1. **КОМПЮТЪРНА СИСТЕМА IBM/ХТ I8086**

**Основни функционални блокове на дънната платка:**

\***Микропроцесор Intel 8086** – централен процесор на системата;

\***Копроцесор 8087 /на цокъл/**- Математически копроцесор за ускоряване на изчислителните операции.

\***RAM- Памет с произволен достъп, оперативна памет** на компютърната система;

\***ROM-BIOS** - Памет само за четене. Съдържа **BIOS** /базовата входно-изходна система/- програмите за връзката между апаратната част и операционната система на компютъра.

**\*Контролер на шината**;

\***Тактов генератор:** Генериране на системния такт;

\***Контролер за прекъсванията:** Управление на логиката за прекъсвания;

\***DMA-контролер** - DMA означава директен достъп до паметта. Осъществява предаване на данни без директното участие на микропроцесора.

**\*Таймер:** - Системен часовник, брояч. Функционалното му предназначение е генериране на цикъл за опресняване на паметта.

**\*PIO** - паралелен вход / изход.

**ISA** e съкращение от Industry Standard Architecture (стандартна за индустрията архитектура) и е шинната архитектура.

Към ISA шината се свързват периферни устройства чрез серийни и паралелни портове.

Оригиналната 8-битова версия работи на **8,33 MHz** стандартна честота за 8/16-битовите версии на ISA шината с цел обратна съвместимост.

Теоретичната максимална скорост на прехвърляне на данните:

**8,33MHz х 1 байта (8 бита) / 2 такта за прехвърляне = 4,165МВ/сек.**

Ефективната пропускателна способност е доста по-ниска - обикновено около половината на максимума.

От физическа гледна точка 8-битовият разширителен ISA слот представлява гнездо с жлеб, в който се поставя печатната платка.

Слотът предоставя 8 линии за данни и 20 адресни линии, които му позволяват да адресира до 1MB памет.

**Системен таймер -** съкращение от Programmable Interval Timer. На първите компютри се е използвала микросхема Intel 8253. *По-късно таймерът влиза в състава на чипсета.* Микросхемите представляват триканални програмируеми **броячи-таймери**. Вътрешните броячи на микросхемите имат разрядност 16 бита, но връзката с тях е 8-битова. Броячите се използват за **подаване на прекъсване от таймера (IRQ 0)**, управление **опресняването на дина­мичната RAM памет**, за **генериране на звук**.

**Системните ресурси** се използват от хардуерните устройства за комуникиране по шината. На най-ниско ниво тези ресурси обикновено включват следното: - адреси от паметта; - IRQ линии; - DMA канали; - адреси на входно/изходни портове.

**ПРОЦЕСОРИ**

За процесор 8086 основните характеристика са:

\*тактови честоти: 5MHz; 8MHz; 10MHz;

\*Ширина на шината - 16 бита за информация, 20 бита за адреси;

\*Брой на транзисторите - 29000 в 3 μm;

\*Адресируема памет 1 MВ –

\*Използван в преносими компютри;

\*Assembler, съвместим с 8080;

\*Използвани са сегментни регистри за достъп до повече от 64K информация наведнъж.

Процесорът 8086 е един от първите 16-битови процесори на пазара, който не само има вътрешни 16-битови регистри, но е и с 16-битова шина за данни. Процесорът 8086 е в състояние да изпълнява нов клас софтуер, използвайки 16-битови инструкции.

Процесорите 8086 работят в т.нар. **реален режим**. Такъв тип софтуер обикновено е еднозадачен - в даден момент може да работи само една програма.

**Режим на мултиплексиране на данните и адресите -** При процесорите 8086 и 8088 данните се предават по обща шина за данни и адреси чрез **мултилексиране -** **представлява споделяне (съвместно използване) на капацитета на една предавателна линия между две или повече комуникиращи станции.**

**Стековата памет** е област от ram паметта, която се адресира от указателя на стека, при прекъсване записва съдържанието на вътрешните си регистри в стека.

**Инструкциите** казват на процесора какво да прави с данните: да ги събере, извади или премести.

**Регистрите на процесора са участъци за временно съхраняване**. Те съдържат данни, които чакат да бъдат обработени с дадена инструкция или данни, които са вече обработени.

**Проследяване на инструкции.** - Една инструкция се състои от много стъпки, които трябва да бъдат изпълнени в специфичен ред.

**Индексно адресиране** - адресът на избраната клетка се образува като сума от съдържанието на втория байт на инструкцията и съдържанието на индексен регистър /Регистъра Х/. Този тип адресиране е едно от най-често използваните и позволява да се обработват циклично повече клетки от паметта.

**Относително адресиране** – използва се при инструкциите за условен и безусловен преход. Действителният адрес на избраната клетка от паметта се получава чрез сумиране съдържанието на програмния брояч с операнда в инструкцията / число/.

* 1. **КОМПЮТЪРНА СИСТЕМА IBM/АТ 80386**

**Дънни платки**

**Основни функционални блокове на дънната платка на компютърна система IBM/АТ с процесор 80386.**

**Основните функционални блокове на дънна платка на компютърна система IBM/АТ с процесор 80386 са:**

**< Микропроцесор 80386**. CPU на системата;

**< Копроцесор 80387 -** математически копроцесор за ускоряване на изчислителните операции специално проектиран да работи съвместно с 386;

< **RAM** (Random Access Memory - памет с произволен достъп). Оперативна памет с обем максимално 4 GB. Използват се основно модули SIMM с 30 извода.;

< **Кеш памет** - служи като посредник между процесора и оперативната памет, като осигурява на процесора най-често използваните команди и данни;

< **ROM-BIOS** (Read Only Memory - памет само за четене). Съдържа базовата входно-изходна система (BIOS), програмите за основните системни операции;

< **Контролер на шината**. Декодиране на сигналите на процесора и генериране на сигналите на шината Управлението върху системната шина може да се поема от DMA-контролерът т.е. да се осъществи се директен достъп до паметта.;

< **Тактов генератор –** за генериране на системния такт;

**< Контролери за прекъсванията -**Управление на логиката за прекъсвания /има два контролера за прекъсванията и така се поддържат 16 канала/;

**< DMA-контролери** (DMA - директен достъп до паметта) - Предаване на данни между периферните устройства и основната памет, без директното участие на микропроцесора. АТ има два контролера за директен достъп и така се поддържат 8 канала.;

< **Таймер: 8254**. Системен часовник, брояч и генериране на цикли за опресняване на паметта;

< **Интерфейс за клавиатура;**

**< Часовник за реално време (RTC – Real Time Clock): Motorоla 146818** с резервно батерийно захранване.

При АТ системите с процесор 80286 започва процес на интеграция на някои от компонентите, които се обединяват в чипсети.

**Разширителни шини – MCA и EISA**

МСА(Micro Channel Architecture) и EISA(Extended Industry Standard Architecture) са относително бавни шини, също като ISA.

**Разширителна шина MCA**:

\*32-битова шина за данни**;**

\*32-битова адресна шина вместо 24-битова и оттук възможност за директно адресиране на 4 GByte вместо 16 MByte памет**;**

\*10 MHz тактова честота на шината, асинхронна спрямо CPU такта;

\*20 MByte/s максимална скорост за предаване на данни;

\*Възможност за 16 управляващи шината устройства (Bus Master);

\*принципно различна от ISA, поради което липсва обратна съвместимост с 8-битовите и 16-битовите ISA карти;

\***plug-and-play** (самонастройваща се) още преди съществуването на официалната спецификация Plug and Play.

\***Автоматично конфигуриране чрез софтуер**. Настройката чрез специална Reference дискета, която се доставя с конкретната система, и Option дис­кети, които съпътстват всяка карта, инсталирана в системата.

\*Контролерът на прекъсванията - възможни са 255 хардуерни прекъсвания

**EISA се характезира с:**

\*32-битова шина за данни.

\*32-битова адресна шина вместо 24-битова адресна шина, което позволява адресирането до 4 GByte оперативна памет, вместо само до 16 MByte.

\*Автоматично конфигуриране чрез софтуер, а не чрез DIP-превключватели. \*Възможност различни устройства да поемат управлението на шината (Bus Master).

\*32-битов DMA-цикъл вместо 8- или 16-битов при системите ISA.

\*Директният достъп до паметта (DMA) при системите EISA e подобрен значително. Вградени са 3 нови DMA-режима (А, В и С) и е запазен стандартният ISA-режим: -A. Съкратена фаза на обръщенията към паметта; -В. Съкратена фаза на обръщенията към паметта и на входно-изходните операции; -С. Режим Burst - 33 MByte/s; -Стандартен ISA - съвместим режим - 4 MByte/s. Режимите A и В могат да се използват без изменения с карти ISA, докато режимът Burst може да се използва само с карти EISA.

\*Приоритетите се присвояват по ротационен принцип;

\*Контролерът на прекъсванията се управлява по нива, а не по фронтове, както е при системите ISA.

**Системните ресурси** се използват от хардуерните устройства за комуникиране по шината. На най-ниско ниво тези ресурси обикновено включват следното: - адреси от паметта; - IRQ линии; - DMA канали; - адреси на входно/изходни портове.

**Процес на директен достъп до паметта.**

Пряк достъп до паметта се нарича способът на обмяна на данни, извършван автономно (независимо) от процесора между основната памет и външни устройства. В режим на пряк достъп до паметта се използва специализирано устройство - контролер за пряк достъп до паметта.

Трансферът на данни, които извършва DMA е ограничаван единствено от пропускателната способност на основната шина и шината към външните устройства.

**DMA контролер -** При работа в режим на пряк достъп до паметта контролерът на DMA изпълнява следната последователност от операции: \*приема заявки от за трансфер на данни от външните устройства; \*формира заявка до централния процесор за заемане на системната шина за трансфер на данни; \*приема сигнал, потвърждаващ разрешението за заемане на системната магистрала за пренос на данни. В това състояние микропроцесорът прекъсва връзката със системната шина; \*формира сигнал, съобщаващ на външното устройство за начало на трансфера на данни; \*подава на адресната шина адреса на клетката от паметта, предназначена за обмен; \*генерира сигнали, с които управлява трансфера на данните; \*при завършване на обмена на данни или организира повторно изпълнение на цикъла, или прекъсва режима на пряк достъп до паметта.

**ПРОЦЕСОРИ**

Intel 80386 е създаден през 1985 г. **Основните му предимства са:**

**1/32-битова шина за данни – вътрешна и входно-изходна;**

**2/Увеличена тактова честота -**  започва с 16 MHz и достига 33 MHz.

**3/Подобрена архитектура на паметта**

\*Увеличен брой адресни линии – 32. Това позволява пряко адресиране на 4 GB физическа памет

\*Възможност за работа с 16 TB (терабайта) виртуална памет.

\***Кеш памет -16 KB** (32, 64, 128 до 256 КВ) – Кеш паметта служи като посредник между процесора и оперативната памет, като осигурява на процесора най-често използваните команди и данни.

4/**Многорежимност**.

**\*реален режим** – режимът, в който работят процесори 8088 и 8086;

**\*защитен режим (32-битов)** –Защитеният режим поддържа многозадачност.;

**\*виртуален реален режим** – представлява виртуална 16-битова среда в реален режим, която работи вътре в 32-битовият защитен режим. Това позволява да се изпълняват едновременно няколко програми на един компютър, без каквито и да било доработки на програмите.

**Регистри с общо предназначение -** Използват се както за данни, така и за адреси. Те са 32 разрядни и са 8 на брой.

**Конвейерна обработка на инструкциите -** Базира се на разделяне на подлежащата за изпълнение операция на по-малки части, наричани фази и заделянето за всяка от тях отделен блок апаратура.

**Прекъсванията** в реален режим **се обработват** (обслужват) с помощта на вектори разположени на постоянни адреси от 000h до 3FFh.

**Разлика между i80386dx и i80386sx –** На базата на 32-битовия процесор 80386, чието точно означение е I386DX е създаден по-слаб процесор 386SX, който външно е с 16-битова шина за данни вместо 32-битова.

Компромисният вариант Intel 80386SX има по-малка мощност, но притежава останалите привлекателни страни на 80386 /има 32-битови регистри и може да работи в същите режими/. 80386SX е значително по-евтин от 80386DX.

* 1. **КОМПЮТЪРНА СИСТЕМА IBM/АТ 80486**

**Дънни платки**

**Блокова схема на дънна платка с процесор Intel 80486.**

В архитектурата на компютърните системи с процесор 80486 се наблюдават следните особености:

* въвежда се умножаване на честотата на процесора спрямо честотата на системната шина;
* въвежда се вграден в процесора кеш L1;
* въвежда се концепцията за локалните шини;
* продължава тенденцията за интеграция на чиповете;
* няма единна концепция за архитектурата на компютърните системи с процесор 486. Срещат се системи, както с шина ISA-16, така и с EISA, MCA, VESA, а в по-новите системи - и с PCI.

Блоковата схема се свежда до новата концепция за локалната шина:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Процесор 80486  Външен кеш  Слотове за вход / изход  Чипове за управление на шината  Вграден вход/изход  RAM  Разширителни слотове  **Локална шина**  **VESA**   |  | | --- | |  | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Локална шина VESA, основни характеристики.** Локална шина е всяка шина, конструирана да взаимодейства по-тясно с процесора или по-близо до процесорната производителност. Тази концепция се реализира в шините VESA, PCI, PCI Express и AGP.

**V**ideo **E**lectronics **S**tandard **A**ssociation – **VESA** техническа спецификация на локалната шина VESA VL-Bus. Разширените слотове **VL-Bus** обикновено се използват съвместно със стандартните слотове **ISA** или **EISA**.

Характеристики: - VL-Bus не може да се конфигурира софтуерно, конфигурирането е ръчно, чрез джъмпъри; -пропускателна способност 133 MB/s; - 32 битова шина; - евтин дизайн, не изисква никакви допълнителни интерфейсни чипове; -**Не съществува дънна платка само със слотове за VL-Bus;**

VESA e проектирана като 32 битова шина за процесор 486 и работи най-добре с него. Тя никога не е внедрявана в системи с процесор Pentium.

**Системните ресурси** се използват от хардуерните устройства за комуникиране по шината. На най-ниско ниво тези ресурси обикновено включват следното: - адреси от паметта; - IRQ линии; - DMA канали; - адреси на входно/изходни портове.

**Процес на хардуерни прекъсвания, контролер на прекъсванията.**

В PC съществуват два основни вида прекъсвания: **хардуерни и софтуерни.**

Хардуерните прекъсвания в PC са общо 16 и имат номера от 0 до 15 (обозначават се IRQ0 до IRQ15). Номерата на прекъсванията определят приоритетите им едно пред друго /с някои изключения, прекъсванията с най-високи приоритети имат най-малки номера/. Когато възникне дадено прекъсване, управлението на системата се поема от специална подпрограма, която най-напред запазва съдържанието на всички регистри на процесора в стека, след което насочва системата към **таблицата с векторите на прекъсванията**, съдържаща списък с адреси от паметта, съответстващи на линиите за прекъсвания

**Чрез използване на прекъсвания компютърната система може да реагира своевременно на външни събития.**

Управлението на прекъсванията се извършва от специална логическо устройство, което се нарича **контролер на прекъсванията**. Контролерът на прекъсванията получава заявки за прекъсвания от различните хардуерни устройства посредством линиите за заявки за прекъсвания IRQ. Заявката за прекъсване от дадено устройство (IRQO - IRQ7) се съхранява в **регистъра на прекъсванията** и се сравнява с байта в **регистъра на маскиране на прекъсването**.

**Каскадно свързване на контролерите за прекъсване.** В системите АТ е въведен втори контролер с още 8 канала, който е свързан каскадно към първия. Единият контролер се означава като главен (Master Controller), а другият (новият) - като подчинен (Slave Controller). Прекъсванията, генерирани от втория контролер, се подават през неизползваното IRQ2 на първия контролер, като така се осигуряват общо 15 използваеми прекъсвания.

**ПРОЦЕСОРИ**

**Характеристики на процесор Intel 80486, шина данни, адресна шина и кеш памет. Цокли за процесори 486 SX,DX, DX2, DX4. Процесори 80486 Over Drive. Предимства на процесори i80486. Работни честоти на процесорите i80486DX2 и i80486DX4. Характеристики на CISC дизайна.**

**Основни характеристики на процесор 486**

\*Процесорът 486DX съдържа 1,2 милиона транзистора и се произвежда по CMOS технология, която предопределя ниска консумация на енергия.

\*Чипът притежава 32-битови вътрешни регистри, 32-битова външна шина за данни и 32-битова адресна шина.

\*Процесорите 486 се предлагат на най-различни тактови честоти, като се започне от 16MHz и се стигне до 120MHz.

\*По отношение на инструкциите 486 е напълно съвместим с предишните процесори на Intel, като например 386, но предлага още няколко допълнителни инструкции (повечето от които служат за управление на вътрешния кеш).

\* Може да адресира до 4GB физическа памет и да управлява до 64ТВ (терабайта) виртуална памет.

\*Процесорът 486 напълно поддържа трите режима на работа, въведени с процесора 386 - реален, защитен и виртуален реален.

**Предимства на процесори i80486.**Процесорната фамилия 486 е конструирана за по-висока производителност, защото интегрира в чиповете устройства, които преди това са били външни. Това са кеш контролери, кеш памет и математически копроцесори. 486 системите са проектирани с възможност за ъпгрейд на процесора.

**Версии на процесор 80486**

Първите **486DX** процесори са с максимална работна честота 25MHz, след което се появяват версии на чипа, работещи на 33MHz и 50MHz и 5V захранващо напрежение.

**DX2/OverDrive** версии за ъпгрейд на 486DX системи. DX2/OverDrive процесорите работят с вътрешна честота, която е два пъти по-голяма от тази на системата. Умножаването на вътрешната честота на процесора е ново архитектурно решение, което се използва и до днес.

**DX4 и DX4 OverDrive процесорите** работят в режим на **утроена** честота (а не учетворена, както би се очаквало от названието).

**Цокли за процесори 486 -** От процесорите 486 Intel започват да конструират процесорите си с възможност да бъдат инсталирани и заменяни от потребителите - конструкция с минимално усилие за поставяне (zero insertion force - ZIF). Цоклите ZIF използват лост за привличане или освобождаване на захващането на чипа. Когато лоста е освободен, чипът лесно може да се вмъкне или извади.. Различните типове цокли приемат различни семейства процесори. Ако е известен типа на цокъла или слота на дънната платка, всъщност се знае какви типове процесори могат да се присъединят.

Съществуват две основни тенденции при развитието на микропроцесорите за съвременните КС – **RISC** и **CISC** технологиите.

**CISC (Complex Instruction Set Computer)** – **Пълна система от команди** на микропроцесора. **Технологията CISC** е свързана с традиционните процесори, при които се поддържат множество инструкции, изпълнявани за различно време. Това време е в зависимост от типа на инструкцията, дължината, метода за адресиране на операндите и пр. При тази технология се предполага, че по-голяма група инструкции ще бъде реализирана за апаратно. Това прави електронната структура на микропроцесора доста по-сложна.

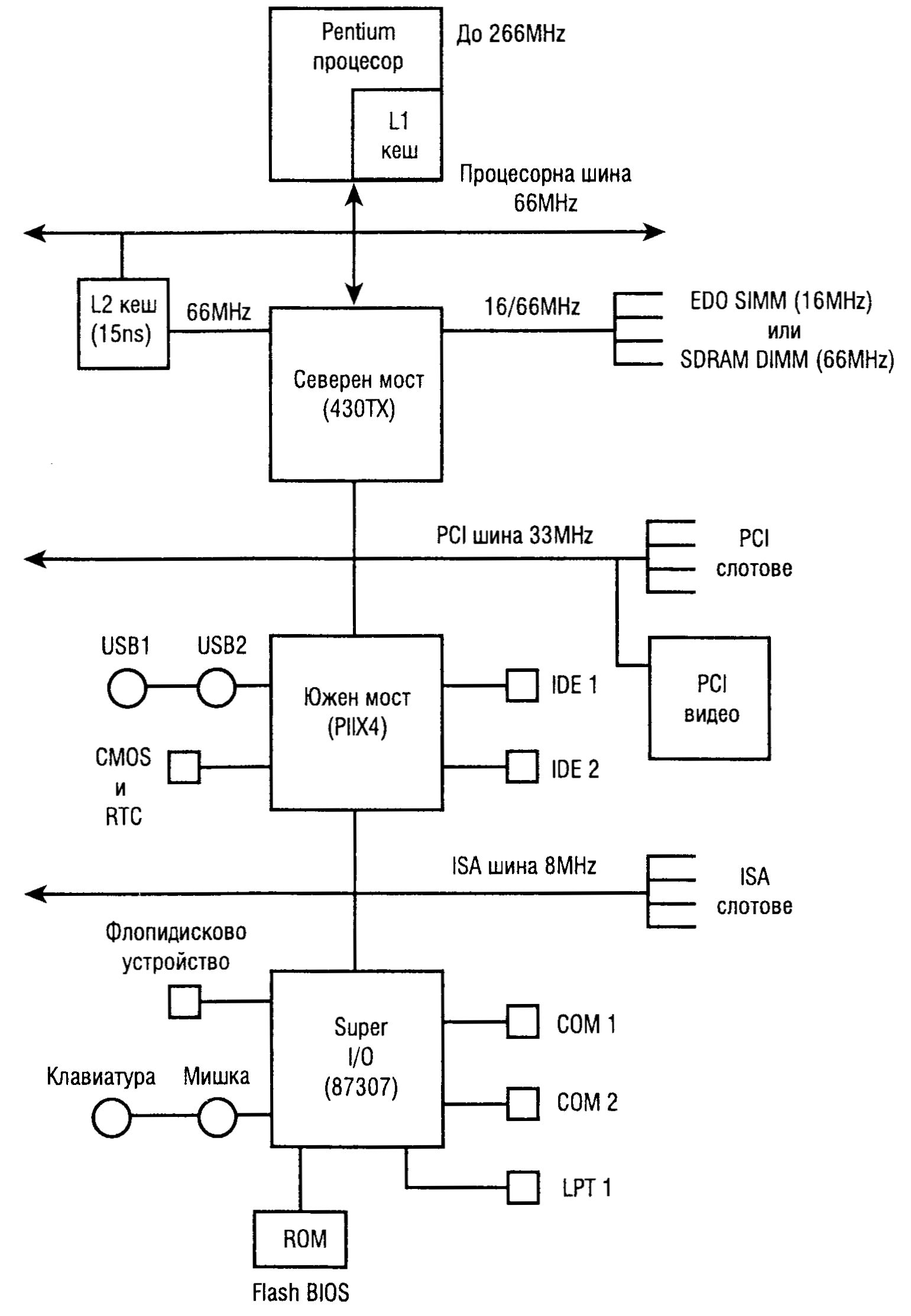
* 1. **КОМПЮТЪРНА СИСТЕМА С ПРОЦЕСОР**

**PENTIUM**

**Дънни платки**

Компютърните системи с процесор Intel Pentium използват архитектура „северен-южен мост”, в която участва новата локална шина PCI.

**Блок-схема на компютърна система с чипсет 430ТХ и процесор Pentium**



**Chipset от серията 430 хх. -** Компютърните системи с този чипсет поддържат процесори Pentium до 266 MHz, с цокъл Socket 7. Кешът от първо ниво L1 е вграден в процесора, а кешът от второ ниво L2 се намира на платката и работи на честота на системната шина, която е 66 MHz.

Първият чипсет на Интел за Pentium е **430LX (Меркурий)** е от три чипа и се характеризира със следното: - Един процесор; - Поддръжка на до 512 KB L2 кеш; - Поддръжка на до 192 MB стандартна DRAM памет с бърз страничен режим (FPM - fast page mode).

**Intel 430NX (Нептун)** - е проектиран да работи с новите 3,3V Pentium процесори от второ поколение. Подобрения: - Поддръжка на два процесора; - Поддръжка на 512МВ системна памет. 430NX се използва в мощни работни станции и мрежовите файлови сървъри от висок клас.

Решение за нисък клас компютри е чипсета **430FX (Тритон).** Чипсетът на Intel **430HX (Triton II)** съчетава най-добрите черти на 430NX и 430FX.

**Чипсетът Intel 430VX** е проектиран като заместник на чипсета от нисък клас 430FX, характеризира се с поддръжката на SDRAM.

**Чипсетът 430ТХ** е разработен за да замести мобилния чипсет 430МХ за лаптопи и ноутбук системи.

**Архитектура на Chipset – Северен-Южен мост (North-South Bridge).**

Устройствата се свързват към три чипа: северен мост, южен мост и входно-изходен контролер (супер I/O чип). Тази архитектура получава името „архитектура северен-южен мост”. **Северният мост** осъществява връзката между високоскоростната процесорна шина и по-бавните AGP и PCI шини. Свързва централния процесор, оперативната памет, видеокартата и южния мост. **Южният мост** осъществява връзката между шина PCI и ISA. Той е директно свързан към северния мост и е отговорен за производителността на входно-изходните: наличните портове за твърди дискове и USB портовете; вградената звукова и мрежова карта; DMA контролерите и контролерите на прекъсванията. **Входно-изходният контролер (Super I/O чипът)** е отделен чип, свързан към ISA шината, съдържа всички стандартни портове за периферия, които са вградени в дънната платка. Осигурява директна комуникация между южния мост и устройствата.

**Недостатъци на архитектурата Северен/Южен мост -** Използването на шината PCI като връзка между двата моста води до значителното й натоварване и превръщането на PCI в тясно място за системата.

**Локална шина PCI, основни характеристики.** PCI (Peripheral Component Interconnect – свързване на периферни компоненти) е стандарт за локална шина, дефиниран от фирмата Intel.

**PCI е паралелна локална входно-изходна шина.** Паралелна означава, че се прехвърлят по 32 или 64 бита ед­новременно, а локална – че се свързва към процесорната шина и стои близо до процесора. PCI шината е с пропускателната способност е 133МВ/сек.: 33.33MHz х 4 байта (32 бита) = 133 МВ/s.

Важно предимство на PCI е, че подпомага работата на системата, защото PCI шината може да работи едновременно с процесорната шина.

PCI осигурява **Bus Mastering -** възможност да се поема управлението на шината от различни устройства, наречени главни устройства (Bus master) и използва централно арбитриране.

PCI поддържа **Plug and Play (PnP)** – включи и задействай.

**PCI слотове и адаптерни (разширителни) карти -** лесно се разпознават на дънната платка по **белия си цвят.** Спецификацията на PCI определя три конфигурации на адаптерни карти:- 5V - за стационарни компютърни системи;

-3,3V - за преносими системи; -универсалната спецификация (5V / 3.3V) е за дънни платки и карти, които работят и в двата вида системи. Всяка спецификация има 32-битова версия и по-дълга 64-битова версия.

**Процесори**

Петото поколение процесори на Intel (с кодово име Р5) се нарича Pentium, a не 586.Pentium процесорът е напълно съвместим с предишните процесори на Intel. Най-съществената разлика е, че Pentium включва два конвейера за данни, което му позволява да изпълнява две инструкции едновременно. **Способността да се изпълняват по две инструкции по едно и също време Intel наричат суперскаларна архитектура.**

**Особености в архитектурата на процесора Pentium:**

**1/Два конвейера за инструкции -** Те се наричат **U-конвейер и V-конвейер.** U-конвейерът, който е главният, може да изпълнява всички целочислени инструкции и инструкции с плаваща запетая. V-конвейерът е вторичен конвейер, който може да изпълнява само прости целочислени инструкции и конкретни инструкции с плаваща запетая.

**2/Буфер за възможните преходи в програмата** - ВТВ (branch target buffer), който използва техника - предсказване на преходи, която позволява на процесора да поддържа и двата конвейера максимално натоварени.

**3/64-битова шина за данни –**

**4/ Отделни вътрешни кешове за инструкции и данни -** организирани като двупосочен асоциативен кеш.Новото при процесорите Pentium е, че притежават кеш с отложен запис (write-back).

**5/ Вторичен кеш (L2) -** обикновено от 512 KB.

**6/ BiCMOS технология -** BiCMOS (Bipolar Complementary Metal - Oxide Semiconductor) технология и суперскаларна архитектура.

**7/ Управление на консумираната енергия -** Всички Pentium процесори са включват SMM за осигуряване на всички енергоспестяващи функции, допринасящи за намаляване на консумираната мощност.

**8/ Вграден математически копроцесор.**

**Основно предимство на цокъла Socket 7** е неговата универсалност и :

-модул за регулиране на напрежение (VRM - Voltage regulator module), който представлява отделна платка или група от схеми, вградени в дънна­та платка, които осигуряват правилното ниво на напрежение и регулират захранването на процесора.;

-поддръжката на широк диапазон от скорости – от 75 до 233 MHz;

-поддръжката на широк диапазон от процесори на различни производители.

Съществуват две основни тенденции при развитието на микропроцесорите за съвременните компютърни системи – **RISC** и **CISC** технологиите.

**RISC (Reduced Instruction Set Computing)** – **Съкратена система команди** на микропроцесора. RISC-процесорите работят с ограничено множество от инструкции и се характеризират със следното: \*броят на инструкциите и видове адресиране са намалени и са въведени явни команди за зареждане и запис на регистрите; \*използват се прости инструкции с еднаква дължина и общ формат; \*операциите са от тип регистър-регистър с три адресна структура; \*премахва се микропрограмното управление, а се използва твърда логика за всяка инструкция; \*избегнати са закъсненията при преходи; \*възможност за изпълнение на 1,5 команди на такт чрез прилагане на конвейеризация на микро ниво и вградена кеш-памет. **Недостатък** на тези технологии са по-сложните процедури при изготвяне на програмното обезпечаване.

* 1. **КОМПЮТЪРНА СИСТЕМА С ПРОЦЕСОР**

**PENTIUM III**

**Дънни платки**

**Блокова схема на дънна платка с процесор Intel Pentium III, socket 370.**



**Chipset от серията 800хх.**

Чипсетите за компютърни системи с процесор Intel Pentium ІІI са от серията 800. Характеризират се със:

- използване на хъбова архитектура;

- видео графичният контролер е интегриран директно в хъба на контролера на паметта (Graphics Memory Controller Hub-GMCH).

Хъбовият интерфейс АНА е 8-битов, с честота 66 MHz и извършва четири трансфера за един такт. Ефективната му пропускателна способност е 4 x 66 MHz x 1 байт =266MB/сек.

**Архитектура на Chipset (Intel Hub Architekture-IHA).**

От началото на 1999 г., новите чипове от серията 800 използват архитектура с хъбове. *Хъб в превод от английски език означава център.*

Севереният мост се нарича **MCH** (Memory Controller Hub – **хъб на контролера на паметта**).

Юженият мост се нарича **ICH** (I/O Controller Hub – **хъб на контролера за вход/изход**).

Системите, които включват интегрирана графика използват **GMCH** (Graphics Memory Controller Hub – **хъб на контролера за графика и памет**) на мястото на стандартните MCH.

MCH чипът осъществява връзката между високоскоростната процесорна шина, шината на паметта, AGP порта и хъбовия интерфейс (всички с 66MHz).

ICH чипът свързва хъбовия интерфейс (66MHz), ATA(IDE) портовете (133MHz) и PCI шината (33MHz). ICH чипът включва и **нова LPC (Low-Pin-Count–малък брой изводи) шина**, представляваща орязана 4-битова версия на PCI, предназначена да осигурява връзка с ROM BIOS-а (наречен хъб за фърмуер FWH- FirmWare Hub) и Super I/O чипа на дънната платка.

Има две основни разновидности на хъбов интерфейс:

**\*АНА (ускорена хъбова архитектура**) използвани от серията 8xx на чипсети. Хъбовият интерфейс АНА е 8-битов, с честота 66 MHz и извършва четири трансфера за един такт. При това положение ефективната му пропускателна способност е 4 x 66 MHz x 1 байт =266MB/сек.

**\*DMI (директен интерфейс на средата)** използван от 9xx и следващи серии чипсети. DMI е PCI Express връзка с 4 алеи (широка 4 бита), позволяваща скорост 1Gb/s (250GHz Х 4 бита) във всяка посока едновременно.

**Предимства на хъбовата архитектура:**

\* **По-голяма пропускателна способност** от PCI;

**\*Намалено натоварване на PCI шината**;

\***По-висока пропускателна способност;**

**\*Намалено опроводяване на платката**;

\*Намаленият брой изводи означава **по-малко писти** по дънната платка, **по-малко електромагнитни смущения и грешки при синхронизацията**, а самите чипове имат по-малко изводи, което ги прави по-малки и **по-изгодни за производство.**

\***LPC шината** има максимална пропускателна способност 16,67 MB/s, която е около 2 пъти по-голяма от тази на ISA шината.

**Тактови честоти на дънната платка, схеми за генериране на тактови честоти, кварцов генератор, настройка. -** На всяка дънна платка са необходими различни тактови честоти, с чиято помощ таймерът и останалата електроника върху платката се снабдява с "работните" си тактови сигнали.

**Кварцов генератор /** **свойство пиезоелектричност/** е основният компонент, който управлява скоростта на компютъра. В персоналните компютри се използват кварцови генератори, които могат да бъдат **кварцови резонатори** или **кварцови осцилатори**. **Съвременния РС има поне два кварцови генератора: главен, с честота 14,31818 MHz, който управлява скоростта на дънната платка и схемите, свързани с нея и генератор за часовника (RTC-real time clock), който е с честота 32.768 KHz.**

**Генериране на необходимите тактови честоти** - В едно РС има йерархия от няколко шини с различни скорости, като всяка по-бавна шина е свързана с по-бързата от нея. **Използваните в компютърните системи тактови честоти са следните:**

\* Системната (процесорната) шина работи с някоя от стандартните честоти: 66 MHz, 100 MHz, 133 MHz, 200 MHz, 266 MHz, 400 MHz, 533 MHz, 1066 MHz.

\*AGP шината работи на 66 MHz, 133 MHz, 266 MHz или 533 MHz.

\* PCI шината работи на 33 MHz или 66 MHz (за работни станции и сървъри).

\* ISA шината (16-бита) работи на 8,33 MHz.

\*Шината на паметта работи на 22 MHz при FPM, 33 MHz при EDO, 67-100-133 MHz при SDRAM, 100 до 400 MHz при DDR, 300 до 1066 MHz при RDRAM. Специален чип, наречен **честотен генератор** **(frequency timing generator - FTG)** или **честотен синтезатор** се използва заедно с кварцовия генератор, за да произведе необходимите честоти. Повечето от тези чипове са програмируеми и настройваеми, така че те могат да променят техните изходни честоти чрез софтуер, което води до работа на системата с различни скорости.

**Настройки на тактовите честоти. Овърклокване.** Овърклокване (издуване, изпържване) се нарича работата на определен компонент или цялата система със скорост, по-голяма от посочената. Настройките се правят чрез постепенно увеличаване на скоростта на процесора, паметта и шините докато системата стане нестабилна. След това се намаляват, докато системата стане отново стабилна. При овърклокване компонентите работят при повишени температури и затова трябва да се вземат мерки за **допълнително охлаждане.**

**AGP порт, основни характеристики, режими 2х. -** AGP е пряка, високопроизводителна връзка от точка до точка, предназначена специално за свързване на графична карта (видеокарта) в дадена система. Чрез AGP може да се включи само един тип устройство – графична карта и то само една, тъй като може да съществува само един AGP порт.Конекторът на **AGP** е подобен на PCI, но притежава допълнителни сигнали и механично и електрически е несъвместим с PCI. За да се различава от PCI, е в **кафяв цвят**. Разположен е близо до северния мост, процесора и RAM.

Подобрява ефективността на използване на системните ресурси по следните начини:

* **Специално предназначен порт**;
* **Pipeline (конвейерно) адресиране**. AGP може да получава множество пакети от данни с една заявка;
* **Адресиране по странична шина (SBA – Side-Band Addressing)** - 8 допълнителни адресни линии освен основните 32 линии. Адресната информация по тази странична шина може да се предава паралелно с данните по основната шина.;
* AGP ускорява процеса на рендиране на графиката и чрез **по-ефективно използване на системната памет**.

**Спецификации и режими на работа на AGP**

**Процесори на AMD съвместими с Pentium III процесорите.**

Серията K6 е голям успех за AMD, който спасява компанията от фалит. Процесорите K6 са също толкова бързи и дори по-бързи при изчисленията с цели числа от своите конкуренти [Pentium II](http://bg.wikipedia.org/w/index.php?title=Pentium_II&action=edit&redlink=1) и [Pentium III](http://bg.wikipedia.org/w/index.php?title=Pentium_III&action=edit&redlink=1). Освен това AMD за пръв път въвеждат свои собствени разширения в набора от инструкции на x86-архитектурата (3DNow!), но в повечето популярни програми така и не се появява широка поддръжка за тези инструкции. K6 приближава AMD до нивото на производителност на Intel, все пак остава назад. Допълнителен проблем се оказва нововъведеният стандарт Super Socket 7, чипсетите за който се произвеждат от по-малки компании и се оказват със съмнително качество и съвместимост, особено с нововъведения стандарт за шина на графичните карти [AGP](http://bg.wikipedia.org/wiki/AGP).

Проектът на K7 е много добре балансиран и премахва повечето тесни места в архитектурата на процесорите от предното поколение. Освен това, поучени от горчивия си опит с K6, от AMD реализират изключително мощен блок за изчисления с плаваща запетая. В резултат, полученият процесор, който след конкурс получава наименованието [Athlon](http://bg.wikipedia.org/w/index.php?title=Athlon&action=edit&redlink=1), не само може да изпълнява повече инструкции при една и съща тактова честота с Pentium III, но и има по-добър потенциал за нарастване на честотата.

През август 1999 г. AMD представя първите чипове Athlon, които работят на забележителната за този момент честота от 650 MHz. За пръв път в историята AMD настига и надминава по производителност флагманските чипове на Intel. Заедно с Athlon AMD въвежда и нов сокет - т. нар. [Slot A](http://bg.wikipedia.org/w/index.php?title=Slot_A&action=edit&redlink=1)

1. **Реален режим.**

* История.

Първите процесори от семейството x86 (8086, 8088 и 80186) са 16-битови и имат един и същ програмен модел, който се поддържа от всички следващи поколения x86 процесори. След появата на защитения режим, този режим започва да носи наименованието реален режим.

Въпреки ограниченията на реалния режим, в момента на появата си това е била една от най-сложните и универсални процесорни архитектури.

В наши дни реалният режим се използва само при стартирането на операционната система, тъй като всеки x86 процесор веднага след включване на захранването автоматично влиза в реален режим. Това налага първите етапи от зареждането на операционната система да стават в този режим. Популярната в миналото DOS на Microsoft работи почти изцяло в реален режим.

* Въведение.

Реален режим (на английски: real mode) е режим на работа на компютърните процесори в архитектурата x86. Всеки x86 процесор влиза в реален режим веднага след включване на захранването му. При реалния режим, максималната адресирана памет е 1 MB (220 байта) и няма апаратна защита на достъпа до паметта.

* Технология.

Реалният режим на работа се характеризира със следните архитектурни особености:

* 14 16-битови регистъра, от които 8 са с общо предназначение.
* 1 MB (220 байта) адресно пространство.
* 64 KB (216 байта) входно-изходно адресно пространство.
* Фиксиран размер на сегмента на паметта – 64 KB (216 байта)
* 7 типа данни.
* 9 метода за адресиране.
* Над 80 машинни инструкции.

Трябва да се има предвид, че горепосочените особености са в сила само за „чистия“ реален режим – т.е. този, който е напълно обратно съвместим с 16-битовите x86 процесори: 8086 и 80186 на Intel. При 32-битовите процесори от x86 фамилията дори в реален режим е възможен достъпът до пълния 32-битов размер на регистрите, както и допълнителни типове данни и методи за адресиране. Ако тези разширения бъдат използвани, то програмата няма да е обратно съвместима с 16-битовите x86 процесори.

* Регистри.

В реален режим всеки x86 процесор има достъп до 14 16-битови регистъра. Осем от тях (AX, BX, CX, DX, SP, BP, SI и DI) са регистри с общо предназначение. Въпреки това, всеки от тях си има свои специфични функции (например достъпът до AX става най-бързо и с най-къса машинна инструкция, а CX се използва за брояч при цикли). Всеки от регистрите AX, BX, CX и DX може да бъде разглеждан и като 2 8-битови регистъра, до които има независим достъп (напр. горните 8 бита на DX са 8-битов регистър с името DH, а долните 8-бита – 8-битов регистър с името DL).

Регистрите SP и BP се използват при работата със стека, като първият е указателят на стека, а вторият обикновено се използва от езиците на високо ниво за формирането на така наречения стек фрейм (stack frame) на процедурите. Регистрите SI и DI се използват при формирането на адреса при някои инструкции (най-вече при операции със стрингове).

Други 4 16-битови регистъри са сегментните регистри CS, DS, ES и SS, които се използват при формирането на адресите. Регистърът FLAGS съдържа еднобитови флагове, които отразяват текущото състояне на процесора и събития като препълване, пренос към по-старши разред, нулев резултат и т.н. Последният регистър е програмният брояч IP, който съдържа адреса на следващата инструкция, която трябва да бъде изпълнена от процесора.

* Типове данни.

В реален режим x86 процесорът може да обработва следните типове данни:

* **Цели числа без знак -** Могат да бъдат 8-битови (байт) или 16-битови (дума). Байтът има диапазон на възможните стойности от 0 до 255 (28 – 1), а думата – от 0 до 65536 (216 – 1).
* **Цели числа със знак -** Могат да бъдат 8-битови или 16-битови, в допълнително кодиране (най-старшият бит е знаков). 8-битовите имат диапазон на възможните стойности от -128 до 127, а 16-битовите – от -32768 до 32767.
* **Двоично-десетични (BCD) числа -** Имат две разновидности – пакетирани и непакетирани. При непакетираните BCD числа във всеки байт от числото се съдържа една десетична цифра. При пакетираните BCD числа във всеки байт са пакетирани 2 десетични цифри (едната в младшите 4 бита, а другата – в старшите 4 бита).
* **Стринг от байтове -** Последователност от байтове с максимален размер до 65535 (216 – 1) байта.
* Методи за адресиране.

Повечето x86 инструкции са двуадресни (т.е. работят върху 2 операнда). Първият операнд е както първия източник, така и вместилище за резултата от операцията, а вторият операнд е вторият източник.

Операндите може да се намират в паметта, в регистър (обикновено някой от регистрите с общо предназначение), или да са зададени като непосредствена стойност, която е закодирана в самата инструкция. В повечето случаи е допустимо само единият операнд да е разположен в паметта или да е непосредствен (закодиран директно в инструкцията).

Когато единият операнд е разположен в паметта, адресът му се определя по следната формула:

**A = B + I + O**

където:

**A** - полученият ефективен адрес на операнда,

**B** - базов адрес (задава се в регистър BX или BP),

**I** - индекс (задава се в регистър SI или DI),

**O** - отместване, което е непосредствено закодирано в самата инструкция като 8- или 16-битово число със знак.

Трябва да се има предвид, че полученият по този начин 16-битов адрес, всъщност представлява отместване в рамките на подразбиращия се сегмент в паметта. Началният адрес на сегмента се получава като се умножи по 16 (т.е. като се измести наляво с 4 разреда) съдържанието на съответния сегментен регистър (CS, DS, SS, ES). Крайният физически адрес на данните в паметта се получава като се събере началният адрес на сегмента с полученото по горната формула отместване.

Кой точно сегментен регистър се използва при изчисляването на адреса зависи от вида на зарежданите данни: CS се използва за инструкциите, DS – за данните, SS – за стека, а ES – при някои инструкции за работа със стрингове. В повечето случаи е възможно подразбиращият се сегментен регистър да бъде променен с префикс на машинната инструкция.

Сегментната организация на паметта в реалния режим се използва единствено за да е възможно адресирането на 1 MB памет от 16-битов процесор. От гледна точка на процесора сегментите винаги са дълги 64 KB и няма начин да бъде ограничен достъпът до някои части от паметта.

Поради описаната сегментна организация на паметта, се използват два вида указатели: 16-битови (наричани близки – near) и 32-битови (наричани дачечни – far). Близките указатели представляват отместване в рамките на един сегмент, а далечните указатели се състоят от 16-битов начален адрес на сегмента (който се зарежда в съответния сегментен регистър) и 16-битово отместване в рамките на сегмента.

За разлика от други процесорни архитектури, в x86 е възможно адресирането на думи, намиращи се на произволен адрес. Все пак, когато адресът не е кратен на размера на думата, е възможно достъпът да е по-бавен.

* Входно-изходно адресно пространство.

При x86 архитектурата могат да се използват и двата основни начина за връзка с входно-изходните устройства на компютъра – вмъкване в основното адресно пространство и използването на специално входно-изходно адресно пространство.

Входно-изходното адресно пространство е с размер 64 KB (216 байта) и достъпът до него става с помощта на специални инструкции (IN, OUT). Методите на адресиране, които се използват от тези инструкции, са: непосредствено задаване на адреса в инструкцията или задаване на адреса в регистрите AL или AX. Когато адресът е в първите 256 байта от входно-изходното адресно пространство, може да се използва 8-битов адрес вместо 16-битов.

* Инструкции.

Инструкциите на x86 са сложни и гъвкави (x86 е CISC архитектура). Инструкциите в реален режим са основният вид инструкции, които се използват в програмите за тази архитектура (тези инструкции са достъпни и в защитен режим, но с повече типове данни и методи на адресиране). Инструкциите може да се разделят условно на няколко групи.

* Инструкции за трансфер на данни.
  + MOV – копиране на данни от първия във втория операнд. (Поне единият трябва да е регистър)
  + PUSH – записване на данни в стека и декрементиране на стековия указател.
  + POP – четене на данни от стека и инкрементиране на стековия указател.
  + IN, OUT – четене от и запис във входно-изходното адресно пространство.
  + XCHG – размяна на съдържанието на двата операнда. (Поне единият трябва да е регистър)
  + LEA – изчисляване на ефективния адрес и прехвърлянето му в регистър.
  + XLAT – бърз достъп до таблици с индекс с размер до 1 байт.
* Аритметични инструкции.
  + ADD, SUB – събиране и изваждане на цели числа.
  + ADC, SBB – събиране и изваждане на цели числа с отчитане на преноса.
  + MUL, IMUL – умножение на цели числа без и със знак.
  + DIV, IDIV – делене на цели числа без и със знак.
  + AAA, AAS, DAA, DAS, AAM, AAD – специализирани инструкции за корекции на резултата след аритметична операциия с BCD числа.
  + NEG – смяна на знака на цяло число.
  + INC – увеличаване с 1 на цяло число.
  + DEC – намаляване с 1 на цяло число.
  + CBW – разширяване на 8-битово число със знак в 16-битово.
  + CWD – разширяване на 16-битово число със знак в 32-битово.
  + CMP – сравняване на цели числа.
* Логически инструкции.
* AND, OR, NOT, XOR – побитови логически операции.
* ROL, ROR, SAL, SAR, SHL, SHR – аритметични и логически побитови измествания.
* TEST – побитово сравнение.
* Инструкции за работа със стрингове.
  + CMPS – сравнение на стрингове.
  + SCAS – сравнение на стринг с регистър.
  + MOVS – трансфер на стринг от паметта на друго място в паметта.
  + LODS – прехвърляне от стринг в регистър.
  + STOS – прехвърляне от регистър в стринг.
  + REP, REPE/REPZ, REPNE/REPNZ - префикси, които се използват заедно с инструкциите за работа със стрингове.
* Инструкции за преход.
* JMP – безусловен преход.
* JO, JNE, JB/JNAE/JC, JNB/JAE/JNC, JE/JZ, JNE/JNZ, JBE/JNA, JS, JNS, JP/JPE, JNP/JPO, JL/JNGE, JNL/JGE, JLE/JNG, JNLE/JG - условен преход (в зависимост от съдържанието на регистър FLAGS).
* CALL – извикване на подпрограма.
* RET – връщане от подпрограма. (RETF ако програмата е извикана чрез Сегмент и Отместване)
* LOOP – безусловен преход в рамките на цикъл.
* LOOPE/LOOPZ, LOOPNE/LOOPNZ - условен преход в рамките на цикъл.
* Други инструкции.
  + INT, INT3, INTO – предизвикване на прекъсване.
  + IRET – връщане от подпрограма за обработка на прекъсване.
  + BOUND – проверка за нарушаване на границите на масив.
  + SAHF, LAHF – прехвърляне на младшия байт на FLAGS във и от AH.
  + PUSHF, POPF – запис и четене на FLAGS във и от стека.
  + CLC, STC, CMC, CLD, STD, CLI, STI – нулиране и установяване на различни флагове във FLAGS.
  + HLT – спиране на процесора.
  + WAIT – чакане на копроцесор.
  + NOP – празна инструкция.
* Проблеми.

Както вече беше споменато, в момента на създаването си, x86 архитектурата (която тогава е еквилалентна на сегашния реален режим) е много авангардна. За пръв път се използва сегментна организация на паметта (макар и доста ограничена) в микропроцесор. Регистрите са много повече от тези на тогавашните микропроцесори, методите за адресиране и инструкциите са гъвкави и разнообразни.

С течение на времето обаче, 16-битовият реален режим става все "по-тесен" на съществуващите операционни системи и приложни програми по следните причини:

Ограничението от 1 MB адресно пространство е твърде малко в сравниение с нарастващите нужди на приложните програми.

Липсата на апаратна защита на паметта води до ниска стабилност и сигурност на операционните системи, които изключително лесно могат да бъдат блокирани и повредени от зле работещи или злонамерени програми (компютърни вируси).

16-битовият размер на машинната дума е твърде малък и често се налага обработката на големите числа с няколко инструкции, което значително забавя изпълнението на програмния код.

Всички тези проблеми бяха решени с разширяването на машинната дума до 32 бита и въвеждането на т.нар. защитен режим (protected mode).

* Приложения.

Изключително популярната в миналото операционна система DOS работи почти изцяло в реален режим. Първите версии на операционната система Windows (Windows 1.0 и Windows 2.0) също използваха реалния режим на работа.

В наши дни реалният режим се използва изключително рядко - най-вече в секторите за начално зареждане. Всички съвременни операционни системи и приложни програми използват защитения режим.

1. **Защитен режим.**

* История.

Защитеният режим се характеризира с апаратно предотвратяване на достъпа на дадена програма извън границите на заделената ѝ памет. Първата версия на защитения режим се появи в 16-битовия процесор 80286. Въпреки че някои от тогавашните операционни системи (най-вече OS/2 и Windows 3.0) включваха поддръжка за 16-битовия защитен режим. Въпреки че 80286 има апаратна защита на паметта, той наследява малкото адресно пространство на своя предшественик Intel 8086 и в резултат 16-битовият защитен режим не получава широко разпространение (дори се стига до това, че в следващите процесори от семейството x86 няма обратна съвместимост със 16-битовия защитен режим на 286). Скоро се появява по-универсалния 32-битов защитен режим с пускането на процесора 80386. От този момент под защитен режим ще се подразбира 32-битовият защитен режим на 80386 и следващите x86 процесори.

През 1985 г. Intel пуска на пазара първия 32-битов x86 процесор 80386, който поддържа 32-битовия защитен режим. Към средата на 90-те години на 20-и век Microsoft започва да използва предимно 32-битов код в своите операционни системи и от този момент нататък 32-битовият защитен режим става основен режим на работа на x86 процесорите.

В съвременните операционни системи се използва защитеният режим на работа. Освен това сегментирането се използва много ограничено, като защитата и управлението на паметта стават основно с помощта на механизма на страницирането. Именно поради тази причина, в 64-битовото разширение на x86 архитектурата AMD64, сегментирането е „орязано“ до минимум.

В наши дни (2006) все още над 95% от софтуера за x86 процесори работи в 32-битовия защитен режим, но вече започнаха да се появяват операционни системи и приложни програми, които работят в новия 64-битов защитен режим, въведен от AMD под името long mode.

* Въведение.

Защитен режим (на английски: protected mode) е режим на работа на компютърните процесори в архитектурата x86. При защитения режим, максималната адресирана памет е 4 GB (232 байта), има 'странициране' и 'сегментиране' на паметта, както и апаратна защита на достъпа до паметта и входно-изходното пространство. Почти всички съвременни операционни системи за x86 процесори работят изключително в защитен режим (и неговия наследник - long mode, въведен с разширенията x86-64).

* Технология.

Тъй като защитеният режим е базиран на реалния режим и наследява повечето му архитектурни особености, тук ще бъдат посочени само разликите с реалния режим.

* Разширяване на всички регистри от 16 на 32 бита.
* 4 GB (232 байта) адресно пространство.
* 64 KB (216 байта) входно-изходно адресно пространство.
* Сегментиране на паметта с гъвкав размер на сегмента.
* Странициране на паметта.
* 2 нови типа данни.
* 5 нови метода за адресиране.
* Множество нови инструкции.
* Регистри.

При защитения режим всички регистри стават 32-битови, а имената им се образуват от префикса E и името на съответния 16-битов регистър. 32-битовите регистри с общо предназначение са EAX, EBX, ECX, EDX, ESP, EBP, EDI и ESI, флаговият регистър е EFLAGS, а програмият брояч - EIP. Сегментните регистри си остават 16-битови (но адресът се формира по друг начин), като към вече съществуващите CS, DS, SS и ES се прибавят два нови сегментни регистъра, наречени FS и GS.

Няма директен начин за достъп до старшите 16 бита на 32-битовите регистри EAX, EBX, ECX и EDX, но се запазва възможността за директен достъп до младшия и старшия байт на младшите 16 бита под имената AL, AH, BL, BH и т.н.

Всички нови регистри са достъпни в пълния им 32-битов размер и в реален режим, но подразбиращият се размер е 16-битов и трябва да се използва специален префикс на инструкцията, за да се работи с 32-битовите регистри. В реален режим сегментните регистри FS и GS работят като останалите сегментни регистри и дават възможност за достъп до 6 16-битови сегмента наведнъж.

* Типове данни.

В добавка към типовете данни, които могат да бъдат обработвани в реален режим, в защитения режим могат да бъдат обработвани следните типове данни:

* 32-битови цели числа без знак. Имат диапазон на възможните стойности от 0 до 232 - 1.
* 32-битови цели числа със знак. Имат диапазон на възможните стойности от -231 до 231 - 1.
* Стринг от байтове. Последователност от байтове с максимален размер до 232 - 1 байта.
* Методи за адресиране.

При защитения режим физическият адрес се формира по по-сложен начин в сравнение с реалния режим. От самата инструкция се определя т.нар. ефективен адрес, който се преобразува в линеен адрес чрез механизма за сегментиране на паметта. Ако не е включен механизма за странициране на паметта, то линейният адрес съвпада с физическия, но ако механизмът за странициране е включен, той преобразува линейния адрес в съответния физически адрес.

* Формиране на адресите.

Ефективният адрес при защитения режим представлява отместване в рамките на един от текущо използваните сегменти (също като при реалния режим). Ефективният адрес се определя по следния начин:

A = B + I\*S + O

където:

A е полученият ефективен адрес на операнда,

B е базов адрес (задава се в някой от регистрите с общо предназначение),

I е индекс (задава се в някой от регистрите с общо предназначение, с изключение на ESP),

S е мащаб (може да бъде 1, 2, 4 или 8) (задава се като непосредствен операнд в самата инструкция),

O е отместване, което е непосредствено закодирано в самата инструкция като 8-, 16-, или 32-битово число със знак.

Както се вижда, адресирането при защитения режим е значително по-гъвкаво от това при реалния режим. Трябва да се има предвид, че при процесорите, които поддръжат защитения режим, горепосочените разширения в начина на адресиране са достъпни и в реален режим.

* Сегментиране на паметта.

Полученият по посочения в предната точка начин ефективен адрес, всъщност представлява отместване в рамките на сегмент от паметта. Докато в реалния режим сегментите се използват само като средство за преодоляване на ограниченото адресно пространство, в защитения режим сегментите служат изключително за осигуряването на апаратна защита на паметта от непозволен достъп.

В защитения режим сегментите са дълги до 4 GB (232 байта), като началният им адрес, размерът им и другите им атрибути са записани в така наречения сегментен дескриптор. Сегментните дескриптори са подредени в таблица в паметта, наречена дескрипторна таблица. Може да има множество такива таблици, като във всеки един момент трябва да има поне 2 дефинирани таблици - глобалната дескрипторна таблица (GDT), която е една и съща за всички процеси; и локалната дескрипторна таблица (LDT), която може да е различна за всеки процес. Адресите на GDT и LDT се зареждат в специални 32-битови регистри.

Всяка дескрипторна таблица може да съдържа до 213 дескриптора на сегменти (213 - 1 за GDT, защото първият запис не се използва). Сегментните регистри съдържат 16-битов индекс на някои от дескрипторите в GDT или LDT. Един от битовете на индекса определя дали се прави достъп до LDT или GDT, други два определят обявеното ниво на привилегированост на сегмента (използва се при защитата на паметта), а останалите 13 бита съдържат номера на съответния дескриптор на сегмент в съответната таблица.

По отношение на защита на паметта, всеки сегмент има като атрибут т. нар. ниво на привилегированост (цяло число от 0 до 3, като 0 е най-привилегированото ниво). Обикновено операционната система работи на ниво 0, а потребителските програми - на ниво 3. Останалите нива не се използват при повечето съвременни операционни системи.

Действието на защатита на паметта може да бъде обобщено по следния начин:

* От дадено ниво на привилегированост може да се прави достъп до данни само в същото ниво на привилегированост или в по-малко привилегировано ниво.
* От дадено ниво на привилегированост може да се викат подпрограми и да се предава управлението само в същото ниво на привилегированост или в по-привилегировано ниво.
* Странициране на паметта.

Полученият след сегментирането линеен адрес съвпада с физическия адрес само ако не е включено страницирането на паметта (страницирането на паметта е опционално и може да се изключи). При включено странициране на паметта, линейният адрес се преобразува във физически по описаната по-долу схема.

Страниците на паметта при защитения режим са с размер 4 KB (212 байта) и са подравнени на гранците 4 KB (по-късно са въведени страници с по-голям размер - 4 MB, за да може да се използват по-големи адресни пространства). 32-битовият линеен адрес може да се разглежда като съвкупност от:

* Най-старшите 10 бита определят индекс в каталога на страниците, който представлява таблица от 1024 елемента в паметта. Всеки от тези елементи сочи към адреса на една от таблиците на страниците, или към началния адрес на страница от 4 MB; или е невалиден.
* Следващите 10 бита определят индекс на елемент в таблицата на страниците, която също има 1024 елемента, всеки от които или сочи към страница в паметта или е невалиден.
* Най-младшите 12 бита от адреса опреледят отместването в рамките на една страница. Ако страницата е с размер 4 MB, то най-младшите 22 бита са отместването в страницата.

Създаването на каталога и таблиците на страниците е задължение на операционната система, като началният адрес на каталога на страниците е записан в един от управляващите регистри на процесора.

Страницирането на паметта осигурява възможност за използването на виртуална памет, тъй като една страница може да бъде обявена за невалидна в таблицата на страниците и всеки достъп до нея предизвиква специално прекъсване, което дава възможност на операционната система да зареди съответната страница в паметта.

В съвременните операционни системи за управление на паметта се използва предимно страницирането, а сегментирането почти не се използва (обикновено всички сегменти са с максимален размер и начален адрес 0, което на практика ги обезсмисля). Това се налага поради факта, че при писането на приложни и системни програми сегментирането създава много главоболия и проблеми, а страницирането е напълно невидимо за потребителските програми.

Защитата на паметта от гледна точка на страницирането е доста "орязана": в описанието на страницата има само 2 еднобитови флага, които осигуряват следните нива на достъп: забранен всякакъв достъп, разрешено четене и разрешено четене и запис. В по-съвременните x86 процесори (произведени след 2003 г.) е въведен т. нар. NX бит, който показва дали е разрешено изпълнението на програмен код в съответната страница. Това повишава сигурността на операционната система, предотвратявайки една от най-често използваните хакерски атаки - препълването на буфер.

* Входно-изходно адресно пространство.

При защитения режим входно-изходното адресно пространство е със същия размер като при реалния режим (64 KB), но достъпа до него е разрешен само при определени условия:

* Нивото на привилегированост на текущо изпълнявания процес трябва да е по-голямо от зададеното в двубитовото поле IOPL на регистъра EFLAGS. Естествено, съдържанието на IOPL не може да се променя от задачите с ниско ниво на привилегированост.
* Ако предната проверка е положителна, то се прави допълнителна проверка чрез така наречената таблица за защита на входно-изходното пространство. Тази таблица съдържа по 1 бит за всеки от адресируемите 64 536 входно-изходни порта, като този бит разрешава или забранява достъпа до съответния порт.

Ако някоя от горните проверки даде отрицателен резултат, следните инструкции не могат да бъдат изпълнени:

* IN, OUT, INS, OUTS (правят се и двете проверки)
* CLI, STI (прави се само първата проверка)

Както се вижда, освен достъпа до входно-изходното адресно пространство, се ограничава и способността на програмата да маскира прекъсванията, за да не може да "завземе" процесорното време, игнорирайки прекъсванията от часовника.

Както полето IOPL, така и таблицата за защита на входно-изходното пространство са специфични за всеки процес, която позволява на операционната система да управлява достъпа до входно-изходното пространство изключително гъвкаво.

* Инструкции.

Следва описание на новите инструкции в сравнение с реалния режим. Трябва да се има предвид, че повечето нови инструкции могат да се използват и в реален режим. Новите инструкции, които могат да бъдат използвани само в защитен режим, са специално отбелязани.

* Инструкции за трансфер на данни.
* MOVSX - разширяване на операнда от 8 до 16 или от 16 до 32 бита с отчитане на знака и прехвърляне на резултата в регистър с общо предназначение.
* MOVZX - разширяване на операнда от 8 до 16 или от 16 до 32 бита без отчитане на знака (допълване с нули отляво) и прехвърляне на резултата в регистър с общо предназначение.
* PUSHA - записване на всички регистри с общо предназначение в стека.
* POPA - зареждане на всички регистри с общо предназначение от стека
  + Инструкции за побитови операции.
* BSF, BSR - търсене на първия ненулев бит, започвайки съответно от най-старшия и най-младшия бит на операнда.
* BT - проверка на даден бит (бита се записва в полето CF на на регистъра EFLAGS).
* BTC - също като BT, като стойността на адресирания бит се променя.
* BTR - също като BT, като адресираният бит се нулира.
* BTS - също като BT, като адресираният бит се установява в 1.
* Инструкции за работа със стрингове.
* INS - зареждане на стринг от порт във входно-изходното адресно пространство.
* OUTS - запис на стринг в порт във входно-изходното адресно пространство.
  + Инструкции за преход.
* ENTER - създаване на стек фрейм (stack frame) на подпрограма.
* LEAVE - освобождаване на стек фрейм (stack frame) на подпрограма.
* SETO, SETNE, SETB/SETNAE/SETC, SETNB/SETAE/SETNC, SETE/SETZ, SETNE/SETNZ, SETBE/SETNA, SETS, SETNS, SETP/SETPE, SETNP/SETPO, SETL/SETNGE, SETNL/SETGE, SETLE/SETNG, SETNLE/SETJG - условно установяване на байт.
  + Инструкции, работещи само в защитен режим.
* ARPL - настройка на нивото на привилегированост.
* LAR - процитане на част от дескриптора на сегмент от паметта.
* LGDT, SGDT - запис и четене на адреса на глобалната дескрипторна таблица в съответния регистър.
* LIDT, SIDT - запис и четене на адреса на дескрипторната таблица за прекъсванията в съответния регистър.
* LLDT, SLDT - запис и четене на адреса на локалната дестрипторна таблица.
* LMSW, SMSW - запис и четене на младшите 16 бита на регистър CR0.
* LSL - прочитане на размера на сегмент.
* LTR, STR - запис и четене на регистъра на задачата.
* VERR, VERW - проверка на сегмент за право на четене и четене/запис.
* Проблеми.

Като цяло защитеният режим е много гъвкав и напълно задоволяваше нуждите на операционните системи и приложните програми през последните 20 години. Все пак съществуват определени проблеми:

* 32-битовото адресно пространство дава достъп до 4 GB директно адресирирана памет. Към 2006 повечето персонални компютри все още използват значително по-малко памет, но вече не са рядкост приложенията, за които 4 GB памет са недостатъчно. Този проблем бе разрешен от 64-битовото разширение на архитектурата, наречено x86-64.
* Повечето съвременни операционни системи не използват механизма за сегментиране на паметта, а само този за странициране на паметта. Това е проблем от гледна точка на сигурността, понеже няма начин да се забрани изпълняването на програмен код в дадена страница, ако процеса има достъп за четене до нея. Този пропуск се използва от хакерите чрез т.нар. атака чрез препълване на буфера, при която се записва и изпълнява програмен код в буферите за данни на програмата. Този проблем бе разрешен с добавянето на специален флаг в дескриптора на страницата, наричан NX бит.
* Броят на регистрите с общо предназначение на x86 процесорите винаги е бил считан за твърде малък. Свръхбързите кеш памети от първо ниво на съвременните процесори донякъде облекчиха този проблем, но истинско подобрение дойде с въвеждането на x86-64, при което бяха добавени още 8 регистъра с общо предназначение.
* Приложения.

Защитеният режим се използва масово във всички съвременни операционни системи и приложения за x86 процесорите. В бъдеще се очаква плавен преход към 64-битовия long mode (дълъг режим), който е пряк наследник на защитения режим. Новите операционни системи се разработват и за двата режима, но повечето приложни програми ще продължат да работят в защитен режим още дълго време, тъй като 64-битовите операционни системи могат да работят с 32-битови програми.

* Процесори, които поддържат защитен режим.
* Произведени от Intel: 386, 486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4, Celeron, Pentium M, Core, Core 2, Xeon.
* Произведени от AMD: Am386, Am486, K5, K6, Athlon, Athlon XP, Athlon 64, Athlon X2, Duron, Sempron, Opteron, Geode, Phenom.
* Произведени от Cyrix: 486SLC, 486DLC, 5x86, 6x86, MII, MediaGX.
* Произведени от VIA: C3, C7.

1. **Инструкции за числа с плаваща запетая (x87 инструкции).**

До появата на 486, x86 архитектурата не разполага с инструкции за обработка на числа с плаваща запетая. Още от самото начало Intel предлага математическия копроцесор 8087, който работи съвместно с 8086/8088 и ускорява изчисленията с плаваща запетая.

Повечето производители на компилатори създават емулиращи програмни библиотеки, които позволяват на програмиста да използва един и същ програмен интерфейс независимо дали компютърът, на който се изпълняват програмите, има математически копроцесор или няма такъв. С течение на времето се появяват копроцесорите 80187, 80287, 80387 и 80487. С въвеждането на процесора 486, математическият копроцесор става част от процесорното ядро, а инструкциите за обработка на числа с плаваща запетая (наричани още x87 инструкции) стават неразделна част от x86 архитектурата.

1. **MMX.**

През 1996 г. Intel въведе ново разширение на x86 архитектурата с пускането на процесора Pentium MMX. MMX означава Matrix Math Extensions (разширения за изчисления с матрици), но по-късно съкращението започна да се дешифрира като Multi-Media Extensions (мултимедийни разширения).

MMX разширенията позволяват да се изпълни една и съща математическа операция (например събиране, изваждане и т.н.) едновременно върху няколко цели числа, пакетирани в някои от MMX регистрите.

Въпреки широката рекламна кампания от страна на Intel, MMX инструкциите дълго време не бяха използвани в приложните програми, а когато такава поддръжка все пак се появи, резултатите не бяха особено впечатляващи. Някои от причините за този скромен успех са:

За да не се налагат промени в операционните системи, MMX регистрите всъщност са преименуваните x87 регистри за работа с числа с плаваща запетая. Това не позволява да се използват едновременно MMX и x87 инструкции, а бавното превключване между двата режима допълнително влошава нещата.

MMX инструкциите дълго време можеха да бъдат използвани само на асемблер. Поддръжката в програмните езици от високо ниво бе ограничена.

MMX инструкциите оперират само върху цели числа. Повечето съвременни мултимедийни приложения изискват работа с числа с плаваща запетая.

Все пак, MMX инструкциите станаха част от x86 архитектурата и се поддържат от всички съвременни реализации на тази архитектура.

1. **3DNow!**

През 1998 г. с пускането на пазара на процесора K6-2, AMD въведе нов набор от инструкции, наречен 3DNow!. 3DNow! инструкциите са SIMD инструкции, подобни на MMX, но боравещи и с числа с плаваща запетая. Малко по-късно Intel въведе подобен (но по-разширен и несъвместим с 3DNow!) набор от SIMD инструкции, наречени SSE.

В процесорите K6-2+, K6-III и първите Athlon, AMD добави още 19 инструкции за предварително зареждане на данни и др. подобни. Разширеният по този начин набор от инструкции получи името Enhanced 3DNow! (наричани още Extended 3DNow! и 3DNow+). С процесора Athlon XP, AMD въведе третото поколение SIMD инструкции, наречени 3DNow! Professional, като единствената съществена разлика бе поддръжката на SSE инструкциите на Intel.

Въпреки че 3DNow! инструкциите се появиха почти година преди SSE инструкциите на Intel, те не получиха широка подкрепа от разработчиците на софтуер, най-вече поради ниския пазарен дял на AMD. Въпреки че отстраняват най-големия недостатък на MMX (невъзможността за работа с числа с плаваща запетая), 3DNow! инструкциите отстъпват по възможности на SSE инструкциите, които използват нови 128-битови регистри и могат да оперират едновременно върху четири 32-битови или две 64-битови числа с плаваща запетая.

В крайна сметка 3DNow! инструкциите загубиха битката с SSE и в момента се поддържат само от процесорите на AMD и VIA и от години не са били добавяни нови инструкции към тях. Като се добави фактът, че AMD процесорите вече поддържат и SSE, бъдещето на 3DNow! изглежда мрачно.

1. **SSE, SSE2, SSE3.**

През 1999 г. Intel пусна процесора Pentium III, поддържащ нови SIMD инструкции, наречени SSE (Streaming SIMD Extensions). Също като 3DNow! инструкциите, SSE позволяват паралелната обработка на повече от едно число с плаваща запетая. С въвеждането на SSE, Intel успя да отстрани най-големите слабости на MMX и последвалите ги 3DNow! инструкции.

SSE инструкциите получиха широко разпространение (във всеки случай много по-широко от MMX и 3DNow!) и в крайна сметка AMD добавиха поддръжка за SSE инструкциите в техния процесор Athlon XP и следващите го процесори.

С въвеждането на процесора Pentium 4, Intel добави множество нови инструкции към SSE, създавайки SSE2. SSE2 инструкциите се поддържат от процесорите Athlon 64 на AMD.

С появата на последната ревизия на Pentium 4, Intel въведоха нови допълнения към SSE, наречени SSE3. SSE3 инструкциите се поддържат от последните модели Athlon 64 на AMD.

С течение на времето, SSE, SSE2 и SSE3 инструкциите се наложиха като дефакто стандартни SIMD инструкции във x86 архитектурата, за разлика от 3DNow! инструкциите, които се поддържат само от процесорите на AMD.

1. **x86-64.**

В продължение на почти 20 години (от пускането на пазара на процесора Intel 386 до 2003 г.) наборът от инструкции на x86-съвместимите процесори бе 32-битов. С течение на времето това започна да се превръща в тясно място, защото 32-битовият процесор не може да адресира повече от 4 GB (232 байта) памет, което се оказва недостатъчно за някои приложения. Intel временно облекчи проблема с модификации в начина на адресиране, но те нито са елегантни, нито особено ефективни.

Гледната точка на Intel по това време беше, че x86 архитектурата вече е изживяла времето си и трябва да бъде замена с перспективната VLIW архитектура IA-64, разработвана от години съвместно с компанията HP. Въпреки теоретичните предимства на тази архитектура, нейната производителност при работа със съществуващите 32-битови приложения е разочароваща, а цената – главозамайваща.

В този момент AMD реши да разработи 64-битови разширения за x86 архитектурата, които бяха кръстени AMD64. Естествено, повечето компании не пожелаха да използват наименованието AMD64 от страх да не предизвикат Intel и в крайна сметка възприеха по-неутралните обозначения x86-64 и x64.

Въпреки коментарите на Intel (както се оказа впоследствие – до голяма степен оправдани), че 64-битовите разширения не са необходими на болшинството от потребителите и няма да бъдат необходими поне още 4 – 5 години, големите компании Microsoft, Sun Microsystems и дори свободната операционна система Linux подкрепиха новата архитектура. Изправени пред тези факти, Intel бяха принудени да приемат като стандарт 64-битовите разширения на AMD, но, разбира се, не под името AMD64, а под името EM64T.

В наши дни поддръжката за x86-64 се превърна в стандарт за процесорите на AMD и Intel, но броят на приложните програми, които реално използват 64-битов код, все още е не голям.

1. **Симулация на х86 с Multi2sim**

1. Симулация на х86

Симулацията на програма за x86 гост може да бъде разделена на две основни модула: функционална симулация (емулация ) и тайминг ( подробна или архитектурна симулация).

Чрез изпълними ELF ( изпълними и с възможност за свързване printf ) файлове, функционалната симулация симулира изпълнението на програма на архитектура x86 . Подробната симулация предоставя модел на

хардуерни структури на x86-базирани машина. Тя използва тайминг и използва статистически данни за всеки хардуерен компонент на х86, в зависимост от поддържаните от емулатора инструкции.

Функционална симулация на x86

Изпълнимият файл при зареждането на Multi2Sim е m2s , който е унифициран инструмент както за функционална така и за подробна симулация. Избираемата команда --x86-sim functional позволява x86 емулация (стандартна конфигурация). Тази конфигурация позволява изпълнението на емулацията. Приема като входни параметри един или повече ELF файлове, и емулира изпълнението им, осигурявайки набор от статистически данни, основаващи се на код на изпълнима програма гост. Основните функции на емулатора могат да бъдат класифицирани в изпълнение на програма , иемулация на х86 инструкции , и емулация на system call(kernel call).

• Изпълнение на програма. Изпълнението на дадена програма гост

Представлява виртуално изображение на паметта и набор от стойности на логически регистър на х86(ЕАХ, ЕВХ и т.н.)

Приложният бинарен интерфейс (ABI) на линукс определя първоначална стойност и за двете. Първоначалното състояние се определя най-вече от ELF двоична програма и от потребителя(чрез команден ред) по време на процеса, наречен изпълнение на програма. В една реална система, изпълнението на програма се извършва от операционната система след execv system call или някой от неговите варианти. В симулационна среда, Multi2Sim е отговорен за изпълнението на програмата. Тома се осъществява, чрез следните стъпки:

- Първо, двоичен код за x86 се парсва с ELF парсва. Един ELF файл съдържа х86 инструкции и инициализирани данни, съвместно с виртуалния адрес, от където те трябва да бъдат първоначално заредени. Програмния зареждач получава виртуалния адрес на всеки ELF раздел, след което го копира в нужната част от виртуалната памет.

- Инициализация на стека. Стекът е участъг от виртуалния образ на паметта на гост програмата, към който сочи регистър ESP . Първоначално той съдържа набор от програмни библиотеки копирани от ELF файл, последвани от масив от променливи и аргументи зададени от потребителя, чрез командния ред.

- Инициализират се й86 регистрите. ESP регистърът сочи към последния добавен елемент в стека, а регистъра EIP към мястото в паметта където се съдържа кода, чакащ да бъде изпълнен(първият на опашката).

• емулация на x86 инструкции. След като първоначалния образ на новия контекст е готов, неговата емулация може да започне. Итеративно, функционалната симулатора чете последователност от байтове в гост

адреса на паметта, сочен от регистър EIP . След това инструкцията се

декодира чрез x86 декодера на Multi2Sim и Disassembler. Инструкцията се емулира след което регистър EIP започва да сочи към следващата чакаща инструкция.

• емулация на системните повиквания. Специална инструкция е софтуерното прекъсване на x86 - инструкция INT . По-конкретно, инструкция " INT 0x80 " се използва за извършване на системни функции

В повечето случаи, Multi2Sim ще трябва да извърши същия host system call като програмата гост, като това се извършва преди и след обработка на аргументите и резултатът, съответно. Например, когато програмата гост работи на open system call, тя предоставя в специфичен x86 регистър

указател към низ съдържащ път към системното повикване, и очаква файлов дескриптор като връщана стойност, също в определен x86 регистър.

The Подробна симулация

Командата --x86-SIM подробно позволява x86 подробна симулация. Подробната симулация активира модела на x86 CPU конвейер с поддръжка на прогнози и изпълнение. Първата фаза на конвейер (извличащ етап) взаимодейства с емулаторния модул или кернел библиотеката на Multi2Sim, чрез използване на прост интерфейс. Итеративно, m2s кара емулатора да емулира следващата гост x86 инструкция и да върне някаква информация за него. Въз основа на тази информация, m2s може разбера кои хардуерни структури се активират и изпълни подробната симулация. Функционалната симулатор знае по всяко време, коя точно е следващата инструкция за изпълнение. Този процес се моделира в m2s както следва.

При съвпадение на прогнозата за адреса на следващата инструкция и състоянието на емулатора, както функционалнат, така и подробната симулация се синхронизират. Въпреки това, модула misprediction ще накара

m2s да извлече погрешни инструкции. За това m2s премества пойнтера на EIP регистъра във функционалния симулатор, което автоматично запазва състояние то на инструкцията. m2s извличане инструкции принуждавайки функционалния симулатор, да ги изпълни, докато mispredicted позволи преминаване към следващата фаза на конвейера. Когато това стане m2s се връща към запапазените по-рано инструкции.

2. Обобщение на x86 Статистика

Обобщението на статистиката x86 е представена в края на симулацията, ако е налице ефективна функционална или подробна симулация за архитектурата x86. Тя се изхвърлят в стандартния изход грешка, следвайки

INI файлов printf, в раздел, наречен [x86] . За всички архитектури, следните x86-специфични променливи присъстват:

• Нишки . Максимален брой нишки, които могат да работят едновременно в x86 гост програми по време на симулацията.

• Памет . Максимален размер на паметта в байтове, използвани общо от всички нишки.

Следните променливи присъстват в статистиката само за подробна x86 симулация:

• FastForwardInstructions . Брой на x86 макро-инструкции превъртя напред с функционална симулация, както е посочено в конфигурационния файл x86.

• CommittedInstructions . Брой на x86 макро-инструкции, извършени във всички x86 конвейери, на всички ядра и хардуерни нишки. Тази стойност е винаги равна на или по-ниска от брой пример на емулираните инструкции

• CommittedInstructionsPerCycle . Брой на x86 инструкции, извършено в рамките на цикъла, изчислени като частното на CommittedInstructions и циклите.

• CommittedMicroInstructions . Брой на микро-инструкции, извършени във всички x86 конвейери, при отчитане на всички ядра и хардуерни нишки. Тъй като всеки x86 инструкции генерира най-малко една микро-инструкция, тази стойност е винаги равно или по-голямо от CommittedInstructions.

• CommittedMicroInstructionsPerCycle . Брой на микро-инструкции, извършени в рамките на цикъла, изчислен като отношение на CommittedMicroInstructions и цикли .

• BranchPredictionAccuracy . Клон точност предиктор, изчислен като броя на правилно предсказани клонове, разделен на общия брой на клоновете (т.е. клон микро-инструкции).

• x86LastInst . Емулацията на x86 програма е изпълнена последната инструкция, която е посочена в командния ред като --x86-миналата-инс <инс> .

• x86MaxInst . Максималният брой на x86 инструкции е било постигнато, както е посочено в команден ред опцията --x86-макс-инс <бр> . В функционална симулация, това ограничение се прилага под printfа на

Максималният брой на пример за подражание инструкции. В подробна симулация, ограничението е дадена в брой

на извършени (не-спекулативни) x86 инструкции.

• x86MaxCycles . Максималният брой на x86 симулационни цикъла е било постигнато, както е посочено в опции на командния ред --x86-MAX цикли <цикли> . Само за детайлна x86 симулация.

3. Съставяне и Симулиране Source Code

Статично и динамично свързване

При съставяне на програма, има два основни подхода за свързване на обектни файлове в изпълнима форма, наречени динамично и статично свързване.

• Статично свързване . GCC компилаторът може да бъде конфигуриран да генерира статично свързан изпълним код чрез добавяне на -static опция в командния ред. В този случай, кодовете на която и да е споделена библиотека

използвана от програмата (като математическата библиотека, библиотеката на POSIX нишките и т.н.) се свързват заедно с програмата. Това включва, например, прилагане на PRINTF функция, заедно с много други процедури инициализиране на програмата. Дори за простата “Hello world” програма, се генерира огромен изпълним файл. Предимството от това е, че този файл може да се използва на всяка Linux машина съвместима с версия на ядрото, независимо от версиите на останалите инсталирани пакети за развитие и библиотеки.

• динамично свързване. Зададено по подразбиране за GCC . Когато програмата е свързан динамично, кодът на библиотеката не се прилага за крайния изпълним файл. Вместо това, всяко позоваване на външен

символ, като PRINTF функция, се оставя неизпълнен първоначално. Компилаторът добавя в изпълним код за зареждане на динамичен зареждач , който също е динамична библиотека присъстваща във вашия система, обикновено под /etc директория. Когато програмата се изпълнява, самия код копира кода на динамичния зареждач. След това, на динамичния зареждач се опитва да намери всички споделени библиотеки, изисквани от вашата програма и зарежда своя код в процеса на изобразяване. Накрая, управлението се насочва обратно към програмния код, който продължава с други инициализиращи действия.

Разлики

Този раздел показва практически пример за разликите в статичното и динамичното свързване на кодове в Multi2Sim. Пример за “Hello world” програма:

# include <stdio.h>

int main ()

{

printf ( "Hello World \ Н");

return 0;

}

Първо, нека се генерира статично свързана версия на програмата в Multi2Sim. Добра представа за поведението на програмата ще бъде дадено от системните повиквания. Подобно на изхода

предоставена от strace инструмента, следа от извършените системни повиквания, техните аргументи, и стойностите за връщане

може да се получи с Multi2Sim чрез използване на опция на командния ред --x86-debug-syscall <файл> .

*$ gcc hello.c -o hello -static*

*$ m2s --x86-debug-syscall stdout hello*

*syscall ’newuname’ (code 122, inst 418, pid 1000)*

*syscall ’brk’ (code 45, inst 738, pid 1000)*

*syscall ’set\_thread\_area’ (code 243, inst 850, pid 1000)*

*[...]*

*syscall ’open’ (code 5, inst 911, pid 1000)*

*filename=’/dev/urandom’ flags=0x0, mode=0x0*

*return=0x3*

*syscall ’read’ (code 3, inst 932, pid 1000)*

*guest\_fd=3, pbuf=0xfffdffbd, count=0x3*

*return=0x3*

*syscall ’close’ (code 6, inst 948, pid 1000)*

*guest\_fd=3*

*return=0x0*

*[...]*

*syscall ’fstat64’ (code 197, inst 7973, pid 1000)*

*fd=1, pstatbuf=0xfffdfe58*

*return=0x0*

*syscall ’mmap2’ (code 192, inst 8028, pid 1000)*

*addr=0x0, len=4096, prot=0x3, flags=0x22, guest\_fd=-1, offset=0x0*

*prot={PROT\_READ|PROT\_WRITE}, flags={MAP\_PRIVATE|MAP\_ANONYMOUS}*

*return=0xb7fb0000*

*syscall ’write’ (code 4, inst 8881, pid 1000)*

*guest\_fd=1, pbuf=0xb7fb0000, count=0xc*

*buf="hello world\n"*

*return=0xc*

*syscall ’exit\_group’ (code 252, inst 9475, pid 1000)*

Може да се отбележи, че програмата извлича информация информация от кернела ( newuname ), актуализира размера на heap-а и разпределя на памет ( BRK , mmap2 ), получава случайни цели числа за инициализация ( open, read, close ), получава информация за стандартния изход ( fstat64 ),

показване на “Hello world!“ низ ( write ), и излизане от програмата ( exit\_group ). Сега нека се опитаме на същото е и с динамично свързана версия на програмата:

*$ gcc hello.c -o hello*

*$ m2s --x86-debug-syscall stdout hello*

*syscall ’brk’ (code 45, inst 1122, pid 1000)*

*syscall ’newuname’ (code 122, inst 2499, pid 1000)*

*syscall ’open’ (code 5, inst 6906, pid 1000)*

*filename=’/etc/ld.so.cache’ flags=0x0, mode=0x0*

*return=0x3*

*syscall ’fstat64’ (code 197, inst 6931, pid 1000)*

*fd=3, pstatbuf=0xfffdf924*

*return=0x0*

*syscall ’mmap2’ (code 192, inst 6967, pid 1000)*

*addr=0x0, len=61684, prot=0x1, flags=0x2, guest\_fd=3, offset=0x0*

*prot={PROT\_READ}, flags={MAP\_PRIVATE}*

*host mapping created for ’/etc/ld.so.cache’*

*host\_ptr=0xb77e9000*

*host\_fd=5*

*return=0xb7fa0000*

*syscall ’close’ (code 6, inst 6983, pid 1000)*

*guest\_fd=3*

*return=0x0*

*syscall ’open’ (code 5, inst 8171, pid 1000)*

*filename=’/lib/libc.so.6’ flags=0x0, mode=0x0*

*return=0x3*

*syscall ’read’ (code 3, inst 8192, pid 1000)*

*guest\_fd=3, pbuf=0xfffdfa58, count=0x200*

*return=0x200*

*[...]*

*syscall ’mprotect’ (code 125, inst 80556, pid 1000)*

*start=0xb7f9a000, len=0x2000, prot=0x1*

*return=0x0*

*syscall ’write’ (code 4, inst 91187, pid 1000)*

*guest\_fd=1, pbuf=0xb7faf000, count=0xc*

*buf="hello world\n"*

*return=0xc*

*syscall ’exit\_group’ (code 252, inst 92139, pid 1000)*

*return=0x0*

Може да се отбележи, че програмата издава няколко open system calls

/etc/ld.so.cache и /lib/libc.so.6 , последвани от прочетените , mmap и mprotect calls, насочени към актуализиране на памет на гост кода и разрешения за изпълнението му.

Натоварването на динамично свързване може да се наблюдава в стойността на променливите Инструкции в раздел [x86] на статистиката в края на симулацията и в двата случая. Докато статично свързаната програма изпълнява около 10K инструкции, динамично свързан версия изпълнява около 100K инструкции

при добавяне на допълнителните за инициализация.

Съобщения за грешки

Много обширна имплементация в Multi2Sim и за двата набора от x86 инструкции и повечето системни повиквания на Unix. Изпълнението на тези функции се базира, както на използване на инструкции, така на тестващи изпълними програми. За момента, много различни версии на GCC са използвани за генериране на изпълними програми, включително

различни версии на споделени библиотеки и Линукс ядра. По време на този процес се добавят нови инструкции и системни повиквания, за осигуряване на доста стабилна и пълна версия на функционалната симулация на програма.

Въпреки това е възможно вашата специфична комбинация от споделени библиотеки и GCC компилатор да генерира изпълним файл, който включва специална инструкция или системно обаждане, което не се поддържа от Multi2Sim.

Когато се опитате да извършите функционална симулация на неподдържан файл, програмата изведе съобщение за грешка, подобно на това:

fatal: context 1000 at 0x0804f800: instruction not implemented: 0f a1 b0 ...

— or —

fatal: not implemented system call ’tgkill’ (code 270) at 0x0802010

4. Конвейерен процесор

Подробната симулация на x86 конвейера може да бъде активирана с опция --x86-sim detailed. Когато е активна, опция --x86-config може да се използва, за да премине на конфигурационния файл на x86 конвейер, докато вариант

--x86-config евентуално ще изхвърли подробен файл с подробна. Конфигурационният файл x86 е в INI файлов формат, където раздел [General] позволява използването на тези тези променливи (между другото):

• Frequency . Честота в MHz на x86 конвейерите. Всяка латентност, дадена в цикли в рамките на x86 конфигурационен файл ще се приеме в рамките на тази честотна област. Стойността по подразбиране за тази променлива е 1000 (= 1 GHz).

Фигура 2.2: x86 суперскаларна процесор конвейер.

Има 6 фази в суперскаларния процесор: fetch, decode, dispatch, issue, writeback, and commit.

В етапа на извличане(fetch), инструкции се прочитат от инструкция или трейс кеша. В зависимост от произхода си, те са нареждат в опашката за извличане или на опашката за следи .

В етап декодиране, инструкциите се прочитат от тези опашки и се декодират, ако е необходимо. След това, uops са поставени в опашката UOP .

Етапите извличане и декодиране образуват предния край на конвейера.

Етапът на изпращане отнема uops от UOP опашката, преименува техния източник и регистри за дестинация, и

ги поставя в буфера за повторна поръчка (ROB) и на опашката за инструкции (IQ) или load-store опашката (LSQ).

Етапът issue е отговорен за търсене на инструкции от IQ и LSQ с готови операнди.

Когато и UOP завърши,

Writeback етапът запазва операнда за адресация обратно в регистровия файл.

Накрая, завършени uops в началото на ROB се взимат от commit етапът и изпълнението на инструкцията е потвърдено.

5. CISC Инструкции - декодиране

Архитектурата на x86 дефинира набор на CISC инструкции [ 3 ]. Всяка една инструкция определено в x86 ISA е с размер между 1 и 15 байта, и може да изпълнява широк набор от различни действия. Някои инструкции включват няколко сложни действия, като например четене на паметта, последвано от аритметично изчисление и запис в паметта. Такива сложни инструкции са вътрешно декодирани като отделни микро-инструкции ( uops ).

Всеки UOP има максимум четири възможни изходни, образувани от логическите регистри и флагове за състояние.

Максивалния брой входове UOP е три, като се има предвид логическите регистри и флагове за състоянието, сред които най-много два могат да бъдат логически регистри.

• Mov EDX, DWORD PTR [EBX-0x4]

Тази инструкция прочита стойността на 32-бито адрес, посочен от регистър EBX минус 4 и запазва резултата в регистър EDX . Генерираният микрокод трябва изчислява ефективния адрес на паметта въз основа на EBX регистър, зарежда стойността на този адрес съхранява резултата в EDX .

Това е генерираната последователност на uops:

effaddr ea/ebx

load edx/ea [0x8004300,4] Аргументите за uops са логическите регистри. Те са представени като

odep1, odep2, ... / idep1, idep2, ... , където idepXX е вход, и odepYY е изход. Регистъра ea е вътрешен логически регистър и се използва за съхраняване на резултатите от ефективните адресни.

• add DWORD PTR [EBX + 0x8], EAX

Това е пример за сложна инструкция, изискваща да се зареди със стойност от място в паметта, сочещо EBX плюс 8, да се добави тази стойност със съдържанието на регистър EAX , и да се съхрани резултата обратно в паметта.

The data регистърът е друг временен регистър, използван за съхраняване на данни за uops памет.

**effaddr ea/ebx**

**load data/ea [0x8004300,4]**

**add data/data,eax**

**store -/data,ea [0x800430,4] • rep movsb**

На movsb копира инструкция един байт от паметта, сочена от регистър ESI в паметта, сочена от EDI. Допълнителният префикс rep е причината за тези действия, които да се повтарят толкова пъти, колкото стойността на

регистъра ECx, докато във всяка итерация на стойностите на ESI и EDI се увеличава (или намалява, в зависимост от стойността на флаг DF ) с 1. Едно повторение на цикъла генерира следния микрокод:

*load aux/edi [0x80d789f,1]*

*store -/esi,aux [0x80d789c,1]*

*add edi/edi,df*

*add esi/esi,df*

*sub ecx/ecx*

ibranch -/ecx Инструкцията load зарежда в регизтъра AUX стойността прочетена от регистър EDI , която е съхранена в следващия адрес на ESI . След това, регистрите ESI и EDI се увеличават като се използват две add uops, стойността в ECx се намалява и се изпълнява първато инструкция от микрокода в зависимост от стойността на ecx.

1. **Използвана литература:**

<https://bg.wikipedia.org/wiki/X86>

<https://bg.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BB%D0%B5%D0%BD_%D1%80%D0%B5%D0%B6%D0%B8%D0%BC>

<https://bg.wikipedia.org/wiki/%D0%97%D0%B0%D1%89%D0%B8%D1%82%D0%B5%D0%BD_%D1%80%D0%B5%D0%B6%D0%B8%D0%BC>

<https://bg.wikipedia.org/wiki/X87>

<https://bg.wikipedia.org/wiki/MMX>

<https://bg.wikipedia.org/wiki/3DNow>!

<http://www.referati.org/ciscrisk-intel-x86-mikroprocesori/22511/ref>

<https://domashke.net/referati/referaty-po-informatike/referat-programmnaya-model-processorov-semejstva-x86>