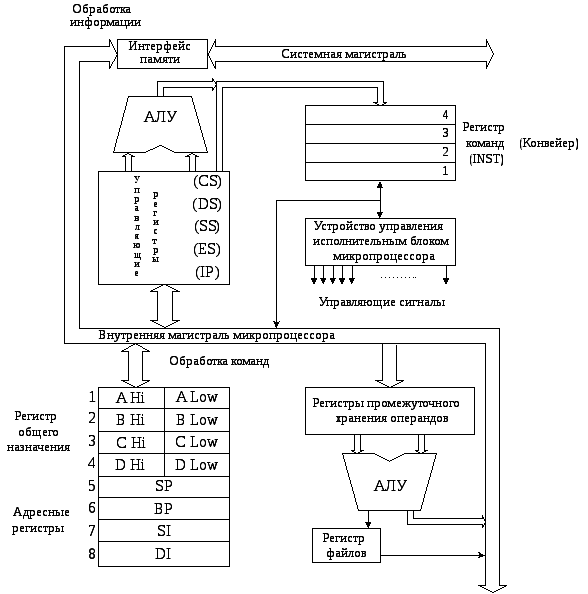
Структурна схема мікропроцесора



команд

BusInterface

Unit

(АЛУкоманд+

регістри упр)

ExecUnit

(ар.блок+

pегістри P3П)

**Регістри РЗП (РОН):**

в **x86-16** - вісім 16-бітних регістрів - AX, BX, CX, DX, BP, SI, DI, SP

+16-бітний покажчик (вказівник) IP та 16-бітний регістр флагів FLAGS

в **x86-32** - вісім 32-бітних регістрів - ЕAX, ЕBX, ЕCX, ЕDX, ЕBP, ЕSI, ЕDI, ЕSP

+32-бітний покажчик (вказівник) ЕIP та 32-бітний регістр флагів ЕFLAGS

в **х64** - архітектурі  
16 цілочисельних 64-бітних регістрів ЗП (RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8 — R15),  
**8 80-бітних регістрів з плаваючою крапкою (ST0 — ST7),**  
8 64-бітних регістрів Multimedia Extensions (MM0 — MM7, мають спільний простір з регістрами ST0 — ST7),  
16 128-бітних регістрів SSE (XMM0 — XMM15),  
 +64-битний покажчик (вказівник) RIP та 64-бітний регістр флагів RFLAGS.

**Сегментні регістри**

CS, DS, ES, SS / GS, FS (16 – бітні)

**4 системні регістри**

GDTR, LDTR, IDTR, TR + 4 рег управління (CR0-CR4) + 6 рег налагодження

### Регистры

Програмна модель МП містить вісім регістрів загального призначення, шість регістрів сегментних адрес, вказівник команд, регістр системних флагів, регістри системних адрес, чотири регістра управління і шість регістрів налагодження.

Вісім регістрів ЗП мають довжину в 32 біти і використовуються для зберігання адрес або даних. Вони підтримують операнди-дані довжиною 1, 8, 16, 32 і (при використанні двох регістрів) 64 біти; бітові поля від 1 до 32 біт; операнди-адреси довжиною 16 і 32 біти. Ці регістри (в 32 розрядній архітектурі) мають назву EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP. Доступ до молодших 16 бітів цих регістрів здійснюється незалежно при використанні відповідних імен 16-бітних регістрів: AX, BX, CX, DX, SI, DI, BP и SP. Також може використовуватися індивідуально молодший (біти 0-7) і старший (біти 8-15) байти регістрів AX, BX, CX, DX. Їм відповідають позначення AH, DH, CH, BH и AL, DL, CL, BL.

МП включає шість безпосередньо доступних 16-бітних регістрів сегментів. З кожним сегментним регістром асоційований програмно-недоступний кеш дескриптора відповідного сегмента, що містить базову адресу сегмента в лінійному адресному просторі, межа сегмента і атрибути сегменту. Цей кеш заповнюється при завантаженні значення в сегментний регістр. У реальному режимі межа сегмента завжди 0FFFFh, атрибути ігноруються, а базова адреса обчислюється зсувом значення селектора на 4 біти ліворуч. У захищеному режимі кеш заповнюється відповідними значеннями з дескрипторної таблиці.

Не всі сегментні регістри рівнозначні. Регістр CS зберігає селектор сегмента коду. МП витягує чергову інструкцію для виконання, формуючи логічну адресу з селектора в CS і зміщення в регістрі EIP. Значення цього регістру можна змінити безпосередньо, воно змінюється в командах межсегментного переходу (FAR JMP), межсегментного виклику (FAR CALL), при виклику обробника переривання (INT) і при поверненні з далекої процедури (RETF) або обробника переривання (IRET).

Регістр SS зберігає селектор сегмента стека. Стек використовується для передачі параметрів підпрограм і для збереження адреси повернення при виклику підпрограми або обробника переривання. Вершиною стека вважається байт, логічна адреса якого утворюється з селектора в регістрі SS і зсуву в регістрі ESP. Програма може безпосередньо змінити значення SS, що дає їй можливість перемикатися між кількома стеками.

Регістри DS, ES, FS і GS зберігають селектори сегментів даних. Якщо інструкція звертається до пам'яті, але містить тільки зміщення, то вважається, що вона звертається до даних в сегменті DS. Сегмент ES може використовуватися без явного вказівки в цепочечних командах. Сегменти FS і GS використовуються при зверненні до пам'яті тільки при явному використанні в інструкції префіксів цих сегментів.

Покажчик команд (EIP) є ​​32-розрядним регістром. Він містить зміщення наступної команди, яка підлягає виконанню. Відносний адреса відраховується від базового адреси сегмента виконується завдання. Покажчик команд безпосередньо недоступний програмісту, але він змінюється явно командами управління потоком, переривань і винятками (JMP, CALL, RET, IRET, команди умовного переходу). Отримати поточне значення EIP можна, якщо виконати команду CALL, а потім прочитати слово на вершині стека.

Регістр системних флагів EFLAGS містить групу флагів стану, управління і системних прапорів. Молодші 16 біт регістра є 16-розрядний регістр флагів і стану МП 8086, званий FLAGS, який найбільш корисний при виконанні програм для МП 8086 і Intel-286. Деякі з флагів можуть бути змінені спеціально призначеними для цієї мети інструкціями. Для зміни або перевірки групи прапорів можна скористатися наступними командами:  
• LAHF / SAHF - завантаження / збереження молодших 8 бітів регістра флагів в регістрі AH;  
• PUSHF / POPF - переміщення / витяг з стека молодших 16 бітів регістра флагів;  
• PUSHFD / POPFD - переміщення / витяг з стека 32-бітного регістра EFLAGS.

Регістри управління сегментованої пам'яттю, відомі також як регістри системних адрес, вказують на структури даних, які керують механізмом сегментованої пам'яті. Вони призначені для посилань на таблиці або сегменти, що підтримуються моделлю захисту МП.  
• Регістр глобальної дескрипторної таблиці (GDTR). Містить 32-бітний лінійний адреса і 16-бітну кордон глобальної дескрипторної таблиці. Значення цього регістру можна завантажити / зберегти за допомогою привілейованих інструкцій LGDT / SGDT. У реальному режимі цей регістр не використовується. Перед переходом в захищений режим в цей регістр слід завантажити коректні значення.  
• Регістр локальної дескрипторної таблиці (LDTR). Містить 16-розрядний селектор локальної дескрипторної таблиці. З регістром пов'язаний програмно-недоступний кеш дескриптора для зберігання базової адреси, межі і атрибутів відповідної дескрипторної таблиці. Значення цього регістру можна завантажити/ зберегти за допомогою привілейованих інструкцій LLDT / SLDT. У реальному режимі цей регістр не використовується, і спроба звернення до нього генерує особливий випадок "недійсний код операції" (виняток # 6). З кожним завданням в захищеному режимі може бути пов'язана своя локальна дескрипторній таблиці, тому селектор LDT зберігається в TSS і автоматично завантажується при перемиканні задач.  
• Регістр таблиці дескрипторів переривань (IDTR). Вказує на таблицю точок входу в програми обробки переривань. Регістр містить 32-розрядний лінійний базовий адресу та 16-бітний межа таблиці. Значення цього регістру можна завантажити / зберегти за допомогою привілейованих інструкцій LIDT / SIDT. При ініціалізації МП базовий адресу IDT встановлюється в 0, а межа - 0FFFFh. У реальному режимі ця таблиця зберігає 4-байтниє вектори переривань, а в захищеному - 8-байтниє дескриптори шлюзів обробників переривань і виключень. Це єдиний реєстр серед перерахованих, який використовується в реальному режимі.  
• Регістр завдання (TR). Вказує на інформацію, необхідну МП для визначення поточного завдання. Регістр містить 16 біт селектор дескриптора сегмента стану завдання. З регістром пов'язаний програмно-недоступний кеш дескриптора TSS для зберігання базової адреси, межі і атрибутів відповідного сегмента стану завдання. Значення цього регістру можна завантажити / зберегти за допомогою привілейованих інструкцій LTR / STR. У реальному режимі цей регістр не використовується, і спроба звернення до нього генерує особливий випадок "недійсний код операції" (виключення #6).

МП має чотири 32-розрядних регістра управління CR0-CR4, в яких зберігаються флаги стану МП або глобальні флаги. Разом з регістрами системних адрес ці регістри зберігають інформацію про стан МП, яка впливає на всі завдання в системі.

Шість доступних регістрів налагодження (DR0-DR3, DR6, DR7, регістри DR4 і DR5 зарезервовані) розширюють можливості налагодження. Вони встановлюють точки зупину за даними і дозволяють встановлювати точки зупинки по командах без модифікації сегментів програм. Регістри DR0-DR3 призначені для зберігання чотирьох лінійних адрес точок зупину. Регістр DR6 відображає поточний стан точок зупину. Регістр DR7 задає умову для точок зупинки.

**Режими роботи**

#### Реальний режим (Real Mode)

#### Режим системного управління (System Management Mode)

#### Захищений 16-розрядний режим (Protected Mode)

Захищений 32-розрядний режим (Protected Mode)

**Робота мікропроцесора по тактах**

* завантаження команди
* декодування операції
* декодування даних
* виконання інструкцій
* вивантаження результату

**Схема сегментної адресації пам’яті (реальний режим роботи МП)**

IP (код)

CS

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

DS/ES + BP, SI, DI (дані)

SS + SP (стек)

АДРЕСАЦІЯ команд: (CS):(IP) | (CS):(EIP)

АДРЕСАЦІЯ даних: (DS):(BX| BP | SI | DI) | (ES):(SI | DI)

(DS):(EBX| EBP | ESI | EDI) | (ES):(ESI | EDI)

АДРЕСАЦІЯ стекових даних: (SS):(SP) | (SS):(ESP)

Зміна адрес коду: адреса наступної команди (на прикладі ІР)

1. IP=IP+довжина інструкції
2. Зміна IP=> в межах сегмента NEAR-перехід
3. Зміна СS,IP=> за межами сегмента FAR-перехід

Зміна адрес стеку (на прикладі SР):

1. запис –(SP)=(SP)-2 + 2 байти за адресами (SS):(SP), (SS):(SP+1);
2. читання –2 байти за адресами (SS):(SP), (SS):(SP+1) + (SP)=(SP)+2 .

!!! Переповнення стеку.

**Формат команд мікропроцесора (15 байт)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| префікс | КОП (OpCode) | Mode R/M | SIB | Зсув | Операнд |
| 0/1 байт | 1/2 байт | 0/1 байт | 0/1 байт | 0/1/2/4 б. | 0/1/2/4 б. |

Префікси: - повтору REP, REPE/REPNE, REPZ/REPNZ

- зміни сегменту

- довжини інструкції

- блокування шини

Префікс - необов'язкова частина інструкції, яка дозволяє змінити деякі особливості її виконання. У команді може бути використано відразу кілька префіксів різного типу. Типи префіксів: командні префікси (префікси повторення) REP, REPE / REPZ, REPNE / REPNZ; префікс блокування шини LOCK; префікси розміру; префікси заміни сегмента.  
КОП - код операції.

Байт "Mod R / M" визначає режим адресації, а також іноді додатковий код операції. Необхідність байта "Mod R/M" залежить від типу інструкції.

Байт SIB (Scale-Index-Base) визначає спосіб адресації при зверненні до пам'яті в 32-бітному режимі. Необхідність байта SIB залежить від режиму адресації, що задається полем "Mod R/M".

Крім того, інструкція може містити безпосередній операнд і / або зсув операнда в сегменті даних.

На розмір інструкції накладається обмеження в 15 байт. Інструкція більшого розміру може вийти при некоректному використанні великої кількості префіксів. В IA-32 в такому разі генерують виключення #13.

Якщо інструкція мікропроцесора вимагає операнди, то вони можуть задаватися наступними способами:

*безпосередньо в коді інструкції (тільки операнд-джерело);*

*в одному з регістрів;*

*через порт введення-виведення;*

*в пам'яті.*

Для сумісності з 16-бітними процесорами архітектура IA-32 використовує однакові коди для інструкцій, що оперують як з 16-бітними, так і 32-бітових операндами. Нова архітектура передбачає також нові можливості в адресі для операнда в пам'яті. Як процесор буде вважати операнд або його адресу, залежить від ефективного розміру операнда і ефективного розміру адреси для даної команди. Ці значення визначаються на основі режиму роботи, біта D дескриптора використовуваного сегмента і наявності в інструкції певних префіксів.

Безпосередній режим адресації має на увазі включення операнда-джерела в код інструкції. Операнд може бути 8-бітових або 16-бітових, якщо значення ефективного розміру операнда - 16. Операнд може бути 8-бітових або 32-бітових, якщо значення ефективного розміру операнда - 32. Зазвичай безпосередні операнди використовуються в арифметичних інструкціях.

Регістровий режим адресації визначать операнд-джерело або операнд-приймач в одному з регістрів процесора або сопроцесора. У деяких випадках, (наприклад, в інструкціях DIV і MUL) можуть використовуватися пари 32-бітних регістрів (наприклад, EDX: EAX), утворюючи 64-бітний операнд.

Адресація через порт вводу-виводу має на увазі отримання операнда або збереження операнда через простір портів вводу-виводу. Адреса порту вводу-виводу або безпосередньо включається в код інструкції, або береться з регістра DX.  
Дуже поширений спосіб адресації операнда - адресація через пам'ять. Таким чином, може бути вказаний операнд-джерело або операнд-приймач. Слід зазначити, що процесор не дозволяє одночасно ставити обидва операнда через пам'ять (за винятком деяких ланцюгових команд).

Для отримання операнда з пам'яті процесору необхідно знати селектор сегмента і зміщення в сегменті. У деяких командах селектор може бути вказаний безпосередньо в коді інструкції. В інших випадках процесор може явно чи неявно використовувати значення одного з сегментних регістрів. Під неявним використанням сегментних регістрів мається на увазі те, що в залежності від призначення операнда процесор використовує певний сегментний регістр для звернення до пам'яті: CS -для вибору інструкцій; SS - для роботи зі стеком або звернення до пам'яті через регістри ESP або EBP; ES - для отримання адреси операнда-приймача в ланцюгових командах; DS - при всіх інших зверненнях до пам'яті. Явне використання сегментних регістрів можливе, якщо в код інструкції включається префікс зміни сегмента. Застосування ​​префікса зміни сегмента допустимо не для всіх команд: не можна змінювати сегмент для команд роботи зі стеком (завжди використовується SS); для ланцюгових команд можна змінювати сегмент тільки операнда-джерела (операнд-приймач завжди адресується через ES).

Логічна адреса команд, даних та стеку визначається парою регістрів (СР):(РЗП). Для формування фізичної адреси потрібно враховувати режим роботи МП. Мах розмір ОП, який можна адресувати з використанням АШ, називається адресним простором.

Адресний простір у реальному режимі роботи 1 МБ, ширина шини – 20 розрядів. ОП розбивається на блоки, які називаються сегментами, початкова адреса кожного з яких кратна 16 (параграф).

Лінійна адреса = BASE\_ср \* 16 + DISPLACEMENT\_рзп

Ефективна або фізична адреса EA в реальному режимі дорівнює лінійній.

Приклад запису інструкцій мовою асемблера

mov ax,0123

//код mov ax =B8 (1 байт, найменша адреса), за ним дані [al=23 | ah=01]

// припустимо (CS)=13C6 (IP)=0100

// довжина команди 3 байти => нові значення (CS)= 13C6 (IP)=0103

add ax,25

//код add ax =05, за ним дані [25 | 00]

Приклад програми мовою асемблера

text segment

assume Cs:text,DS:data

begin: mov ax,data

mov ds,ax

mov ax,0123h

add ax,25

mov res,ax

;Вивід рядка

mov ah,09h

mov dx,offset mesg

int 21h

;Завершення роботи

mov ah,4Ch

mov al,0

int 21h

text ends

data segment

res dw ?

mesg db 'OK!$'

data ends

