**Процесори із стековою архітектурою. Сопроцесор**

На сьогоднішній день найбільше поширення набули наступні структури команд: одноадресні (1A), двоадресні (2A), триадресні (3A), безадресні (БА), команди з великою довжиною слова (VLIW – ВДС) (мал. 4):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1А ~ |  | КОП | А1 |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 2А ~ |  | КОП | А1 | А2 |  |  |  |
|  |  |  |  |  |  |  |  |
| 3А ~ |  | КОП | А1 | А2 | А3 |  |  |
|  |  |  |  |  |  |  |  |
| БА ~ |  | КОП |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| ВДС ~ |  | КОП | Адреси | Тегі | Дескриптори | |  |

Мал. 4. Структури команд різних процесорів

Причому операнд може вказуватися як адресою, так і безпосередньо значенням в структурі команди.

Стек – це область оперативної пам'яті, яка використовується для тимчасового зберігання даних і операцій. Доступ до елементів стека здійснюється за принципом LIFO (last in, first out) – першим ввійшов, останнім вийшов. Крім того, доступ до елементів стека здійснюється тільки через його вершину, тобто користувачу "видно" лише той елемент, який поміщений в стек останнім.

Принцип функціонування процесора із стековою організацією пам'яті.

При виконанні різних обчислювальних процедур процесор використовує або нові операнди, що дотепер не вибиралися з пам'яті комп'ютера, або операнди, що приймали участь у попередніх операціях. У процесорах з класичною структурою звернення до будь-якого операнда вимагає циклу пам'яті.

Розглянемо приклад. Нехай процесор обчислює значення виразу



Програма для вирішення цієї задачі для одноадресного комп'ютера може бути наступною (табл. 1).

Як випливає з наведеної програми, операнд *а* вибирається з пам'яті 2 рази (команди 4 і 5), *b* – 3 рази (команди 2, 7 і 8). Крім того, потрібно додаткові звернення до пам'яті для запам'ятовування і вибору з пам'яті результатів проміжних обчислень (команди 3, 6, 9, 10).

Головним чинником, що обмежує швидкодію комп'ютера, є час циклу пам'яті. Очевидно, що принципово необхідні тільки звернення до пам'яті за даними вперше.

Таблиця 1. Приклад програми

|  |  |  |
| --- | --- | --- |
| **Номер команди** | **Команда** | **Коментар** |
| *1* |  |  |
| *2* |  |  |
| *3* |  | *– робоча комірка* |
| *4* |  |  |
| *5* |  |  |
| *6* |  | *– робоча комірка* |
| *7* |  |  |
| *8* |  |  |
| *9* |  |  |
| *10* |  |  |

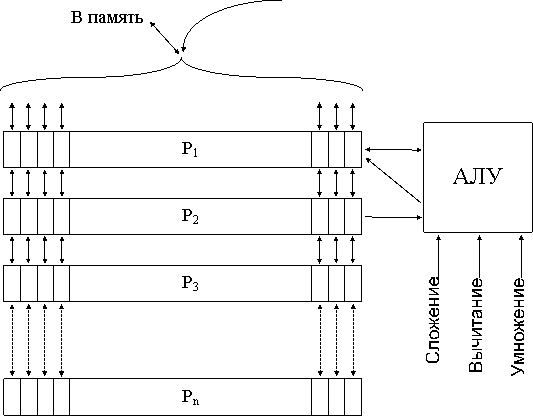
Вказані міркування одержали своє втілення у ряді логічних структур процесора. Одна з них – сопроцесор із стековою пам'яттю. Принцип її роботи пояснює схема, представлена на мал. 2.

Стекова пам'ять МП є набором з 8 регістрів, кожний з яких є 80-бітним/ 128-бітним і здатний зберігати одне значення у форматі ІЕЕЕ754. Назви регістрів ST(0),…, ST(7). Однойменні розряди регістрів зв”язані між собою ланцюгами зсуву. Тому весь набір регістрів може розглядатися як група регістрів однакової розрядності, зміст яких може бути переміщений як у стеку вгору або вниз.

Позначимо для скорочення викладок Pі+1 = ST(i), *i*=0,1,..,7. Тоді:

- Рух вниз: (P1)    P2, (P2)    P3, ..., а P1 заповнюється даними з головної пам'яті.

- Рух вгору: (Pn)    Pn‑1, (Pn‑1)    Pn‑2, а Pn заповнюється нулями.



Мал. 2. Стекова організація процесора

Регістри P1 і P2 пов'язані з АЛП, утворюючи два операнди для виконання арифметичної операції. Результат операції записується в P1. Отже, АЛП виконує операцію  .

Одночасно з виконанням арифметичної операції здійснюється просування операндів вгору, не зачіпаючи P1, тобто (P3)    P2, (P4)    P3 і т.д. Таким чином, в команді достатньо мати тільки поле, що визначає код операції. Тому комп'ютери із стековою пам'яттю називають безадресними. У той же час команди, здійснюючі виклик або запам'ятовування інформації з головної пам'яті, вимагають вказівки адреси операнда. Тому в ПК для реалізації процесорів із стековою пам'яттю використовуються команди змінної довжини.

Команди розташовуються в пам'яті у вигляді безперервного масиву складів незалежно від меж елементів пам'яті. Це дозволяє за один цикл звернення до пам'яті викликати декілька команд. Для ефективного використовування можливостей такої пам'яті в ПК вводяться спецкоманди:

· дублювання ~ (P1)    P2, (P2)    P3, ... і т. д., а (P1) залишається при цьому незмінним (double);

· реверсування ~ (P1)    P2, а (P2)    P1, що зручне для виконання деяких операцій (swap).

Розглянемо наведений вище приклад для нової ситуації (табл. 2):



Таблиця 2. Реалізація програми із стековою пам'яттю

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Команда** | **P1** | **P2** | **P3** | **P4** |
| *1* | *Виклик b* | *b* |  |  |  |
| *2* | *Дублювання* | *b* | *b* |  |  |
| *3* | *Виклик c* | *c* | *b* | *b* |  |
| *4* | *Додавання* | *b+c* | *b* |  |  |
| *5* | *Реверсування* | *b* | *b+c* |  |  |
| *6* | *Дублювання* | *b* | *b* | *b+c* |  |
| *7* | *Множення* | *b2* | *b+c* |  |  |
| *8* | *Виклик a* | *a* | *b2* | *b+c* |  |
| *9* | *Дублювання* | *a* | *a* | *b2* | *b+c* |
| *10* | *Множення* | *a2* | *b2* | *b+c* |  |
| *11* | *Додавання* | *a2+b2* | *b+c* |  |  |
| *12* | *Ділення* |  |  |  |  |

Головна перевага використання стекової (магазинної) пам'яті полягає у тому, що при переході до підпрограм (ПП) або у разі переривання немає необхідності в спеціальних діях по збереженню вмісту арифметичних регістрів в пам'яті. Нова програма може негайно почати роботу. При введенні в стекову пам'ять нової інформації дані, що відповідають попередній програмі, автоматично просуваються вниз. Вони повертаються назад, коли нова програма закінчить обчислення.

Разом з вказаними перевагами стекової пам'яті відзначимо також:

· зменшення кількості звернень до пам'яті;

· спрощення способу звернення до ПП і обробки переривань.

Недоліки стекової організації пам'яті:

· невелике число регістрів з швидким доступом;

· необхідність в додатковому обслуговуванні, щоб стежити за переповненням стекової пам'яті, бо число регістрів пам'яті кінцеве;

· застосування головним чином для вирішення наукових задач і у меншій мірі для систем обробки даних.

**Подання даних з плаваючою крапкою. Команди СОПРОЦЕСОРА**

**ІЕЕЕ 754 формат подання даних з плаваючою крапкою**

Припустимо, що для подання дійсних чисел використовуються **-байтні двійкові представлення. Це означає, що будь-яке дійсне число ** може бути подане у двійковому вигляді

**,

де *-* знак числа **, ,  - порядок числа ** за основою 2 (степінь, до якої необхідно піднести число 2, щоб виконувалась умова **), ** - деяка константа, величина якої обчислюється за формулою **,  - мантиса числа ** без уявної одиниці, ** та ** - відповідно кількості розрядів для представлення характеристики та мантиси числа **, ** .

Тип даних з плаваючою точкою:

float (одинарна точність) n=4, p=8, q=23, BIAS=127

long (подвійна точність) n=8, p=11, q=52, BIAS=1023

0 = 0000000… 00000000….

**= 1111111… 00000000….

NaN = 1111111… 010010100…. (мантиса не 0)

Приклад:

х=-0.4. s=1, k=2, m=1.6;  = m-1 = 0.6 = .1 0 0 1 (1 0 0 1)

0,6x2=1,2 ; 0,2x2=0,4 ; 0,4x2=0,8 ; 0,8x2=1,6 ; 0,6 …..

k=2+127=129 ->  = 10000001

Остаточно отримуємо

1 10000001 1001(1001)

Сопроцесор окрім регістрів стеку містить регістр управління (використовується при заокругленні результату) та індексний регістр, який дозволяє працювати напряму з обраним регістром, номер якого заноситься у цей регістр.

**Основні команди сопроцесора**

finit **–** ініціалізація регістрів сопроцесора

fadd (fsub, fmult, fdiv) – виконання операцій додавання, віднімання, множення та ділення над двома верхніми регістрами сопроцесора

fild (fld) – завантаження числа з фіксованою (плаваючою) крапкою в st(0)

fist (fst) – копіювання st(0) у пам’ять

fistp (fstp) – копіювання st(0) у пам’ять і виштовхування st(0) (pop st(0))

fxch st(i) – обмін значень st(0) та st(i)

fwait – очікування завершення операції

fnop – виконання неіснуючої операції

(в ЛР можно використовувати swap і double)

**Приклад** програми.

text segment

assume Cs:text,DS:data

begin: mov ax,data

mov ds,ax

finit

fild x

fild y

fadd ; можна fadd st,st[1]

fist z ; st(0)=z

finit

fld a

fadd b

fstp c ; st(0)=0

fwait

mov ah,4Ch

mov al,0

int 21h

text ends

data segment

a dd 2.

b dd 6.

c dd ?

x dw 1234

y dw 4320

z dw ?

data ends

end begin; Результат =5554**Запис програм та їх виконання на рівні процесора**

Програма - це лише список команд, що виконуються ЦП для вирішення певного завдання. Функціонування комп'ютера з технічної точки зору засновано на наявності або відсутності сигналів, що визначають адреси, дані та управління. Можна вважати, що схемотехнічно процесор складається з мільйонів невеликих перемикачів (електронних), які можуть знаходитися або у включеному стані, або у відключеному. Для представлення цих станів використовуються значення 1(включити) і 0 (вимкнути).

Написання та відлагодження програми на низькому рівні здійснюється з використанням мнемонічного коду операцій (наприклад 2+5 за допомогою інструкцій ADD,LOAD):

LOAD 2

LOAD 5

ADD

і є досить простим. Але слід врахувати, що ЦП розпізнає тільки 0 та 1, а не коди. Після написання програми з використанням мнемокоду (або коду, записаного мовою асемблер) необхідно виконати спеціальну програму-компілятор, яка перетворить мнемокод в послідовність 0 і 1. Використання мов високого рівня дозволяє записати оператори, які складають код програми. Оператори розміщуються в текстовому файлі, який називається вихідним файлом. Потім виконується спеціальна програма-транслятор або компілятор, яка і перетворює оператори в послідовності 0 і 1. Компілятор зберігає двійковий код у файлі, записаному у форматах типу .exe або .com.

Усередині комп'ютера електричні сигнали, забезпечують взаємодію пристроїв між собою, передаються через групи провідників, які називаються шинами. Наприклад, одна шина дозволяє ЦПУ взаємодіяти з платами розширення, а інша - забезпечувати зв'язок з відеокартою. Основна шина, (магістраль) називається системною, зв’язує ЦП з рештою елементів набору материнської плати (chipset). Системну шину називають також фронтальною шиною FSB (front side bus), щоб координувати сигнали, що передаються по її провідниках, системна шина діє з фіксованою частотою тактів. Наприклад, ЦП поміщає на шину дані, які необхідно зберегти в пам'яті. Дані (елементи сигналу) зберігаються на шині певний час, в даному випадку достатньо довго, щоб контроллер пам'яті зміг прийняти і помістити їх в ОП. У міру збільшення швидкодії процесорів швидкість системній шині збільшується. На кожному такті процесор виконує одну операцію. На жаль, якщо команда потребує виконання операції з пам'яттю, ЦП повинен припиняти роботу та чекати завершення операції передачі даних за системною шиною. Це суттєво знижує швидкодію ПК. Повільніша системна шина гальмує роботу більш швидкісного ЦП. Т.ч. вона є вузьким місцем продуктивності ПК.

Програми оперують адресами, запис яких визначається режимом роботи МП. Для звертання до фізичної адреси в ОП, необхідно перетворення логічної адреси у фізичну. Таке перетворення відбувається за типовою схемою перед розміщенням адреси на адресній шині.

Розглянемо різні варіанти схем перетворення.

І. Реальний режим роботи (Real Mode)

Логічна одреса подається парою регістрів (СР):(Зсув) (наприклад, адреса коду (CS):(IP)). В рамках сегментованої моделі адресації для програми пам'ять представляється групою незалежних адресних блоків, які мають назву сегментів, розмір яких 64К.

Ширина адресної шини 20 біт. Тому схема перетворення має вигляд:

лінійна адреса = (СР)\*16+(Зсув)

фізична адреса = лінійна адреса

Отримане значення виставляється на адресну шину і визначає фізичне місце розташування сукупності байтів коду, даних або стеку.

ІІ. Захищений 32-розрядний режим (Protected Mode, див. оглядовий матеріал <http://mf.grsu.by/UchProc/livak/po/lections/lec9.htm>)

Ширина адресної шини 32/36/48 біт.

Основним *режимом роботи* МП є захищений режим. Ключові особливості захищеного режиму: віртуальний адресний простір, захист і багатозадачність. МП може бути переведений в захищений режим установкою біта 0 (Protect Enable, РЕ) в регістрі CR0. Повернутися в режим реальної адреси МП може по сигналу RESET або скиданням біта PE.

У захищеному режимі програма оперує з адресами, які можуть відноситися до фізично відсутніх елементів пам'яті, тому такий адресний простір називається **віртуальним**. Розмір віртуального адресного простору програми може перевищувати місткість фізичної пам'яті і досягати 64Тбайт.

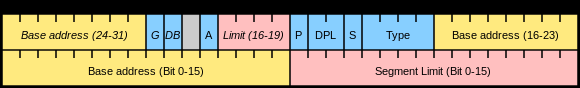
Перетворення логічної адреси у фізичний відбувається в два етапи: спочатку блок управління сегментами виконує трансляцію адреси відповідно до сегментованої моделі пам'яті, отримуючи 32-бітову лінійну адресу, а потім блок *сторінкового перетворення* виконує розбиття на сторінки, перетворюючи 32-бітову лінійну адресу в 32-бітову або 36-бітову (P6) фізичну. МП не передбачає механізмів заборони сегментації; з іншого боку, сторінкова трансляція є опціональним механізмом і може використовуватися або не використовуватися залежно від особливостей операційної системи.

Кожна програма має набір фрагментів ОП. При застосуванні фрагментарної моделі для адресації байта пам'яті програма також використовує логічну адресу, що складається з пари значень, поданих у сегментному регістрі та регістрі зсуву. Але відповідні регістри у цьому випадку містять селектор фрагменту і величину зсуву. Селектор визначає певний фрагмент, а зсув указує на конкретний байт в адресному просторі вибраного фрагменту (на основі значення регістра загального призначення).

З кожним фрагментом пов'язана особлива структура, що зберігає інформацію про нього: **дескриптор**. Дескриптор - це 8-байтна одиниця опису фрагменту даних, що розпізнається менеджером управління пам'яттю в захищеному режимі та зберігається в дескрипторній таблиці. Дескриптор фрагменту містить базову адресу фрагменту, границю фрагменту і права доступу до фрагменту. У захищеному режимі сегменти можуть починатися з будь-якої лінійної адреси (який називається базовою адресою фрагменту) і мати будь-який розмір в межах обсягу ОП.

0

**Мал. 1.**  Структура дескриптора фрагмента ОП



(https://lawbooks.news/telekommunikatsionnyie-sistemyi-kompyuternyie/struktura-deskriptora-segmenta-60013.html)

Значення довжини фрагменту (сегменту) ОП визначається 2 ділянками дескриптора, сумарна довжина яких рівна 20 біт. Довжина фрагменту може вимірюватися в байтах або сторінках, довжина яких дорівнює 4 КБ/2 МБ/ 4 МБ. Вибір одиниці вимірювання довжини фрагменту залежить від значення біту гранулярності G (лат. granulum — зернятко). При значенні G=0 довжина фрагменту вимірюється в байтах, при цьому 20-бітове поле забезпечує можливість задання довжини фрагменту в межах від 1 байта до 1 Мбайт. При значенні G = 1 довжина фрагменту вимірюється в сторінках. В цьому випадку вона може змінюватися в межах від однієї сторінки (4 Кбайт) до 1М сторінок (1М • 4 Кбайт = 4 Гбайт).

S=0 – системний фрагмент, S=1 – фрагмент коду або даних.

DPL – descriptor privilege level.

У зв'язку з необхідністю підтримувати сумісність з попередніми 16-бітовими моделями в дескрипторі фрагменту передбачений біт розрядності D|B (від Dimension - вимірювання), значення якого визначає розрядність операндів і адрес, що використовуються для фрагменту. Якщо значення D|B = 0, то використовується 16-бітова, а при D|B = 1 - 32-бітова розрядність адрес і операндів.

4 біти типу фрагменту Type = (І|x|x|x)

На відміну від реального режиму, в захищеному режимі можливі тільки два принципово різних типи фрагментів: сегмент коду і сегмент даних. Тип фрагменту визначається значенням біта призначення сегменту I (від Intending — призначення). Якщо значення I = 0, то дескриптор описує сегмент даних; якщо ж I = 1, то це - сегмент коду.

**Дескрипторні таблиці** - це масиви пам'яті змінної довжини, які складаються з 8-байтних елементів, дескрипторів. Дескрипторна таблиця може мати довжину від 8 байт до 64 Кбайт і в кожній таблиці може бути до 8192 дескрипторів. Існують дві обов'язкові дескрипторні таблиці - глобальна дескрипторна таблиця (Global Descriptor Table - GDT) і дескрипторна таблиця переривання (Interrupt Descriptor Table - IDT), а також множина (аж до 8191) необов'язкових локальних дескрипторних таблиць (Local Descriptor Table - LDT), з яких в кожен момент часу процесору доступна лише одна. Розташування дескрипторних таблиць визначається регістрами процесора GDTR, IDTR, LDTR.

GDT містить дескриптори, доступні всім задачам в системі. GDT може містити дескриптори будь-яких типів: і дескриптори фрагментів, і системні дескриптори (окрім шлюзів переривань і пасток). Перший елемент GDT (з нульовим індексом) не використовується. Йому відповідає нуль-селектор, що позначає "порожній" покажчик.

LDT забезпечують спосіб ізоляції сегментів програми і даних задачі, що виконується, від інших задач. LDT пов'язана з конкретною задачею і може містити тільки дескриптори фрагментів, шлюзи викликів і шлюзи задач.

Фрагмент не може бути доступний задачі, якщо його дескриптор не існує ні в поточній таблиці LDT, ні в таблиці GDT. Використання двох дескрипторних таблиць дозволяє, з одного боку, ізолювати і захищати сегменти задач, що виконуються, а з іншої - дозволяє розділяти глобальні дані і код між різними задачами.

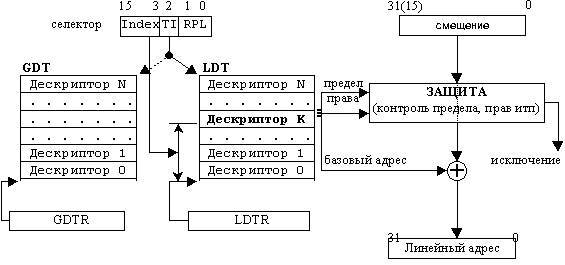
IDT містить тільки шлюзи завдань, шлюзи переривань або шлюзи пасток.

Для обчислення лінійної адреси МП виконує наступні дії (рис. 2):

1. МП використовує селектор фрагменту для знаходження дескриптора сегменту. Селектор містить індекс дескриптора в дескрипторній таблиці (Index), біт TI, що визначає, до якої дескрипторної таблиці проводиться звернення (LDT або GDT), а також права доступу до сегменту (RPL). Якщо селектор зберігається в сегментному регістрі, то звернення до дескрипторних таблиць відбувається тільки при завантаженні селектора в сегментний регістр, оскільки кожен сегментний регістр зберігає відповідний дескриптор в программно-недоступному ("тіньовому") регістрі-кеші.
2. МП аналізує дескриптор сегменту, контролюючи права доступу (фрагмент доступний з поточного *рівня привілеїв*) і межу сегменту (зсув не перевищує межу);
3. МП додає зсув до базової адреси сегменту і отримує лінійну адресу.

Якщо сторінкова трансляція відключена, то сформована лінійна адреса вважається фізичною і виставляється на шину процесора для виконання циклу читання або запису пам'яті.

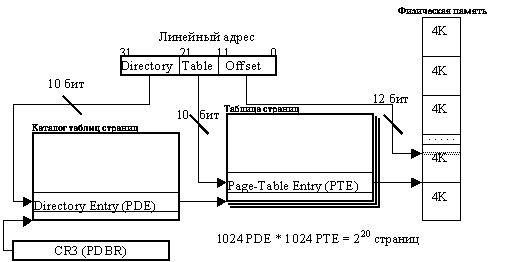
**Мал. 2.**  Схема перетворення логічної адреси в лінійну.



Механізм *сегментації у Protected Mode (фрагментації)* забезпечує захист, але він не дуже зручний для реалізації віртуальної пам'яті (підкачки). У дескрипторі фрагменту є біт присутності, по ньому процесор визначає, чи знаходиться даний фрагмент у фізичній пам'яті або на зовнішньому пристрої, де він запам'ятовується (на вінчестері). У останньому випадку генерується виключення #11, обробник якого може завантажити фрагмент в пам'ять.

Незручність полягає в тому, що різні фрагменти можуть мати різну довжину. Цього можна уникнути, якщо механізм підкачки реалізовувати на основі сторінкового перетворення. Особливістю цього перетворення є те, що процесор в цьому випадку оперує з блоками фізичної пам'яті рівної довжини (4 Кбайт) - сторінками. Сторінки не мають безпосереднього відношення до логічної структури програми. Крім того, в МП підрежиму P6 сторінкова трансляція забезпечує 36-бітову фізичну адресацію пам'яті (64 Гбайт). Сторінкове перетворення діє тільки в захищеному режимі і включається установкою в 1 біта PG в регістрі CR0.

У *сторінковому перетворенні* беруть участь два типи структур: каталоги таблиць (Page Directory) і таблиці сторінок (Page Table). Ці структури складаються з 1024 32-бітових елементів. Елементи містять старші 20 біт фізичної адреси об'єктів, що адресуються. Елементи таблиці сторінок (Page Table Entry - PTE) адресують сторінки, а елементи каталога таблиць (Page Directory Entry - PDE) адресують таблиці сторінок. Старші 20 біт фізичної адреси каталога таблиць зберігаються в регістрі CR3 (Page Directory Base Register - PDBR) (це єдиний регістр процесора, який містить фізичну адресу пам'яті). Всі структури вирівняні на границі сторінки.

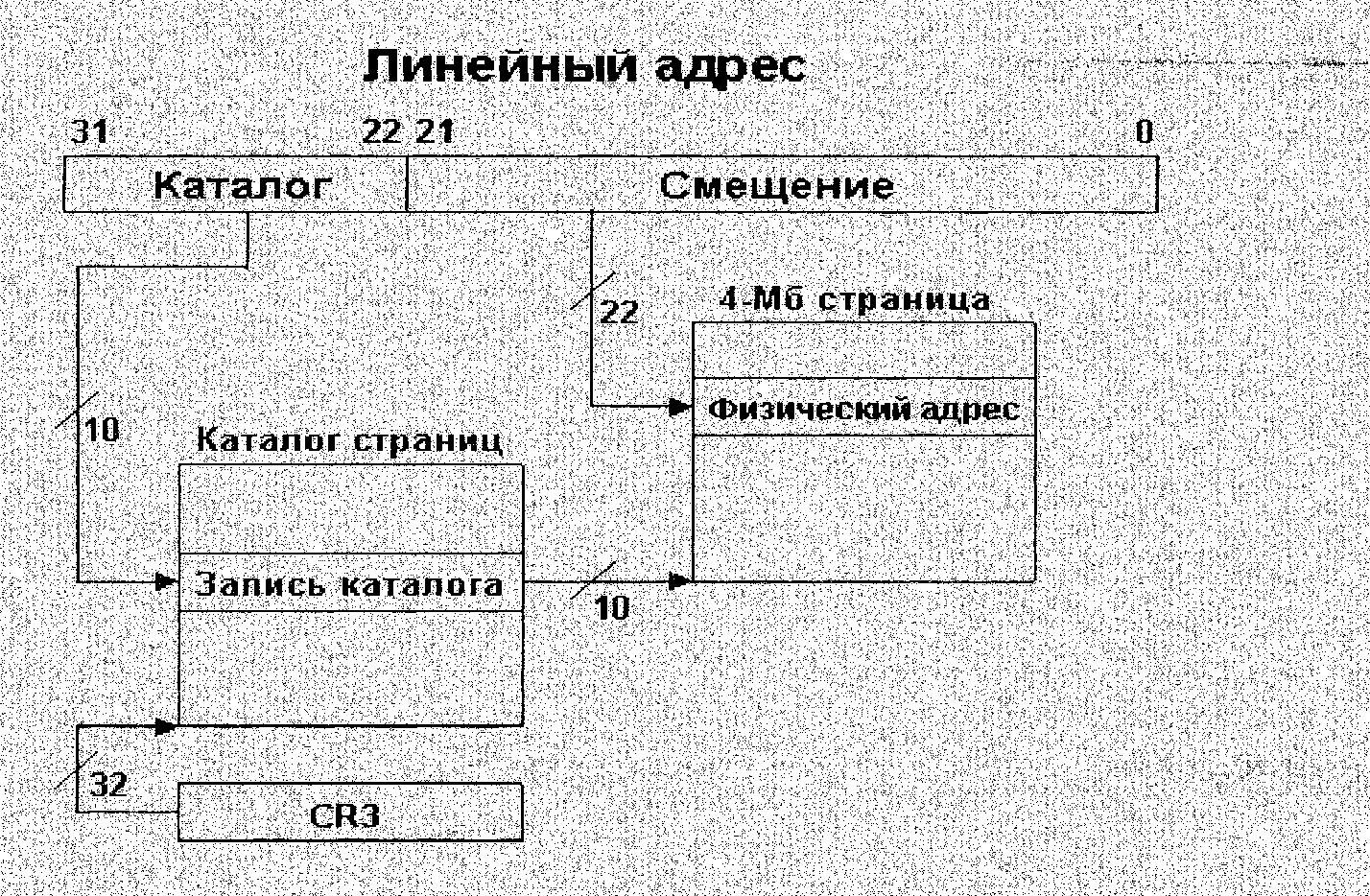


**Мал. 3.**  Стандартна 2-рівнева схема сторінкової трансляції

В процесі сторінкової трансляції адрес отримана лінійна адреса розбивається на три частини. Старші десять біт (Directory) лінійної адреси є індексом елементу з каталога таблиць. По цьому елементу визначається фізична адреса таблиці сторінок. Біти 21-12 (Table) лінійної адреси вибирають елемент з цієї таблиці сторінок. Вибраний елемент визначає фізичну адресу сторінки. Молодші 12 біт (Offset) лінійної адреси визначають зсув від початку сторінки.

Сторінки починаються на границі 4 Кбайт областей пам'яті, тому молодші 12 біт адреси сторінки завжди рівні нулю. У каталозі таблиць елементи зберігають фізичні адреси таблиць сторінок. У таблиці сторінок елементи зберігають фізичні адреси самих сторінок.

У МП реалізувано також іншу можливість - розширення розміру сторінок (Page Size Extension - PSE). PSE дозволяє використовувати сторінки розміром 4 Мбайт і однорівневий механізм сторінкової трансляції. У підрежимі P6 шина адреси була розширена до 36 біт. Відповідно, процесори Pentium Pro, Pentium II, Pentium III і пізніші (сучасні) здатні адресувати до 64 Гбайт фізичної пам'яті. Подальше розширення ширини адресної шини дозволило розширити адресний простір ще більше. Ця можливість називається розширенням фізичної адреси (Physical Address Extension - PAE) і доступна тільки при використанні сторінкової трансляції. Таким чином, застосування PSE і PAE впливає на *сторінкове перетворення.*

**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Таблиця 1. Особливості сторінкового перетворення в PSE і PAE** | | | | | |
| **CR0.PG** | **CR4.PAE** | **CR4.PSE** | **PDE.PS** | **Розмір сторінки** | **Розрядність фіз. Адреси** |
| 0 | X | X | x | - | 32 бит |
| 1 | 0 | 0 | x | 4 Кб | 32 бит |
| 1 | 0 | 1 | 0 | 4 Кб | 32 бит |
| 1 | 0 | 1 | 1 | 4 Мб | 32 бит |
| 1 | 1 | X | 0 | 4 Кб | 32 бит |
| 1 | 1 | X | 1 | 2 Мб | 32 бит |

Вбудовані засоби перемикання завдань забезпечують реалізацію *багатозадачності* в захищеному режимі. **Задача** - це "одиниця вимірювання" завдань для процесора, яке процесор може виконувати, припиняти і здійснювати над ним диспетчеризацію. Як задача може виконуватися прикладна програма, сервіс операційної системи, ядро операційної системи, обробник переривання або виключення і т.п. У захищеному режимі архітектура надається механізм для збереження стану задачі та перемикання з однієї задачі на іншу.

Середовище задачі складається з вмісту регістрів МП і всього коду з даними в просторі пам'яті. МП здатний швидко перемикатися з одного середовища виконання в інше, імітуючи паралельне виконання декількох задач. Для деяких задач може емулюватися управління пам'яттю, як у МП 8086. Такий режим виконання задачі називається режимом віртуального 8086 (Virtual 8086 Mode). Про перебування задачі в даному стані сигналізує біт VM в регістрі флагів. При цьому задачі віртуального МП 8086 ізольовані і захищені, як від один одного, так і від звичайних завдань захищеного режиму.

Задачу складають два компоненти: адресний простір задачі та сегмент стану завдання (Task State Segment - TSS). У адресний простір задачі входять доступні для неї фрагменти коду, даних і стека. Сегмент стану завдання TSS зберігає стан регістрів (контекст) процесора:

* стан сегментних регістрів (селектори сегментів, що формують адресний простір задачі);
* стан регістрів загального призначення;
* стан регістра флагів (EFLAGS);
* значення регістра-лічильника для визначення наступної команди (EIP);
* значення регістра CR3 (PDBR);
* значення регістра LDTR.

У багатозадачних системах TSS надає механізм для вкладення задач. Кожна задача ідентифікується селектором відповідного їй TSS. Цей селектор завантажується в регістр завдання (Task Register - TR) при перемиканні на задачу. У тіньову частину регістра завантажуються базова адреса, границя і атрибути TSS. Операційна система може передбачати для кожної задачі свій лінійний адресний простір (свій набір сторінок для *сторінкового перетворення*), тоді при перемиканні завдання завантажується і регістр CR3 (PDBR), що зберігає адресу каталогу таблиць для сторінкового перетворення.